CAS Computational and Applied Science

**Volume 1, Issue 1**

**2026**

# COMPUTATIONAL AND APPLIED SCIENCE

International Peer-Reviewed Open Access Journal

Independent International Scientific Journal

Tbilisi

2026

**Computational and Applied Science (CAS Journal)** is an independent, international, peer-reviewed open access scientific journal dedicated to the development and dissemination of innovative theoretical, methodological, and applied research across a broad range of computational and applied disciplines. The journal particularly emphasizes research in computational and applied mathematics, scientific and numerical computing, algorithms and data structures, artificial intelligence and machine learning, data science and analytics, computational modeling and simulation, computer science and information technologies, digital transformation, business and decision analytics, educational technologies, and other related interdisciplinary fields.The journal provides a platform for the dissemination of theoretical, methodological, and applied research that contributes to the advancement of computational and applied sciences and supports scholarly dialogue at the international level.

All manuscripts submitted to the journal undergo a rigorous peer-review process to ensure academic integrity, originality, and scientific relevance.

The authors are solely responsible for the content, accuracy, and originality of their articles. Opinions expressed in published papers do not necessarily reflect the views of the editorial board or the publisher.



---

**Journal Title:** Computational and Applied Science
**Abbreviation:** CAS Journal
**Publication Type:** Open Access
**Peer Review:** Double-blind peer review
**Language:** English
**Publisher:** CAS Journal
**Country of Publication:** Georgia

**ISSN 3088-4152 (Online)**

# Editorial Team

# Contents

Data Processing Pipeline for Course-Level Outcome Analytics in Higher Education: Time-Based and Clustering-Based Hybrid Approaches
Besiki Tabatadze, Sophio Khundaze ........................................................................................5

A Cloud-Based Virtual Sensor Approach for Intelligent Marine Water Quality Monitoring
Nino Topuria ..........................................................................................................................28

Performance Optimization of CI/CD Pipelines through Redundant Operation Elimination
Saba Gotsiridze, Giorgi Kuchava, Teimuraz Sturua, Ana Khinchikashvili, Ekaterine Somkhishvili, Ioseb Kartvelishvili, Maka Mantskava, Shalva Chokhonelidze, Nana Momtselidze .......................37

Leadership Dynamics as Determinants of Project Success
Nino Lolashvili, Anastasia Bajiashvili .....................................................................................56

Performance and Efficiency Evaluation of GitOps-Based Deployment Workflows
Giorgi Tseradze, Giorgi Kuchava, Teimuraz Sturua, Ana Khinchikashvili, Ekaterine Somkhishvili, Ioseb Kartvelishvili, Maka Mantskava, Shalva Chokhonelidze, Nana Momtselidze .......................66

AI-Assisted Programming in Practice Conceptual Foundations and Data-Informed Observations
Besiki Tabatadze ......................................................................................................................84

Use of Artificial Intelligence in IT Project Management
Anastasia Bajiashvili, Nino Lolashvili ...................................................................................101

When Digital Attacks Turn Physical: National Security in the Age of Cyber Threats
Akaki Shekeladze ..................................................................................................................110

PM Methodology Implementation in IT Management
Anastasia Bajiashvili, Nino Lolashvili ...................................................................................121

Application of Shortest Path Algorithms in Telecommunication Channel Coding: A Comparative Study
Tamar Kamkhadze .................................................................................................................131

Enhancing CI/CD Security with Artificial Intelligence: State of the Art, Challenges, and Integrated Approaches
Iveri Jajanidze .......................................................................................................................138

# Data Processing Pipeline for Course-Level Outcome Analytics in Higher Education: Time-Based and Clustering-Based Hybrid Approaches

Besiki Tabatadze

*PhD (Applied Mathematics), Professor, European University, Georgian American University, Tbilisi, Georgia.*

***Email:*** *tabatadze.besik@eu.edu.ge*

Sophio Khundaze

*PhD, European University, Tbilisi, Georgia.*

***Email:*** *skhundadze@eu.edu.ge*

## Abstract

One of the key tasks of quality assurance in higher education is the systematic monitoring of course-level outcomes and the adoption of appropriate decisions based on the analysis of these results. Although quality assurance standards and guidelines place strong emphasis on data-informed monitoring and the periodic review of programmes, practical analytical approaches applied at the course level often remain diverse and unevenly formalized. This situation is largely influenced by the institutional autonomy granted to higher education institutions, which results in diverse practices across institutions.

The aim of this paper is to review and comparatively analyze contemporary data-informed and analytics-based approaches to course-level quality assurance. The study is based on a systematic examination of existing approaches, including the analysis of aggregated assessment data, the use of learning analytics and educational data mining, as well as the role of explainable analytics in supporting academic and managerial decision making. The review is complemented by a practice-oriented example based on real data, which examines the longitudinal analysis of assessment outcomes for the same course over multiple academic years.

The findings indicate that the analysis of grade distributions and performance dynamics can be effectively used to identify the need for improvements in course content, teaching methods,

allocation of instructional time and assessment systems. The paper highlights the specific characteristics of different approaches, their strengths and limitations, and emphasizes the importance of using modern technologies to support course-level quality assurance processes.

**Keywords:** Quality Assurance in Higher Education, Learning Analytics, Data Processing, Hierarchical Clustering, Unsupervised Learning, Grade Distribution Analysis

## Introduction

Quality assurance has long been recognized as one of the central issues in higher education and encompasses multiple dimensions related to teaching, learning, assessment, institutional effectiveness, and other processes taking place within higher education institutions (HEIs). The definition of quality assurance in HEIs emphasizes its multidimensional and context-dependent nature, which includes the transformation and enhancement of teaching processes in order to ensure learning outcomes (Harvey & Green, 1993).

Within this process, quality assurance at the course level plays a critical role, as the objectives of individual courses are aligned with the objectives of the corresponding academic program, which in turn reflect institutional goals, in accordance with the principles of outcomes-based education (Biggs et al., 2022). Thus, the learning outcomes achieved by students within individual courses determine the attainment of program-level outcomes, which consequently contributes positively to the overall quality development of the HEI. It can be argued that outcomes achieved at the course level represent the primary interface between institutional objectives and students' learning experiences.

The Standards and Guidelines for Quality Assurance in the European Higher Education Area (ESG) place strong emphasis on the continuous monitoring, periodic review, and evidence-based improvement of study programmes and courses (European Association for Quality Assurance in Higher Education [ENQA], 2015). At the same time, ESG does not prescribe specific analytical methods for evaluating course-level outcomes, which has led to the emergence of diverse practices and approaches to the analysis of course results across higher education institutions. These approaches differ in terms of the types of data used, the depth of analysis, and their intended purposes and modes of interpretation, thereby creating a clear need for their systematic examination and comparative analysis.

Proper alignment between course learning outcomes, subject content, teaching activities, and assessment methods is a fundamental prerequisite for quality learning (Biggs et al., 2022). Although the aggregated assessment results (grades) obtained by students in a course do not fully capture the quality of teaching and learning, the thoughtful and contextualized analysis of these outcomes plays an important role in supporting broader and continuous quality assurance processes (Boretz, 2004). Moreover, the systematic analysis of assessment results can inform meaningful adjustments in course design and implementation.

Data-informed decision making has gained increasing importance in higher education governance and curriculum development. Research indicates that the effective use of data can contribute to improvements in teaching and learning processes, provided that data are interpreted within appropriate pedagogical and organizational contexts (Schildkamp & Kuiper, 2010; Webber & Zheng, 2020). Advances in learning analytics, educational data mining, and related technologies have further expanded the possibilities for analyzing student achievement and learning processes (Shafiq et al., 2022). In parallel, the digitalization of educational processes and the development of data processing and analytical technologies have significantly broadened opportunities for enhancing the quality of teaching and learning (Martin et al., 2020).

At the course level, observing and analyzing the numerical distribution of students' assessment outcomes can serve as an important source of information for evaluating the effectiveness of individual courses (Rexwinkel et al., 2013). Recent practice-oriented studies further demonstrate how the use of data and data visualization techniques within learning management systems can enhance the interpretability of course-level outcomes and support informed academic decision making (Tabatadze, 2023; Tabatadze & Sokhadze, 2023).

In this context, the aim of the present paper is to review and comparatively analyze contemporary data-informed and analytics-based approaches to course-level quality assurance in higher education. The study seeks to demonstrate how explainable and accessible analytical methods can be employed to support effective course review processes and continuous quality improvement.

## Review of Assessment-Based Approaches to Course-Level Outcome Analysis

The present study is based on aggregated assessment data, specifically the quantitative distribution of final grades (A - F) awarded to students in individual courses. The paper examines the types of analytical approaches and evaluative interpretations that can be applied on the basis of course-level outcomes after course completion and compares these approaches with one another. Although this type of data does not allow for a detailed analysis of individual students' learning trajectories, it nevertheless represents an important source for identifying systemic patterns and potential issues at the course level (Schildkamp & Kuiper, 2010; Rexwinkel et al., 2013). Such data can be conceptualized as outcome-level distribution data, which are widely used in quality assurance practices to evaluate the effectiveness of individual courses (Boretz, 2004; Rexwinkel et al., 2013).

Based on the assessment outcomes achieved by students within a specific course, a variety of approaches can be employed to analyze course-level results in support of quality assurance processes. These approaches differ in terms of analytical depth, temporal scope, and interpretative focus. Some methods rely on descriptive statistical analysis of grade distributions, while others adopt comparative or longitudinal approaches, enabling the identification of trends, changes, and potential anomalies across different academic years and student cohorts (Schildkamp & Kuiper, 2010; Rexwinkel et al., 2013). In addition, so-called hybrid approaches are increasingly used, integrating elements of learning analytics, data visualization, and rule-based interpretation in order to enhance the explainability and practical usability of analytical results for academic staff and decision makers (Martin et al., 2020; Tabatadze, 2023; Tabatadze & Sokhadze, 2023).

Within the context of course-level outcome analysis, several widely used approaches can be identified. These include descriptive statistical analysis of grade distributions, comparative analysis across courses or student cohorts, longitudinal analysis of course outcomes over multiple academic years, threshold- and rule-based evaluation approaches, as well as analytics-driven methods that employ statistical techniques, data visualization tools, and contemporary technological capabilities, including machine learning models (Schildkamp & Kuiper, 2010; Rexwinkel et al., 2013; Martin et al., 2020; Tabatadze, 2023; Tabatadze & Sokhadze, 2023). These approaches differ in terms of analytical depth, interpretative scope, and their practical applicability to course-level quality assurance processes.

*Descriptive Grade Distribution Approaches*

Course-level outcome analysis frequently relies on descriptive representations of assessment results, such as the distribution of grades on an A–F scale and pass/fail indicators. Such outcome-level distribution data are widely used as an analytical tool that enables the timely interpretation of the overall performance profile of a course and the identification of potentially critical system-wide signals. It is noteworthy that, in institutional practice, grade distribution analysis is often employed as an initial stage of quality assurance processes, particularly when the objective is to monitor course outcomes and to support the early detection of problematic trends (Boretz, 2004; Rexwinkel et al., 2013; Schildkamp & Kuiper, 2010).

*Threshold- and Indicator-Based Quality Signals*

In practice, threshold- and indicator-based approaches are widely applied to identify potential quality issues at the course level. Studies and quality assurance documentation frequently refer to indicators such as a high proportion of low grades (e.g., F) or the cumulative share of weak outcomes (D+E+F), which may signal problems related to assessment design, student workload, or teaching methods. At the same time, an excessive concentration of high grades (A+B) is often discussed as a potential indicator of grade inflation. Importantly, such thresholds are not interpreted as rigid rules; rather, they are commonly used in quality assurance practice as heuristic signals that require further analysis and contextual interpretation (Boretz, 2004; Rexwinkel et al., 2013; Schildkamp & Kuiper, 2010).

*Longitudinal and Trend-Based Approaches*

In course-level quality evaluation, particular importance is attributed to longitudinal and trend-based approaches that rely on the analysis of outcomes from the same course across multiple academic years. The use of multi-year data makes it possible to identify stable trends, recurring deviations, and so-called structural problems that often remain invisible when analysis is based on a single cohort or academic year. In the literature, such dynamics are frequently described in terms of a critical trajectory, indicating systematic deterioration or improvement in course outcomes over a defined period of time (Rexwinkel et al., 2013; Schildkamp & Kuiper, 2010).

*Statistical Significance-Based Approaches*

In some cases, course-level outcome analysis is grounded in methods based on statistical significance testing. Within this context, techniques such as the $\chi^2$ test are employed to assess differences between grade distributions, while effect size indicators are used to capture the practical significance of observed changes. Such approaches are particularly important when the objective is to distinguish random fluctuations from genuine, systematic changes in course outcomes and to strengthen the scientific robustness and credibility of analytical conclusions (Rexwinkel et al., 2013; Schildkamp & Kuiper, 2010).

*Analytics- and Rule-Based Approaches (Explainable)*

In recent years, analytics- and rule-based approaches grounded in the technological processing of educational data have increasingly been applied to enhance the transparency and interpretability of course-level outcome analysis. Within this context, machine learning models are commonly used for clustering tasks, enabling courses to be grouped according to similarities in outcome distributions and facilitating the identification of shared patterns across courses. At the same time, the literature emphasizes that more complex models, such as deep learning algorithms, despite their strong predictive capabilities, are less practical for course-level quality assurance due to their limited interpretability and low level of explainability. Consequently, greater emphasis is placed on simpler and more interpretable approaches, including unsupervised clustering techniques and rule-based decision logic, which are more closely aligned with the practical requirements of quality assurance processes and support explainable, data-informed decision making (Schildkamp & Kuiper, 2010; Martin et al., 2020; Shafiq et al., 2022).

*Hybrid Approaches in Course-Level Quality Assurance*

In quality assurance practice, course-level outcome analysis is rarely based on a single, isolated analytical approach. Instead, so-called hybrid approaches are commonly applied, combining elements of two or more complementary methods selected in accordance with specific analytical objectives and contextual conditions. In practice, such combinations may include, for example, descriptive analysis of grade distributions together with threshold-based indicators, or longitudinal trend analysis combined with tests of statistical significance, while in some cases a broader analytical framework may be employed. This selective integration enables stronger interpretation of results, reduces the impact of limitations inherent in individual methods, and supports more balanced and context-sensitive conclusions based on outcome-level data. Consequently, hybrid approaches represent a flexible and practice-oriented instrument for course-level quality assurance processes.

The approaches discussed above illustrate the diversity of analytical perspectives that can be applied to course-level outcome data. While each method emphasizes different aspects of course performance, their combined use in practice enables a more comprehensive and nuanced interpretation of quality-related signals. Of particular interest, therefore, is not the comparison of individual analytical approaches in isolation, but the examination of selected combinations of these approaches and the outcomes they produce. From a research perspective, it is especially important to assess how the use of two or three complementary methods in combination influences the interpretation, robustness, and practical usability of conclusions drawn from course-level quality assurance analyses.

## Two Hybrid Analytical Approaches

An important area of research concerns the examination of different hybrid analytical approaches and their comparative analysis within the context of course outcome–based quality assurance. Within the framework of the present paper, two hybrid approaches are examined, which are deliberately formulated to reflect two distinct yet complementary paradigms of data-informed quality assurance. The first approach is based on observing students' achieved outcomes over time and focuses on identifying the stability of course results and sustainable trends across multiple academic years. The second approach, in contrast to the first, concentrates on the analysis of single-semester data and employs analytics-based machine learning techniques, specifically clustering, to identify similarities and differences in outcomes across courses.

These two hybrid approaches are not viewed as mutually exclusive or competing alternatives. Rather, they are presented as analytically distinct frameworks that support quality assurance processes in different contexts. The time-based approach is particularly effective for identifying structural and recurring issues, whereas the clustering-based analytical approach is more suitable for rapid, data-driven interpretation in situations where only single-semester data are available.

Within the scope of the study, each hybrid approach is examined separately, taking into account its constituent analytical elements, underlying logic of application, and interpretative capacity. This is followed by a comparative analysis focusing on relevant criteria applied in quality assurance processes. This comparison provides the foundation for the subsequent empirical analysis, in which both hybrid approaches are applied to real-world data.

### *Time-Based Hybrid Analytical Approach*

The time-based hybrid analytical approach is grounded in the observation of course outcomes over time and the analysis of data accumulated across multiple academic years. This approach constitutes a combination of descriptive analysis of grade distributions, threshold- and indicator-based evaluation methods, and longitudinal trend analysis, thereby forming a hybrid analytical framework. The primary objective of this approach is not the evaluation of a single semester or cohort, but the identification of dynamics, stability, and recurring patterns in course outcomes over time. Such a perspective is particularly important in quality assurance processes, where single-point data often fail to capture systemic issues or long-term changes.

This hybrid analytical approach integrates the following analytical components:

- Descriptive analysis of grade distributions;
- Threshold- and indicator-based evaluation approaches;
- Longitudinal and trend-based analytical approaches.

At the first stage, descriptive analysis of grade distributions is applied to provide an overall profile of course outcomes for each academic year. This analysis is complemented by threshold- and indicator-based evaluation, which enables the identification of potentially critical signals, such as changes in the proportion of low or high grades over time.

At the second stage, longitudinal and trend-based analysis is conducted, in which course outcomes are examined as a time series. This makes it possible to identify stable trends, recurring deviations, and so-called structural problems that may indicate imbalances in course content, teaching methods, or assessment design. Particular attention is given to dynamics described as a critical trajectory, referring to a consistent deterioration or improvement in outcomes over a defined period.

The main strength of this hybrid approach lies in its ability to support contextualized and robust interpretation of results. The use of multi-year data reduces the influence of random fluctuations and enhances the reliability of analytical conclusions. At the same time, one of its limitations is its reliance on historical data, which makes it less effective in cases where a course is newly introduced or has undergone recent substantive changes.

Overall, the time-based hybrid analytical approach represents an effective instrument for the systematic evaluation of course outcomes and is particularly well suited to quality assurance processes oriented toward long-term development and the planning of structural improvements.

*Clustering-Based Hybrid Analytical Approach*

The clustering-based hybrid analytical approach focuses on the structural analysis of course outcomes within a single academic period and aims to identify similarities and differences between courses through outcome-based clustering of data. In contrast to time-based approaches, this analytical framework is designed for contexts in which longitudinal data are not available or where the primary objective is to support rapid and comparative interpretation of course-level results. The approach integrates descriptive and rule-based evaluation with analytics-oriented methods, resulting in a clustering-centered hybrid analytical configuration.

This hybrid analytical approach integrates the following analytical components:

- descriptive analysis of grade distributions;
- threshold- and indicator-based evaluation approaches;
- analytics- and clustering-based analytical methods.

At the initial stage, descriptive analysis of grade distributions is applied, whereby outcome profiles for each course are constructed using A–F grade scales, pass/fail indicators, and other relevant metrics. This stage ensures a structured representation of outcome-level data and establishes a common analytical basis for subsequent comparison.

At the next stage, threshold- and indicator-based evaluation is conducted, enabling the identification of potentially critical quality signals, such as a high proportion of weak outcomes or an excessive concentration of high grades. These indicators are not interpreted as rigid rules, but rather are used as heuristic reference points that guide further interpretation.

The central element of this hybrid approach is clustering based on machine learning techniques, which aims to group courses according to similarities in their outcome distributions. The application of unsupervised clustering methods makes it possible to reveal latent structural patterns across courses and to form groups with comparable performance profiles. At the same time, particular emphasis is placed on explainable clustering solutions, which ensure the transparency and practical usability of analytical results for academic staff and quality assurance units.

The main strength of the clustering-based hybrid analytical approach lies in its ability to support rapid, data-informed interpretation in contexts where only single-semester data are available. The combination of descriptive analysis, indicator-based evaluation, and explainable machine learning facilitates the identification of structural similarities between courses and the detection of potentially problematic groups. However, a key limitation of this approach is the absence of a temporal dimension, which restricts the assessment of result stability and long-term trends.

Overall, the clustering-based hybrid analytical approach represents a flexible and practice-oriented instrument for course-outcome-based quality assurance processes, particularly in contexts that require timely analytical responses and the explainable use of data-driven methods.

The approaches described represent two distinct yet complementary analytical configurations for assessment-based quality assurance and reflect different logics and contexts of data-informed analysis. In this regard, a comparative examination of these approaches is of particular interest.

The time-based hybrid approach (1–2–3)[1] supports a dynamic and stable evaluation of course outcomes and is particularly effective in identifying structural and recurring issues. In contrast, the clustering-based hybrid approach (1–2–5)[2] is oriented toward simultaneous structural comparison and enables rapid, data-driven interpretation in contexts where only single-semester data are available.

The analysis is conducted using real assessment data, which allows for the evaluation of the practical effectiveness of both hybrid approaches within an authentic educational context. The use of empirical data enables the translation of theoretically described analytical differences into practically interpretable results and highlights the specific strengths and limitations associated with the application of each approach.

## Empirical Application of Hybrid Analytical Approaches

To demonstrate the practical applicability of the proposed hybrid analytical approaches, both methods are implemented in this study using the same real, aggregated assessment data. The dataset covers course-level outcomes from the Fall semesters of three consecutive academic years (2022–2023, 2023–2024, and 2024–2025). The purpose of the empirical analysis is not to evaluate individual instructors or students, but rather to illustrate how the analysis of grade distributions at the course level can support data-informed quality assurance processes.

The data used in the analysis include both percentage-based and absolute distributions of final student grades (A–F) by course, as well as the total number of students enrolled in each course. To avoid unstable or misleading interpretations, the core analytical conclusions are based only on courses with a minimum enrollment of N students (in this case, N = 10). Courses with

[1] (1) Descriptive grade distribution analysis;
(2) Threshold- and indicator-based evaluation approaches;
(3) Longitudinal and trend-based analysis;.

[2] (1) Descriptive grade distribution analysis;
(2) Threshold- and indicator-based evaluation approaches;
(5) Analytics- and clustering-based methods

small cohort sizes are treated as preliminary signals that require additional contextual evaluation and are not used for drawing final conclusions.

The dataset used in the study comprises 10 courses from the Fall 2022–2023 semester, 15 courses from the Fall 2023–2024 semester, and 17 courses from the Fall 2024–2025 semester. Among these, 7 courses were offered consistently across all three academic years.

*Table 1. Course-level grade distributions (A–F), Fall 2022–2023.*

| Subject | Grades | | | | | | |
|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | Total |
| Computer Graphics Tools (Photoshop) | 28.6% (4) | 21.4% (3) | 14.3% (2) | 0% (0) | 7.1% (1) | 28.6% (4) | 100% (14) |
| Computer Hardware | 3.2% (1) | 3.2% (1) | 29.0% (9) | 32.3% (10) | 19.4% (6) | 12.9% (4) | 100% (31) |
| Computer Networks | 0% (0) | 0% (0) | 0% (0) | 36.4% (4) | 36.4% (4) | 27.3% (3) | 100% (11) |
| Computer Skills | 0% (0) | 7.7% (1) | 7.7% (1) | 23.1% (3) | 30.8% (4) | 30.8% (4) | 100% (13) |
| Database Management Systems (MS SQL Server) | 0% (0) | 0% (0) | 13.3% (2) | 46.7% (7) | 20.0% (3) | 20.0% (3) | 100% (15) |
| English Language B1.1 | 0% (0) | 0% (0) | 0% (0) | 30.8% (4) | 30.8% (4) | 38.5% (5) | 100% (13) |
| Fundamentals of Web Design | 9.7% (3) | 9.7% (3) | 19.4% (6) | 19.4% (6) | 16.1% (5) | 25.8% (8) | 100% (31) |
| Introduction to Programming (Python-Based) | 7.7% (1) | 30.8% (4) | 23.1% (3) | 7.7% (1) | 0% (0) | 30.8% (4) | 100% (13) |
| Linear Algebra and Analytic Geometry | 0% (0) | 0% (0) | 5.6% (1) | 33.3% (6) | 22.2% (4) | 38.9% (7) | 100% (18) |
| Object-Oriented Programming (C++) | 0% (0) | 7.1% (1) | 0% (0) | 28.6% (4) | 28.6% (4) | 35.7% (5) | 100% (14) |

*Table 2. Course-level grade distributions (A–F), Fall 2023–2024.*

| Subject | Grades | | | | | | |
|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | Total |
| Academic Writing | 0% (0) | 20% (2) | 20% (2) | 0% (0) | 0% (0) | 60% (6) | 100% (10) |
| Applied Programming | 0% (0) | 0% (0) | 27.3% (3) | 45.5% (5) | 0% (0) | 27.3% (3) | 100% (11) |
| Calculus | 10% (2) | 15% (3) | 10% (2) | 35% (7) | 15% (3) | 15% (3) | 100% (20) |
| Computer Graphics Tools (Photoshop) | 2% (1) | 7.8% (4) | 7.8% (4) | 21.6% (11) | 15.7% (8) | 45.1% (23) | 100% (51) |
| Computer Hardware | 0% (0) | 0% (0) | 4% (2) | 10% (5) | 14% (7) | 72% (36) | 100% (50) |
| Computer Networks | 0% (0) | 0% (0) | 0% (0) | 13.6% (3) | 45.5% (10) | 40.9% (9) | 100% (22) |
| Computer Skills | 0% (0) | 7.7% (4) | 17.3% (9) | 25% (13) | 23.1% (12) | 26.9% (14) | 100% (52) |
| English for Informatics | 18.2% (2) | 9.1% (1) | 9.1% (1) | 36.4% (4) | 0% (0) | 27.3% (3) | 100% (11) |
| English Language B2.1 | 0% (0) | 0% (0) | 6.7% (1) | 20% (3) | 40% (6) | 33.3% (5) | 100% (15) |
| English Language B2.2 | 10% (1) | 0% (0) | 40% (4) | 30% (3) | 0% (0) | 20% (2) | 100% (10) |
| Fundamentals of Web Design | 4.2% (2) | 2.1% (1) | 8.3% (4) | 10.4% (5) | 12.5% (6) | 62.5% (30) | 100% (48) |
| Introduction to Programming (Python-Based) | 5.8% (3) | 9.6% (5) | 26.9% (14) | 23.1% (12) | 7.7% (4) | 26.9% (14) | 100% (52) |
| Object-Oriented Programming (C++) | 9.5% (2) | 4.8% (1) | 4.8% (1) | 23.8% (5) | 38.1% (8) | 19% (4) | 100% (21) |
| Probability Theory and Mathematical Statistics | 0% (0) | 0% (0) | 7.1% (1) | 57.1% (8) | 14.3% (2) | 21.4% (3) | 100% (14) |
| Web Programming (Server-Side) | 7.1% (1) | 7.1% (1) | 7.1% (1) | 14.3% (2) | 42.9% (6) | 21.4% (3) | 100% (14) |

*Table 3. Course-level grade distributions (A–F), Fall 2024–2025.*

| Subject | Grades | | | | | | |
|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | Total |
| Applied Programming | 0% (0) | 0% (0) | 11.1% (2) | 22.2% (4) | 27.8% (5) | 38.9% (7) | 100% (18) |
| Calculus | 14.3% (2) | 35.7% (5) | 21.4% (3) | 7.1% (1) | 0% (0) | 21.4% (3) | 100% (14) |
| Computer Graphics Tools (Photoshop) | 4% (1) | 16% (4) | 8% (2) | 12% (3) | 0% (0) | 60% (15) | 100% (25) |
| Computer Hardware | 2.9% (1) | 8.8% (3) | 5.9% (2) | 14.7% (5) | 0% (0) | 67.6% (23) | 100% (34) |
| Computer Networks | 0% (0) | 0% (0) | 5.3% (1) | 21.1% (4) | 21.1% (4) | 52.6% (10) | 100% (19) |
| Computer Skills | 0% (0) | 5.9% (1) | 17.6% (3) | 23.5% (4) | 5.9% (1) | 47.1% (8) | 100% (17) |
| Database Management Systems (MS SQL Server) | 7.7% (2) | 7.7% (2) | 11.5% (3) | 30.8% (8) | 11.5% (3) | 30.8% (8) | 100% (26) |
| English for Informatics | 5% (1) | 15% (3) | 35% (7) | 30% (6) | 5% (1) | 10% (2) | 100% (20) |
| Fundamentals of Artificial Intelligence | 0% (0) | 0% (0) | 0% (0) | 16.7% (3) | 55.6% (10) | 27.8% (5) | 100% (18) |
| Fundamentals of Web Design | 0% (0) | 0% (0) | 4.2% (1) | 12.5% (3) | 16.7% (4) | 66.7% (16) | 100% (24) |
| Introduction to Programming (Python-Based) | 6.2% (1) | 18.8% (3) | 6.2% (1) | 12.5% (2) | 6.2% (1) | 50% (8) | 100% (16) |
| Mathematical Modeling | 14.3% (4) | 14.3% (4) | 14.3% (4) | 25% (7) | 0% (0) | 32.1% (9) | 100% (28) |
| Mobile Application Design and Development | 0% (0) | 20% (2) | 10% (1) | 30% (3) | 0% (0) | 40% (4) | 100% (10) |
| Object-Oriented Programming (C++) | 17.6% (3) | 11.8% (2) | 17.6% (3) | 5.9% (1) | 17.6% (3) | 29.4% (5) | 100% (17) |
| Probability Theory and Mathematical Statistics | 0% (0) | 26.3% (5) | 10.5% (2) | 31.6% (6) | 0% (0) | 31.6% (6) | 100% (19) |
| Programming on the JVM Platform I | 16.7% (2) | 8.3% (1) | 25% (3) | 25% (3) | 8.3% (1) | 16.7% (2) | 100% (12) |
| Web Programming (Server-Side) | 6.2% (1) | 6.2% (1) | 6.2% (1) | 18.8% (3) | 18.8% (3) | 43.8% (7) | 100% (16) |

*Practical Application of the Time-Based Hybrid Analytical Approach to Course-Level Data*

The time-based hybrid analytical approach examines course-level outcomes jointly across three academic years. The primary objective of the analysis is to assess the extent to which course outcomes remain stable over time. In this context, particular attention is given to specific segments of grade distributions that are interpreted as indicators of low and high academic outcomes. These indicators are not defined through a single fixed combination but are applied in different aggregated forms, depending on the analytical objective and the contextual characteristics of each course.

In the analysis of low outcomes, both the individual grade F and its extended aggregations (E–F or D–E–F) may be employed, reflecting different levels of interpretative strictness. For instance, focusing solely on grade F highlights cases of extreme failure, whereas the D–E–F aggregation provides a broader picture of the proportion of students who failed to meet minimum course requirements or achieved results at a marginal level. This distinction is important, as a given course may not exhibit a high share of F grades while still being characterized by a substantial concentration of D and E grades near the passing threshold.

Similarly, the analysis of high outcomes is not limited to grade A alone. In practice, both A and the aggregated A–B indicators are used, as they capture different aspects of high achievement distribution. An analysis based exclusively on grade A highlights exceptional performance, whereas the A–B aggregation provides a more stable and less sensitive indicator that better reflects overall academic success at the course level.

Within the framework of this study, grade distribution analysis was conducted using A–B (high outcomes) and E–F (low outcomes) indicators, with 30% and 40% aggregation thresholds, respectively, applied as critical cut-off values. These thresholds are treated as clear and interpretable signals for identifying structural characteristics in course-level assessment profiles.

Courses were identified in which the share of E–F grades reached or exceeded 40% in all three academic years. Such recurrence indicates that a high concentration of low outcomes does not represent a one-off anomaly but rather reflects a stable, time-consistent pattern. In these courses, the E–F share consistently reaches or exceeds the 40% threshold across all three academic years, pointing to systemic challenges related to course content difficulty, assessment design, or instructional workload. These courses are classified as cases of concern and represent the strongest signals identified through the time-based hybrid analytical approach.

*Table 4. Courses with recurring low-outcome signals (E–F ≥ 40%)*

| Subject | E–F Grades | | |
|---|---|---|---|
| | E–F (2022–2023) | E–F (2023–2024) | E–F (2024–2025) |
| Computer Networks | 63.7% | 86.4% | 73.7% |
| Computer Skills | 61.6% | 50.0% | 53.0% |
| Fundamentals of Web Design | 41.9% | 75.0% | 83.4% |
| Object-Oriented Programming (C++) | 64.3% | 57.1% | 47.0% |

In addition, the share of A–B grades was analyzed using a 30% threshold. No course was found to exhibit a recurring A–B share at or above this threshold across all three academic years. In line with the logic of the approach, this indicates that, within the analyzed dataset, concentrations of high outcomes, despite notable increases observed in individual years do not demonstrate temporal stability and therefore do not constitute a systemic pattern. Accordingly, signals derived from the A–B indicator are interpreted as single-year or episodic phenomena rather than as structural characteristics at the course level.

Finally, the application of the time-based hybrid analytical approach demonstrates that, despite the availability of relatively extensive data across three academic years (10 courses in 2022–2023, 15 courses in 2023–2024, and 17 courses in 2024–2025), only 7 courses were suitable for longitudinal analysis, as they were offered consistently across all three years. Among these, only 4 courses exhibited recurring signals of concern, while the remaining courses fell outside the scope of detailed analysis under this approach. This finding indicates that the time-based hybrid analytical approach is effective in identifying stable and clearly defined problematic trends, yet its analytical coverage is inherently limited. Consequently, a more comprehensive and fine-grained understanding of course-level quality requires the application of additional analytical approaches alongside time-based analysis.

*Practical Application of the Clustering-Based Hybrid Analytical Approach to Course-Level Data*

The clustering-based hybrid analytical approach is grounded in a distinct analytical logic and aims to identify structural similarities and differences among courses within the scope of single-semester data. This approach is particularly effective in cases where a subset of courses does not meet the criterion of multi-year recurrence and therefore cannot be incorporated into time-based analytical frameworks.

Within this approach, each course is treated as a multidimensional object described by aggregated characteristics of grade distributions (e.g., proportions of high grades (A–B) and low grades (E–F); the inclusion of intermediate grade distributions is also possible). The objective of clustering is to group courses that occupy similar positions in the structure of their assessment profiles. This approach does not predefine problematic or high-performing cases; rather, it provides a basis for their identification through comparative analysis across courses and interpretation of the internal structure of the resulting clusters.

Relative analysis of courses within the space of assessment profiles compensates for the limitations of time-based analyses that arise from requirements of temporal recurrence. As a result, this approach functions as a complementary and more sensitive analytical tool, enabling the identification of structural patterns within a single-semester perspective, including courses that fall outside the scope of longitudinal analysis.

Based on the resulting clusters, courses can be subjected to substantive interpretation as well as grouped according to academic fields. In addition, similarities in assessment practices among instructors, the use of comparable assessment systems across courses, or other recurring approaches within the teaching and learning process may be identified.

Within the scope of the study, the implementation of the clustering-based hybrid analytical approach employed an agglomerative hierarchical clustering algorithm. This algorithm was selected due to its interpretability, its effective performance on small-scale datasets, and the absence of a requirement to predefine the number of clusters, which is particularly important for the relative analysis of assessment profiles at the course level. The hierarchical structure enables similarities and differences among courses to be assessed in a stepwise manner and visually represented within the assessment profile space.

It should be noted that the proposed analytical framework is not tied to a single clustering algorithm; depending on the research objectives, alternative methods such as k-means, DBSCAN, or other clustering algorithms may also be employed. The choice of algorithm depends on the data structure, the number of courses, and the interpretative goals of the analysis.

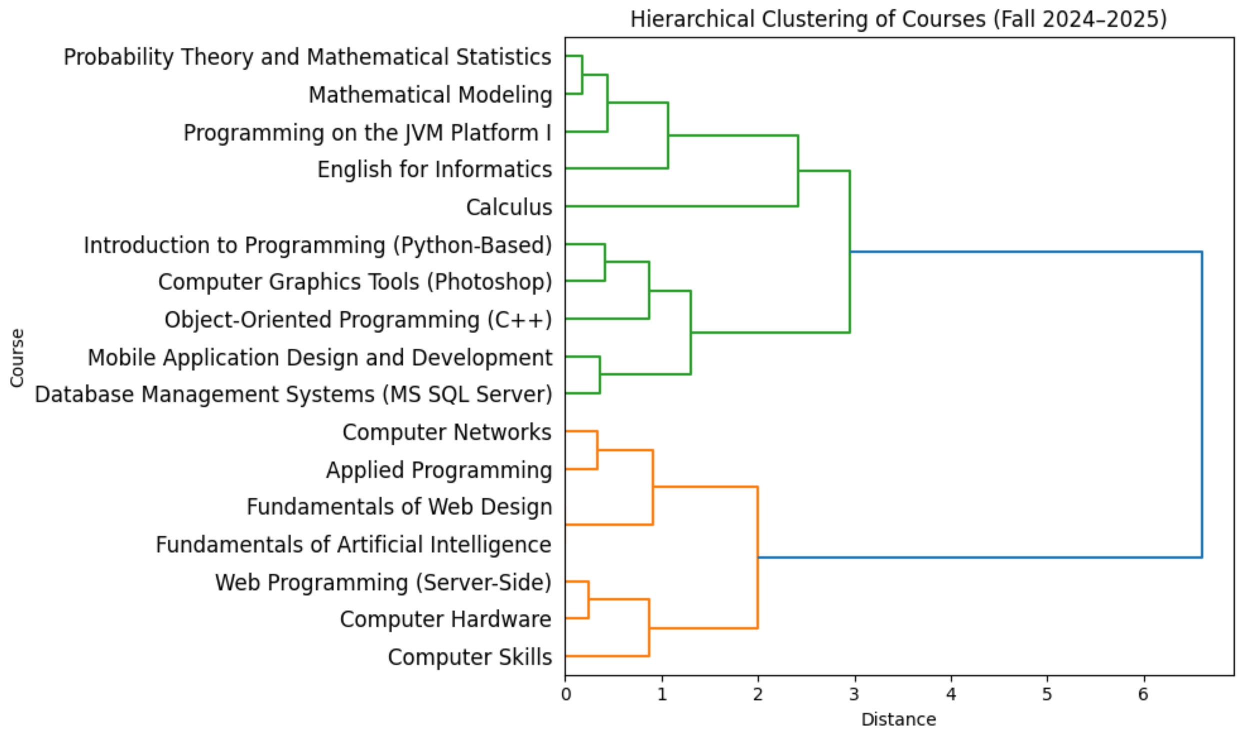
The clustering-based hybrid analytical approach was applied within a single-semester framework and is based on course-level assessment data obtained from the Fall semester of the 2024–2025 academic year. Each course is represented as a two-dimensional object within the assessment profile space, using aggregated proportions of high outcomes (A–B) and low outcomes (E–F). This representation allows for the simultaneous assessment of both academic achievement and academic difficulty at the course level.

To assess the degree of similarity among courses, Euclidean distance was employed, while Ward's linkage method was used for cluster formation, as it minimizes within-cluster variance and ensures the derivation of more stable and interpretable clusters. The implementation of the method was carried out on the Google Colab platform, which ensures accessibility of the computational environment, reproducibility of results, and transparency of the tools employed. Data analysis and clustering were performed using the Python programming language. At different stages of the analytical process, the following libraries were utilized:

- ✓ pandas — for loading and processing aggregated course-level assessment data;
- ✓ numpy — for numerical operations and vector representations;
- ✓ scikit-learn — for distance computation and supporting analytical functions;
- ✓ scipy — for the implementation of the agglomerative hierarchical clustering algorithm;
- ✓ matplotlib — for dendrogram visualization.

To provide a visual representation of the clustering results, a hierarchical dendrogram was constructed, illustrating similarities and differences among courses within the assessment profile space. The dendrogram enables stepwise observation of the cluster formation process and facilitates the identification of the structural relationships among courses within a single-semester perspective.

*Figure 1. Hierarchical clustering dendrogram of courses based on aggregated grade distribution profiles (A–B and E–F ratios).*




Based on the dendrogram, courses were ordered according to hierarchical similarity and transferred into a tabular format, which facilitates clearer interpretation of the clusters and comparative analysis. The table reflects the positioning of courses in accordance with the structure of the dendrogram.

*Table 5. Course-level grade distribution profiles ordered according to hierarchical clustering results.*

| Subject | Grades | | | | | | |
|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | Total |
| **Cluster 1** | | | | | | | |
| Probability Theory and Mathematical Statistics | 0% (0) | 26.3% (5) | 10.5% (2) | 31.6% (6) | 0% (0) | 31.6% (6) | 100% (19) |
| Mathematical Modeling | 14.3% (4) | 14.3% (4) | 14.3% (4) | 25% (7) | 0% (0) | 32.1% (9) | 100% (28) |
| Programming on the JVM Platform I | 16.7% (2) | 8.3% (1) | 25% (3) | 25% (3) | 8.3% (1) | 16.7% (2) | 100% (12) |
| English for Informatics | 5% (1) | 15% (3) | 35% (7) | 30% (6) | 5% (1) | 10% (2) | 100% (20) |
| Calculus | 14.3% (2) | 35.7% (5) | 21.4% (3) | 7.1% (1) | 0% (0) | 21.4% (3) | 100% (14) |
| **Cluster 2** | | | | | | | |
| Introduction to Programming (Python-Based) | 6.2% (1) | 18.8% (3) | 6.2% (1) | 12.5% (2) | 6.2% (1) | 50% (8) | 100% (16) |
| Computer Graphics Tools (Photoshop) | 4% (1) | 16% (4) | 8% (2) | 12% (3) | 0% (0) | 60% (15) | 100% (25) |
| Object-Oriented Programming (C++) | 17.6% (3) | 11.8% (2) | 17.6% (3) | 5.9% (1) | 17.6% (3) | 29.4% (5) | 100% (17) |
| Mobile Application Design and Development | 0% (0) | 20% (2) | 10% (1) | 30% (3) | 0% (0) | 40% (4) | 100% (10) |
| Database Management Systems (MS SQL Server) | 7.7% (2) | 7.7% (2) | 11.5% (3) | 30.8% (8) | 11.5% (3) | 30.8% (8) | 100% (26) |

| Clauter 3 | | | | | | | |
|---|---|---|---|---|---|---|---|
| Computer Hardware | 2.9% (1) | 8.8% (3) | 5.9% (2) | 14.7% (5) | 0% (0) | 67.6% (23) | 100% (34) |
| Applied Programming | 0% (0) | 0% (0) | 11.1% (2) | 22.2% (4) | 27.8% (5) | 38.9% (7) | 100% (18) |
| Fundamentals of Web Design | 0% (0) | 0% (0) | 4.2% (1) | 12.5% (3) | 16.7% (4) | 66.7% (16) | 100% (24) |
| Fundamentals of Artificial Intelligence | 0% (0) | 0% (0) | 0% (0) | 16.7% (3) | 55.6% (10) | 27.8% (5) | 100% (18) |
| Web Programming (Server-Side) | 6.2% (1) | 6.2% (1) | 6.2% (1) | 18.8% (3) | 18.8% (3) | 43.8% (7) | 100% (16) |
| Computer Networks | 0% (0) | 0% (0) | 5.3% (1) | 21.1% (4) | 21.1% (4) | 52.6% (10) | 100% (19) |
| Computer Skills | 0% (0) | 5.9% (1) | 17.6% (3) | 23.5% (4) | 5.9% (1) | 47.1% (8) | 100% (17) |

The clustering-based hybrid approach employed in this study, which utilizes a hierarchical clustering method, yields three clusters that group courses exhibiting relatively similar structures in the distribution of grades according to the A–B and E–F parameters. Accordingly, it may be assumed that these courses share certain common characteristics, which may be related to course content features, teaching–learning methodologies, assessment systems, the responsible instructor, or other specific instructional components.

At the same time, it is important to note that although cluster formation is based on overall profile similarity rather than on a hierarchy of grade levels, the clustering results may nevertheless reveal certain dominant trends in grade distributions within specific clusters. These trends provide additional interpretative opportunities but do not constitute criteria for cluster definition. In the specific case examined in this study, it may be assumed that the clusters identified through clustering differ from one another in terms of structural tendencies in grade distributions. Specifically, the first cluster is characterized by a relatively substantial proportion of high grades (A–B) with a non-dominant distribution of low grades (E–F); the second cluster is marked by a concentration of intermediate grades (C–D), with high and low grades present at moderate levels; and the third cluster exhibits a relatively high concentration of low grades (E–F) accompanied by minimal representation of high grades.

## Conclusion

This study examined the practical applicability of assessment-based, hybrid analytical approaches for course-level quality assurance in higher education using aggregated grade distribution data. Rather than focusing on individual students or instructors, the analysis was deliberately framed at the course level, reflecting the realities of institutional quality assurance processes where aggregated outcomes often constitute the primary available evidence. The findings demonstrate that outcome-level grade distribution data, when interpreted through structured and explainable analytical frameworks, can provide meaningful and actionable insights for data-informed decision making in quality assurance.

The empirical application of the time-based hybrid analytical approach highlighted its particular strength in identifying stable and recurring patterns in course outcomes across multiple academic years. By combining descriptive analysis, threshold-based indicators, and longitudinal trend analysis, this approach enables the detection of structural and time-consistent issues that are unlikely to be revealed through single-semester analysis. The results show that only a limited number of courses met the criteria for longitudinal examination, yet among these, recurring low-outcome signals were clearly identifiable. This finding underscores both the analytical robustness and the inherent limitation of time-based approaches: while they provide high-confidence signals regarding long-term course performance, their applicability is constrained by data availability and course continuity over time.

In contrast, the clustering-based hybrid analytical approach proved effective in contexts where longitudinal data are unavailable or where rapid, comparative interpretation of course outcomes is required. By treating courses as multidimensional objects defined by aggregated outcome characteristics and applying explainable unsupervised clustering techniques, this approach enables the identification of structural similarities and differences among courses within a single academic period. The resulting clusters do not represent predefined quality categories but rather provide a relative analytical space within which course-level performance profiles can be interpreted. This makes the clustering-based approach particularly suitable for exploratory analysis and operational quality assurance tasks, complementing the more stable but less inclusive time-based analysis.

Taken together, the empirical findings demonstrate that the value of hybrid analytical approaches lies not in their direct comparison as competing methods, but in their complementary roles within quality assurance processes. The time-based approach supports strategic, long-term evaluation by emphasizing stability and recurrence, while the clustering-based approach enhances sensitivity and inclusiveness by enabling the analysis of a broader set of courses within a single-semester framework. When used jointly, these approaches provide a more comprehensive and context-aware understanding of course-level outcomes than either method could achieve in isolation.

Several limitations of the study should be acknowledged. The analysis is based on aggregated assessment data from a single institutional context, which restricts the generalizability of the findings. Threshold values and outcome aggregations were applied as heuristic signals rather than normative benchmarks and therefore require careful contextual interpretation. In addition, clustering results depend on the chosen analytical representation and distance measures and should be interpreted as exploratory rather than deterministic indicators of course quality.

Despite these limitations, the study demonstrates that explainable, hybrid analytical frameworks can significantly enhance the interpretability and practical usability of assessment data in course-level quality assurance. Future research may extend this work by applying the proposed approaches across multiple institutions and academic programs, integrating additional outcome indicators, and embedding such analytical pipelines within institutional quality assurance cycles and digital dashboards. Rather than replacing pedagogical judgment, the approaches discussed in this study offer structured analytical support for reflective, evidence-informed quality improvement in higher education.

## REFERENCES

Biggs, J., Tang, C., & Kennedy, G. (2022). Teaching for quality learning at university (5th ed.). McGraw-Hill Education.

Boretz, E. (2004). Grade inflation and the myth of student consumerism. *College Teaching, 52*(2), 42–46. https://doi.org/10.3200/CTCH.52.2.42-46

European Association for Quality Assurance in Higher Education. (2015). *Standards and guidelines for quality assurance in the European Higher Education Area (ESG).* https://ehea.info/media.ehea.info/file/ESG/00/2/ESG_2015_616002.pdf

Harvey, L., & Green, D. (1993). Defining quality. *Assessment & Evaluation in Higher Education, 18*(1), 9–34. https://doi.org/10.1080/0260293930180102

Martin, F., Dennen, V. P., & Bonk, C. J. (2020). A synthesis of systematic review research on emerging learning environments and technologies. *Educational Technology Research and Development, 68*(4), 1613–1633. https://doi.org/10.1007/s11423-020-09812-2

Rexwinkel, T., Haenen, J., & Pilot, A. (2013). Quality assurance in higher education: Analysis of grades for reviewing course levels. *Quality & Quantity, 47*(1), 581–598. https://doi.org/10.1007/s11135-011-9551-9

Schildkamp, K., & Kuiper, W. (2010). Data-informed curriculum reform: Which data, what purposes, and promoting and hindering factors. *Teaching and Teacher Education, 26*(3), 482–496. https://doi.org/10.1016/j.tate.2009.06.007

Shafiq, D. A., Marjani, M., Habeeb, R. A. A., & Asirvatham, D. (2022). Student retention using educational data mining and predictive analytics: A systematic literature review. *IEEE Access, 10*, 72480–72503. https://doi.org/10.1109/ACCESS.2022.3188767

Tabatadze, B. (2023). Modern possibilities of data visualization in digital platform of Moodle learning management system. *Globalization and Business, 8*(16), 111–117. https://doi.org/10.35945/gb.2023.16.010

Tabatadze, B., & Sokhadze, E. (2023). Possibility of improving the training courses content based on statistical analysis of the collected points. *Globalization and Business, 8*(15), 54–56. https://doi.org/10.35945/gb.2023.15.005

Webber, K. L., & Zheng, H. (2020). Data analytics and the imperatives for data-informed decision making in higher education. In K. L. Webber & H. Zheng (Eds.), *Big data on campus: Data analytics and decision making in higher education* (pp. 3–29). Johns Hopkins University Press.

# A Cloud-Based Virtual Sensor Approach for Intelligent Marine Water Quality Monitoring

Nino Topuria

*Candidate of Technical Sciences, Professor, Georgian Technical University, Tbilisi, Georgia.*

***Email:*** *nino.topuria@gtu.ge*

## Abstract

Recent studies indicate that the practical implementation of virtual sensors is closely associated with Internet of Things (IoT) technologies, which enable continuous data acquisition and reliable transmission from distributed sensing devices. The rapid scalability of IoT systems and the exponential growth of telemetric data streams have created an increasing demand for cloud computing integration, allowing efficient storage, processing, and real-time analysis of large-scale datasets [1], [2]. The literature emphasizes that cloud platforms provide a favorable environment for the deployment of virtual sensing frameworks, as they support centralized data management, dynamic allocation of computational resources, and advanced analytical capabilities [3], [4].

The integration of Artificial Intelligence (AI) and Machine Learning (ML) algorithms with IoT-based cloud infrastructures is widely recognized as a key enabler for the development of intelligent monitoring systems. Numerous studies demonstrate that AI/ML techniques significantly enhance anomaly detection accuracy, predictive analytics performance, and real-time decision support, particularly when virtual sensors are employed to estimate environmental parameters or validate physical measurements [1], [5], [6]. This approach reduces the impact of measurement noise and sensor drift while improving the overall reliability and robustness of monitoring systems.

Although peer-reviewed publications specifically focused on Azure IoT Central remain relatively limited, the broader scientific and technical literature confirms the potential of cloud-based IoT platforms to provide secure communication, scalable analytics, real-time data ingestion, and predictive insight generation [2], [3], [7],[8]. Therefore, existing research suggests that the

integration of virtual sensors, IoT technologies, and cloud infrastructures represents a promising and still underexplored direction for the development of intelligent marine water quality monitoring systems, reinforcing the scientific relevance and practical significance of this study.

**Keywords:** Marine Water Quality Monitoring, Virtual Sensors, Internet of Things (IoT), Azure IoT Central, AI-Based Monitoring, Cloud-Based Analytics

## Introduction

Marine water is a critical resource for both human well-being and ecosystem health, playing an essential role at environmental, economic, and societal levels. Climate change, domestic and industrial waste discharges, as well as pollution caused by fertilizers and pesticides, have increasingly negative impacts on the stability and biodiversity of marine ecosystems [9],[10]. Under these conditions, continuous and reliable monitoring of marine water quality has become a fundamental requirement for effective environmental management and informed decision-making.

Conventional marine water quality monitoring systems are predominantly based on physical sensors that directly measure parameters such as temperature, pH, electrical conductivity, and other physicochemical indicators. However, the deployment of physical sensors in marine environments is associated with significant costs and operational challenges, including sensor drift, biofouling, corrosion, and the need for frequent calibration. As a consequence, data reliability may degrade over time, while the overall operational resilience of monitoring systems is reduced [11],[12].

In this context, the adoption of innovative approaches, such as virtual sensors and the integration of Internet of Things (IoT) and Artificial Intelligence (AI) technologies, has gained increasing attention. Virtual sensors enable the estimation of environmental parameters through the exploitation of existing measurements, historical datasets, and predictive models, thereby allowing monitoring systems to become more cost-effective, scalable, and intelligent. Rather than relying exclusively on direct physical measurements, virtual sensing introduces a data-driven paradigm that enhances system flexibility and robustness [13],[14],[15].

The rapid evolution of IoT technologies has significantly expanded the capabilities of environmental monitoring systems. Distributed sensing devices now enable real-time observation of physical, chemical, and biological parameters across both terrestrial and marine environments. Nevertheless, the harsh and dynamic conditions of marine settings continue to limit the effectiveness of approaches based solely on physical sensors. Consequently, alternative and hybrid solutions, particularly those leveraging cloud-based virtual sensing frameworks, have emerged as a promising direction in contemporary environmental research [16].

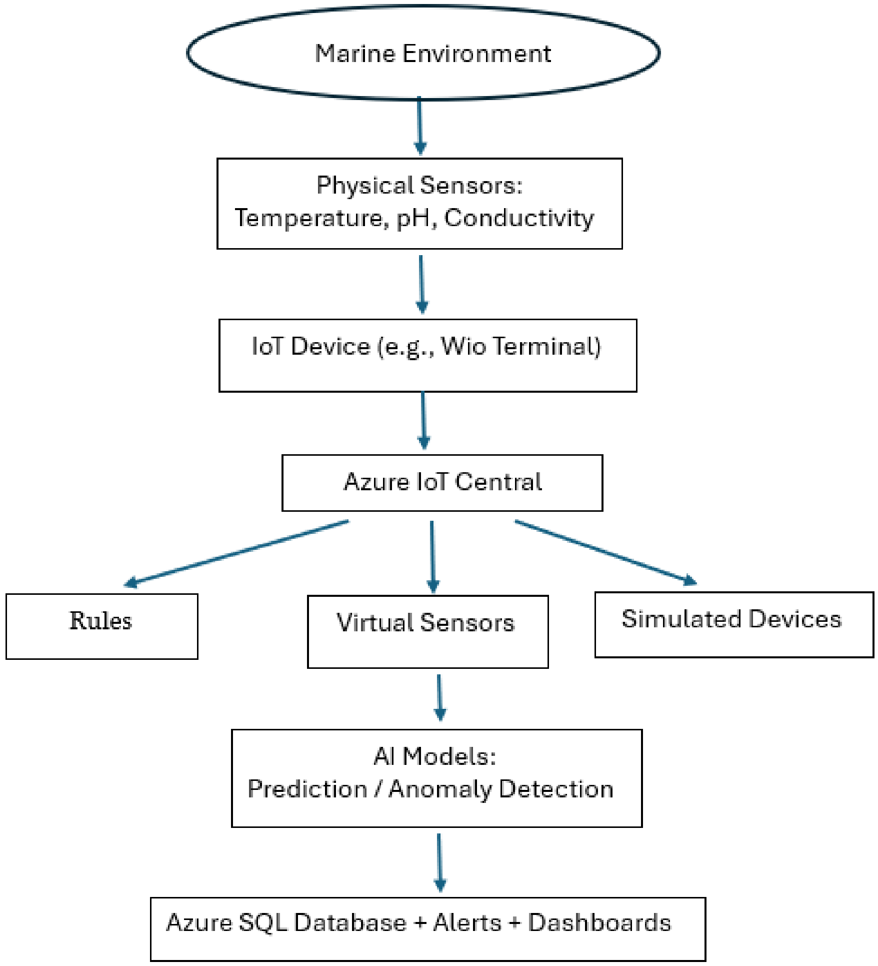
The objective of this paper is to present a conceptual and practical evaluation of virtual sensors implemented on the Azure IoT Central platform for marine water quality monitoring. The study focuses on assessing data reliability, predictive capabilities, and system scalability, highlighting the potential of cloud-based and AI-driven virtual sensing approaches to enhance the efficiency and intelligence of marine environmental monitoring systems.

## System Architecture Overview

The system architecture is based on a multi-layered model, comprising physical sensors, IoT devices, cloud platforms, and analytical components. Virtual sensors are deployed at the cloud layer, functioning as an intelligent intermediary between physical sensors and final decision-making processes [17].

Figure 1 illustrates a modern, cloud-based IoT platform architecture for marine water quality monitoring, integrating physical sensors, virtual sensors, AI models, and automated decision-making mechanisms.

*Figure 1 – A Cloud-Based Virtual Sensor Architecture for Marine Water Quality Prediction*




Monitoring begins in the marine environment, where physical sensors measure key parameters including temperature, pH, conductivity, and other physicochemical attributes. These sensors serve as the primary data source, and their regular measurements provide real-time insights into environmental stability and changes. Data collected by physical sensors is first processed by IoT devices, such as the Wio Terminal, which securely transmits the measurements to the cloud platform. Azure IoT Central serves as the cloud hub for data ingestion, management, and preparation for subsequent processing.

Virtual sensors are data-driven models that do not correspond to physical hardware. Their primary functions include:

- Integration of physical sensor data

- Computation or derivation of new metrics (e.g., Water Quality Index, predicted parameters)
- Provision of input data for AI models and rules engines

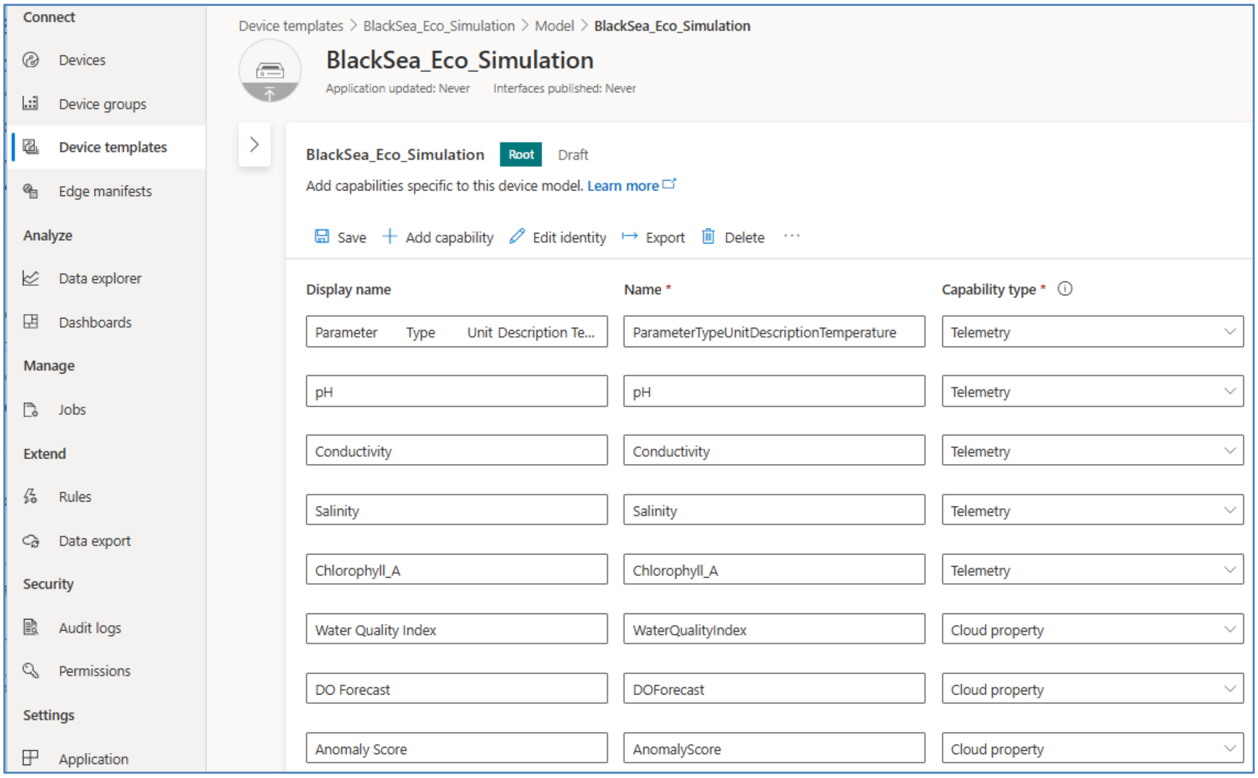
Rules and AI models (e.g., prediction, anomaly detection) utilize virtual sensor outputs to detect unusual water parameter variations and generate forecasts for future changes. All processed data, predictions, and alerts are stored in an Azure SQL Database, which ensures historical record keeping and enables real-time visualization on dashboards accessible to monitoring teams.

Simulated devices are employed for testing and system modeling, generating artificial data streams to validate the resilience of the platform and the accuracy of virtual sensor computations. These devices do not produce real environmental measurements but are essential for verifying anomalies and forecast generation.

Forecasting within the Marine IoT Water Monitoring System involves estimating future values of physicochemical water parameters based on historical measurements from physical sensors combined with additional analytical processes. Data is pre-processed on the virtual sensor layer, including outlier removal, missing value imputation, and normalization or standardization for AI models.

Figure 2 illustrates the Device Template in Azure IoT Central, combining both physical and virtual sensors.

*Figure 2 – Azure IoT Central Device Template*




Physical sensors (Telemetry) represent the primary data source and include parameters such as:

- Temperature (°C);
- pH;
- Conductivity (μS/cm);
- Salinity (PSU);
- Chlorophyll_A (μg/L).

These telemetry parameters are continuously recorded and transmitted to IoT Central, where they are ingested, stored, and visualized on dashboards in real time. Telemetry data provides a basis for environmental monitoring and forms the foundation for virtual sensor computations and AI models.

Virtual sensors (Cloud properties) represent computed or forecasted values not directly sourced from physical devices. In the Device Template, virtual sensors are implemented as cloud properties and include:

- Water Quality Index (WQI) – calculated from physical sensor measurements to indicate overall water quality;
- DO Forecast – AI-generated prediction of future dissolved oxygen levels (mg/L);
- Anomaly Score – AI-derived anomaly indicator (0–1).

Cloud properties enable automatic updates of virtual sensor data for AI models, allowing rules engines to detect abnormal variations and trigger automated alerts. The Device Template also includes a Simulated Device, containing both physical and virtual sensor parameters for system testing, validation of virtual sensor calculations, and AI model verification.

## Conclusion

This study presents a cloud-based virtual sensor approach for intelligent marine water quality monitoring, implemented on the Azure IoT Central platform. The results demonstrate that virtual sensors significantly enhance data reliability, reduce dependence on physical sensors, and enable real-time, intelligent analytics.

The integration of physical sensor measurements, AI models for forecasting and anomaly detection, and automatic cloud-based updates of virtual sensors produces a scalable and cost-efficient monitoring system. The use of simulated devices for system testing and verification further strengthens operational resilience and ensures the accuracy of virtual sensor computations.

The findings confirm that Azure IoT Central–based virtual sensors provide an effective solution for both research and practical applications, particularly in geographically distributed or environmentally challenging marine monitoring sites. Future work may focus on improving the generalization capabilities of AI models across seasonal and hydrological variations, optimizing cloud architecture for real-time analytics, and evaluating long-term operational performance.

Overall, this approach establishes a foundation for the development of modern, intelligent, scalable, and economically sustainable systems for marine water quality management.

## REFERENCES


1. Y. Xu, L. Sun, Y. Liu, and Q. Wang, "Edge computing enabled smart water quality monitoring system based on IoT," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3860–3871, Mar. 2021, doi: 10.1109/JIOT.2020.3028034.
2. K. Kaur and M. Singh, "Cloud-based IoT architectures for environmental monitoring: A comprehensive review," *Journal of Cloud Computing*, vol. 11, no. 1, pp. 1–22, 2022, doi: 10.1186/s13677-022-00293-5.
3. Microsoft, "Azure IoT Central documentation," Microsoft Corp., Redmond, WA, USA, 2021. [Online]. Available: https://learn.microsoft.com/azure/iot-central.
4. X. Gao, X. Li, Y. Zhang, and H. Wang, "Hybrid physical–virtual sensor framework for water quality monitoring using IoT and machine learning," *Sensors*, vol. 22, no. 3, pp. 1–18, 2022, doi: 10.3390/s22030789.
5. J. Kim, S. Park, and H. Lee, "Machine learning-based virtual sensing for environmental monitoring applications," *Environmental Modelling & Software*, vol. 149, p. 105316, 2022, doi: 10.1016/j.envsoft.2022.105316.
6. R. Singh, A. Verma, and P. Kumar, "Artificial intelligence driven anomaly detection in IoT-based water quality monitoring systems," *IEEE Access*, vol. 11, pp. 22345–22358, 2023, doi: 10.1109/ACCESS.2023.3245678.
7. Y. Zhang and J. Wang, "Challenges and solutions in marine water quality monitoring using sensor networks," *Ocean Engineering*, vol. 234, p. 109248, 2021, doi: 10.1016/j.oceaneng.2021.109248.
8. G. Chogovadze, G. Surguladze, N. Topuria, and N. Archvadze, "Implementation of a prediction model with cloud services," *Bulletin of the Georgian National Academy of Sciences*, vol. 14, no. 3, pp. 29–35, 2020.
9. G. Chogovadze, G. Surguladze, N. Topuria, A. Gavardashvili, and T. Namchevadze, "Computer-aided design of the information ecosystem for monitoring of the Black Sea water resources," *Bulletin of the Georgian National Academy of Sciences*, vol. 12, no. 2, pp. 19–26, 2018.
10. T. Lominadze and N. Topuria, "Database realization for the corporation web-portal," in *Information and Computer Technology, Modeling and Control: Proceedings of the International Scientific Conference*, I. V. Prangishvili, Ed., ch. 21, pp. 227–234, 2017.
11. G. Surguladze, N. Topuria, and A. Gavardashvili, "Automation of web-portal construction processes with SQL Server for the Black Sea ecosystem monitoring," *International Journal of Computer and Information Engineering*, vol. 12, no. 2, pp. 169–174, 2018.

12. G. Gavardashvili, G. Surguladze, L. Petriashvili, and N. Topuria, “Designing eco-monitoring information system for the Black Sea coastline based on modern digital technologies,” *Bulletin of the Georgian National Academy of Sciences*, vol. 16, no. 3, pp. 44–49, 2022.
13. G. Gavardashvili, G. Surguladze, L. Petriashvili, T. Zhvania, and N. Topuria, “Automated construction technology of Black Sea eco-monitoring information system,” in *Proc. IEEE Conference*, 2022.
14. G. Surguladze, L. Petriashvili, N. Topuria, and G. Surguladze, “Modelling of designing a conceptual schema for multimodal freight transportation information system,” *International Journal of Computer and Information Engineering*, vol. 9, no. 11, pp. 2320–2323, 2015.
15. G. Surguladze, E. Turkia, N. Topuria, and A. Gavardashvili, “Construction of the multimedia databases and users interfaces for ecological system of Black Sea with ORM/ERM,” in *Information and Computer Technology, Modeling and Control*, ch. 45, Nova Science Publishers, USA, pp. 1–8, 2017.
16. G. Surguladze, E. Turkia, N. Topuria, T. Lominadze, and M. Giutashvili, “Towards an integration of process-modeling: From business-content to the software implementation,” in *Proc. IV Int. Conf. Problems of Cybernetics and Informatics (PCI)*, Baku, Azerbaijan, pp. 1–4, 2012, doi: 10.1109/ICPCI.2012.6486265.
17. G. Surguladze, N. Topuria, and A. Gavardashvili, *Black Sea Ecological Monitoring and Research Information System*. Tbilisi, Georgia: IT-Consulting Scientific Center of GTU, 2018.

# Performance Optimization of CI/CD Pipelines through Redundant Operation Elimination

Saba Gotsiridze

*MSc (Information Systems), Business and Technology University, Tbilisi, Georgia.*

***Email:*** *saba.gotsiridze.1@btu.edu.ge*

Giorgi Kuchava

*PhD (Engineering of Informatics), Associate professor, Georgian Technical University, Tbilisi, Georgia.*

***Email:*** *kuchavagiorgi08@gtu.ge*

Teimuraz Sturua

*Candidate of Technical Sciences, Associate professor, Georgian Technical University, Tbilisi, Georgia.*

***Email:*** *t.sturua@gtu.ge*

Ana Khinchikashvili

*BSc (Information Technologies), Business and Technology University, Tbilisi, Georgia.*

***Email:*** *ana.khinchikashvili.1@btu.edu.ge*

Ekaterine Somkhishvili

*BSc (Information Technologies), Assistant, Business and Technology University, Tbilisi, Georgia.*

***Email:*** *ekaterine.somkhishvili.1@btu.edu.ge*

Ioseb Kartvelishvili

*Candidate of Technical Sciences, Professor, Georgian Technical University, Tbilisi, Georgia.*

***Email:*** *s.kartvelishvili@gtu.ge*

Maka Mantskava

*PhD (Biology), Professor, European University, Tbilisi, Georgia.*

***Email:*** *maia.mantskava@eu.edu.ge*

Shalva Chokhonelidze

*MSc (Computer Science), Senior Scientist, I. Beritashvili Center of Experimental Biomedicine, Tbilisi, Georgia.*

***Email:*** *tchokhonelidze.shalva@atsu.edu.ge*

Nana Momtselidze

*PhD (Biology), Senior Scientist, I. Beritashvili Center of Experimental Biomedicine, Tbilisi, Georgia.*

***Email:*** *nana.momtselidze@eu.edu.ge*

**Abstract**

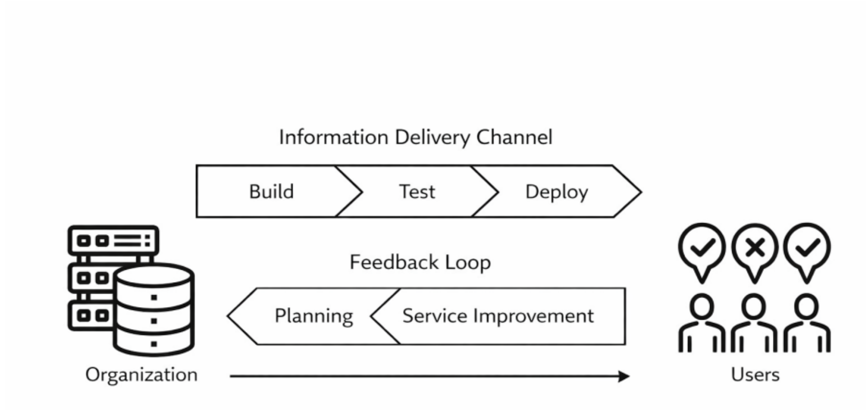
The proposed approach provides actionable insights for DevOps practitioners and supports data-driven decision making in CI/CD pipeline design. This article presents an experimental analysis of Continuous Integration and Continuous Delivery/Deployment (CI/CD) pipelines with a focus on performance optimization through the identification and elimination of redundant operations. Based on controlled experiments conducted in virtual machine environments and real-world CI/CD pipelines, the study demonstrates that unnecessary duplication of build and restore steps significantly increases execution time and resource consumption. A mixed-methods research approach is employed, combining quantitative performance measurements with a practical case study implemented in Azure DevOps. The results indicate a measurable reduction in average build and integration time after applying targeted optimization strategies, including framework-specific configurations, caching mechanisms, and pipeline restructuring. The findings confirm that systematic CI/CD optimization improves software delivery efficiency and supports faster, more reliable release cycles. The study contributes practical guidelines for DevOps teams seeking to enhance pipeline performance while maintaining code quality and system stability [1,2].

**Keywords:** CI/CD, DevOps Optimization, Pipeline Performance, Continuous Integration, Build Time Reduction

## 1. **Introduction**

Before the development of DevOps systems and the widespread availability of personal computers, software throughout its lifecycle was typically developed and maintained by a single team. Subsequent technological advancements contributed to the specialization of software development, testing, and information technology roles, leading to the establishment of structurally distinct positions such as developer, tester, system administrator, network administrator, and project manager. This structural separation introduced a handoff-based workflow, involving the transfer of information and products from one team to another. However, this approach exhibits limited efficiency, as it is practically impossible to fully transfer the knowledge accumulated by one team to another during the handoff process. To illustrate this limitation in efficiency, the following process is considered: Software engineers write code, while IT operations teams manage servers. Developers place the written code into version control repositories, such as GitHub or Azure DevOps, which are managed through Git-based version control systems. Newly introduced code may cause failures in the software system deployed on servers, leading to malfunctioning client-side applications, even though the same code operates correctly on the developer's local machine. As a result of such inefficiencies, both development and operations teams faced various challenges, primarily related to insufficient information sharing and inadequate testing environments. Teams also occasionally made conflicting decisions regarding the trade-off between software delivery flexibility and operational stability. Consequently, the need to overcome inefficient delivery across defined phases of software development became a critical objective. To address these challenges, the DevOps concept was introduced. DevOps is based on establishing a continuous connection between software service producers and end users through a shared delivery pipeline, incorporating structured feedback loops that relay user feedback back to service producers. This approach has evolved into a widely adopted software development strategy that enables organizations to achieve continuous delivery while simultaneously ensuring stable and reliable software evolution. At the core of DevOps lies CI/CD (Continuous Integration and Continuous Delivery/Deployment) , which operates through an information delivery channel known as the pipeline. The logic of Continuous Integration involves the frequent merging of code changes submitted by developers into a shared repository—effectively a common codebase—where updates are continuously integrated with existing functionality. Following the merge process, the system automatically builds and tests the application. Continuous Delivery/Deployment, in turn, refers to the practice whereby code is promoted to a production environment after a manually approved step, according to the delivery logic. Together, CI/CD form a delivery system (Figure 1) that automates the following processes: code build, testing, and deployment into an operational environment. [1,3]

*Figure 1. DevOps Functional Architecture*




Despite their widespread adoption, CI/CD pipelines often suffer from performance inefficiencies caused by redundant operations, suboptimal configurations, and insufficient resource utilization. These inefficiencies increase build times, consume excessive computational resources, and negatively impact developer productivity. As organizations scale their DevOps practices, the need for systematic performance optimization of CI/CD pipelines becomes increasingly important. The objective of this research is to analyze the performance characteristics of CI/CD pipelines, identify sources of inefficiency, and evaluate optimization strategies aimed at reducing execution time without compromising software quality. The study focuses on redundant build and restore operations observed in real-world pipelines and proposes practical optimization techniques validated through controlled experiments and a case study conducted using Azure DevOps [3,4].

The evolution of DevOps has been closely associated with Agile and Lean methodologies, which emphasize iterative development, continuous feedback, and waste reduction. Previous studies have highlighted the importance of CI/CD pipelines as a core component of DevOps, enabling frequent integration and automated testing to improve software reliability and delivery speed. Research on CI/CD performance has primarily focused on process automation, tool integration, and organizational factors influencing DevOps maturity. Several authors have demonstrated that automated testing and continuous integration significantly reduce defect rates and improve deployment frequency. However, fewer studies address the internal efficiency of pipeline execution, particularly the impact of redundant operations on build performance. Recent empirical studies suggest that pipeline optimization through caching mechanisms, parallel execution, and framework-specific configurations can lead to substantial performance gains. Nevertheless, existing literature often lacks detailed experimental validation in controlled environments. This study extends prior work by providing quantitative evidence of performance improvements achieved through the systematic elimination of redundant CI/CD operations, supported by real-world pipeline data [4,5].

## 2. Methodology

Optimization of CI/CD Process Efficiency: As previously mentioned, Continuous Integration and Continuous Delivery/Deployment (CI/CD) processes play a critical role in modern software development. They automate application build, testing, and deployment activities, thereby accelerating the software delivery lifecycle (SDLC). It is important to note that increases in build, compilation, and testing time within these processes can lead to bottlenecks, slow down development velocity, and reduce responsiveness to change. As observed in the literature review, performance evaluation metrics help identify areas for improvement. One of the key metrics is lead time, which encompasses the time spent on code build and testing activities. The conducted study reveals the impact of optimizing repetitive steps in CI/CD processes on reducing build and testing time. A more detailed examination, specifically through the analysis of Azure DevOps pipeline logs, revealed that certain processes contain redundant steps, particularly unnecessary repeated invocations of specific commands. Although individual steps within the pipeline may appear negligible in isolation, their cumulative impact can significantly affect overall efficiency. These repetitive steps potentially contributed to extended build times, thereby hindering the overall effectiveness of the process. Accordingly, this study focuses on analyzing the impact of optimizing repetitive steps in CI/CD processes. [6,7]The central research question examines whether simplifying and optimizing duplicated operations can lead to a significant reduction in build time. This objective aligns with the broader goal of improving CI/CD pipeline efficiency, as the elimination of unnecessary steps can result in faster continuous integration processes. With this understanding of the problem and the potential benefits of optimization, the research objective is defined as analyzing the impact of optimizing repetitive steps in a continuous integration pipeline, using Azure DevOps as a case study, in order to reduce overall build time. The study aims to analyze the effectiveness of eliminating redundant steps in optimizing continuous integration execution time. By combining quantitative analysis with a case study approach, this research examines the potential for significant reductions in build time and provides valuable insights for optimizing CI/CD processes in similar scenarios. By achieving this objective, the study aims to demonstrate a clear relationship between the elimination of duplicated operations and improved build time efficiency in real-world CI/CD process scenarios. The findings provide valuable insights into potential optimization strategies and their effectiveness for similar processes across different frameworks [8,9].

Research Methodology and Its Relevance: This study adopts a mixed-methods approach that combines quantitative analysis with a specific case study in order to comprehensively examine the impact of optimizing repetitive steps on CI/CD execution time. This approach offers several advantages: Quantitative Analysis, Measuring the impact on build time: By running simulations on virtual machines, quantitative analysis enables the execution of controlled experiments to isolate the impact of the proposed changes—specifically the reduction of redundant restore and build operations—on build time. This approach allows for the measurement of actual time differences before and after optimization, thereby providing a clear metric for evaluating efficiency [10,11].

Data-driven decision making: Quantitative analysis enables the application of data analysis techniques to derive statistically meaningful conclusions regarding the effectiveness of the proposed optimization strategy. This strengthens the research findings by providing objective evidence of build time reduction.

Real-world validation: While simulations conducted in a virtual machine environment provide valuable insights, the case study component adds significant real-world applicability. By applying the optimization strategy to real CI/CD pipelines within the Azure DevOps all-in-one platform, it becomes possible to observe its impact on actual continuous integration execution time in a production environment. This enhances the generalizability of the findings by demonstrating effectiveness in a practical, real-world scenario.

Addressing potential variations: Real-world pipelines may involve additional complexities compared to virtual machine environments. The case study approach allows for the identification of any unforeseen factors that may influence build time beyond the targeted optimizations. This provides a more comprehensive understanding of how optimization strategies may interact with existing continuous integration configurations.

Integration of research methods, The combination of quantitative research and a case study approach provides a more comprehensive perspective: Virtual machine experiments provide a controlled environment to isolate the impact of optimization on build time, The case study validates the effectiveness of these findings in a real-world context, while accounting for potential complexities, Together, these methods provide a robust understanding of the potential for reducing build time through the optimization of repetitive steps.

Discussion of Alternative Approaches: Although alternative methodologies could have been employed, a purely quantitative approach (e.g., analyzing only existing pipeline logs) may lack the level of control required to isolate the impact of the proposed optimization strategy. Conversely, a purely qualitative approach (e.g., interviews with software developers and DevOps engineers) may not provide the same level of measurable data regarding build time reduction. The mixed-methods approach establishes a balanced framework by enabling both controlled measurement and real-world application. This combination provides a more comprehensive and reliable understanding of the research objective [11].

## 3. Results and Discussion

Data collection was performed through simulations in a virtual machine environment.

To measure the impact of optimizing redundant steps, experiments were conducted in a virtual machine environment. The experiment involved the creation of two scripts designed to simulate the company's continuous integration process. For automation within the Azure DevOps environment, a Microsoft .NET project was selected.

This choice was motivated by several factors:

Seamless integration with Azure DevOps: Since the Azure DevOps platform had already been analyzed, including its strengths and limitations, it was considered an appropriate choice for the case study. As an all-in-one platform, Azure DevOps simplifies the implementation of the experimental setup by providing an integrated and user-friendly environment with extensive built-in functionality.

Simplified automation for .NET development: Azure DevOps provides extensive support for .NET-related tasks, including building, testing, and deploying .NET applications. This capability simplifies automation workflows for .NET projects within the Azure ecosystem and enables efficient configuration of CI/CD pipelines.

Microsoft support and resources: Selecting a tool developed by Microsoft ensures access to comprehensive and well-documented official resources, including detailed technical documentation and long-term platform support.

Rich CI/CD functionality: Azure DevOps delivers a feature-rich CI/CD solution that supports comprehensive pipeline configuration, execution, and monitoring.

Script 1 (simulation of the existing version): This script represents the commands typically used in a .NET CI pipeline, including dotnet restore, dotnet build, and dotnet test.

```
set log=f:\log_before.txt
cd ../project_path
echo -------  >> %log%
echo %time% >> %log%
dotnet restore project_name
```

```
echo %time% >> %log%
dotnet build project_name
echo %time% >> %log%
dotnet test project_name
echo %time% >> %log%
```

At first glance, this script appears straightforward; however, an examination of Azure DevOps pipeline logs revealed that certain commands are executed multiple times in a redundant manner.

Script 2 (optimized version): This script incorporates the proposed optimizations. To eliminate redundant operations—specifically repeated restore and build processes—the --no-restore and --no-build parameters are applied in the .NET commands.

```
set log=f:\log_after.txt
cd ../project_path
echo -------  >> %log%
echo %time% >> %log%
dotnet restore project_name
echo %time% >> %log%
dotnet build project_name --no-restore
echo %time% >> %log%
dotnet test project_name --no-build
echo %time% >> %log%
```

Although the implementation of optimization strategies varies across different frameworks, the underlying principle remains the same. To illustrate this approach, Java and Python programming environments are considered as representative examples.

A standard Maven build script typically includes the stages of cleaning, building, testing, and packaging. While comprehensive, this process can be time-consuming when repetitive tasks are executed unnecessarily. As an optimization mechanism, Maven profiles can be utilized to define alternative build configurations. For this purpose, a profile named fast-build is created in the pom.xml file (figure 2).

*Figure 2. Maven Profile for Build Process Optimization*

```xml
<profile>
  <id>fast-build</id>
  <build>
    <skipTests>true</skipTests> <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <skip>true</skip> </configuration>
      </plugin>
    </plugins>
  </build>
</profile>
```

When executing the command mvn clean install -P fast-build, the -P parameter specifies the optimized profile, which skips test execution and may potentially bypass the compilation stage if the required artifacts are already available.

In Python, when using pip, the standard build process may include dependency installation, test execution, and the creation of a deployable package. Unlike the .NET and Java examples, built-in optimization mechanisms are not available in the same form; however, optimization can be achieved through the use of virtual environments and dependency caching:

Create a Python virtual environment: python -m venv venv.

Activate the virtual environment: venv/bin/activate on Linux/macOS systems, or venv\Scripts\activate on Windows systems

Install dependencies using caching: pip install --cache-dir ~/cache/pip -r requirements.txt, using a dedicated cache directory to store downloaded packages.

Subsequent build processes can leverage cached dependencies, thereby reducing installation time.

It should be noted that these examples are illustrative. Specific optimization techniques and tools vary depending on the framework and project requirements. Therefore, consulting the official documentation of the selected framework is essential to identify and apply best practices for efficient build and integration processes.

Both scripts developed in this study were executed on a virtual machine configured with specifications comparable to the company's CI environment. To ensure consistency and minimize the influence of external factors, each script was executed five times. During each execution, the start and end times of each build stage (restore, build, and test) were recorded using a timestamping mechanism, specifically the echo %time% command.

Data Analysis: Build Time Improvement

The collected data were analyzed to assess the impact of optimization. To calculate the average build time, the data obtained from each script (existing and optimized) were transferred to an Excel file for further processing and comparison.

Comparison of total process execution time: Based on the data collected from each script, diagrams were constructed and compared. These diagrams illustrate the execution time of each pipeline stage before and after the applied optimizations.

Build time improvement: The percentage improvement in build time achieved as a result of script optimization is calculated using the following formula:

**Build time improvement percentage = (Total execution time of the existing script – Total execution time of the optimized script) / Total execution time of the existing script * 100**

This formula calculates the difference between the total build execution times of the existing and optimized scripts. The resulting percentage represents the improvement in build time achieved through script optimization.

In the presented diagram (Figure 5), a comparison of the total process execution times before and after optimization is shown. As can be observed visually, the execution time of the original script is consistently higher than that of the optimized script at each stage of the pipeline. The change is particularly evident in the testing time comparison diagram (Figure 5), which demonstrates a significant difference in the execution time of the dotnet test stage between the two scripts.

Following script optimization, the total process execution time improved by 26.78%, as shown in Table 1. Based on this observation, it can be inferred that eliminating redundant processes can significantly enhance the efficiency of the continuous integration workflow. In this particular case, the strategic use of the --no-restore and --no-build parameters enabled a streamlined build process.

*Figure 3. Number of Runs and Continuous Integration Simulation Time.*

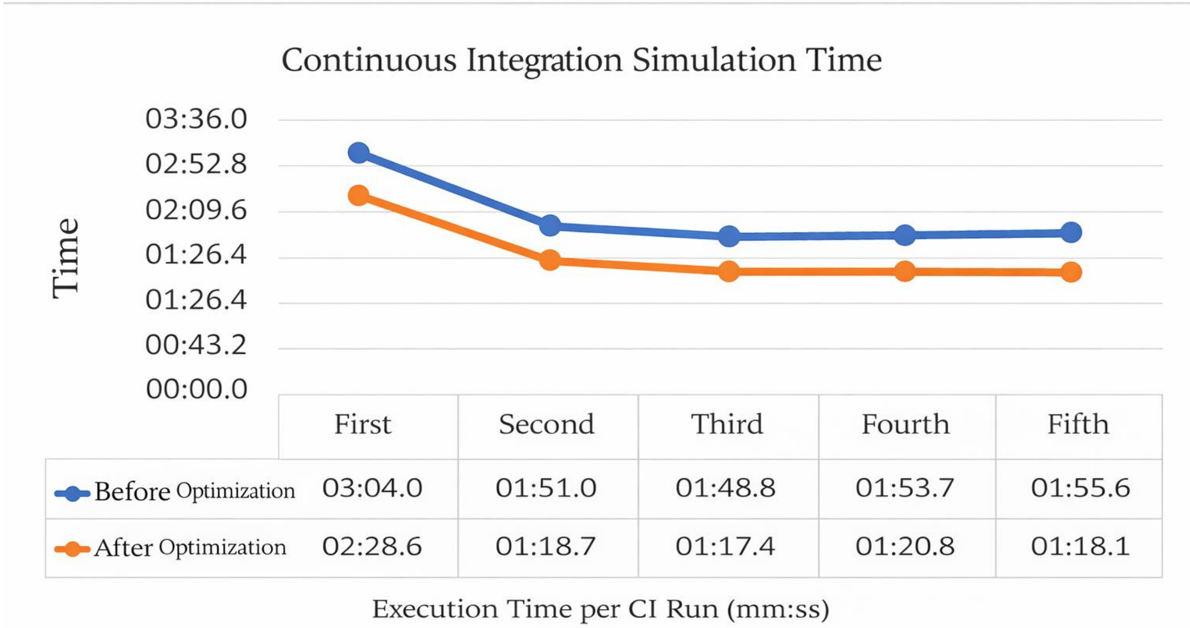



*Figure 4. Number of Runs and Restore Time.*

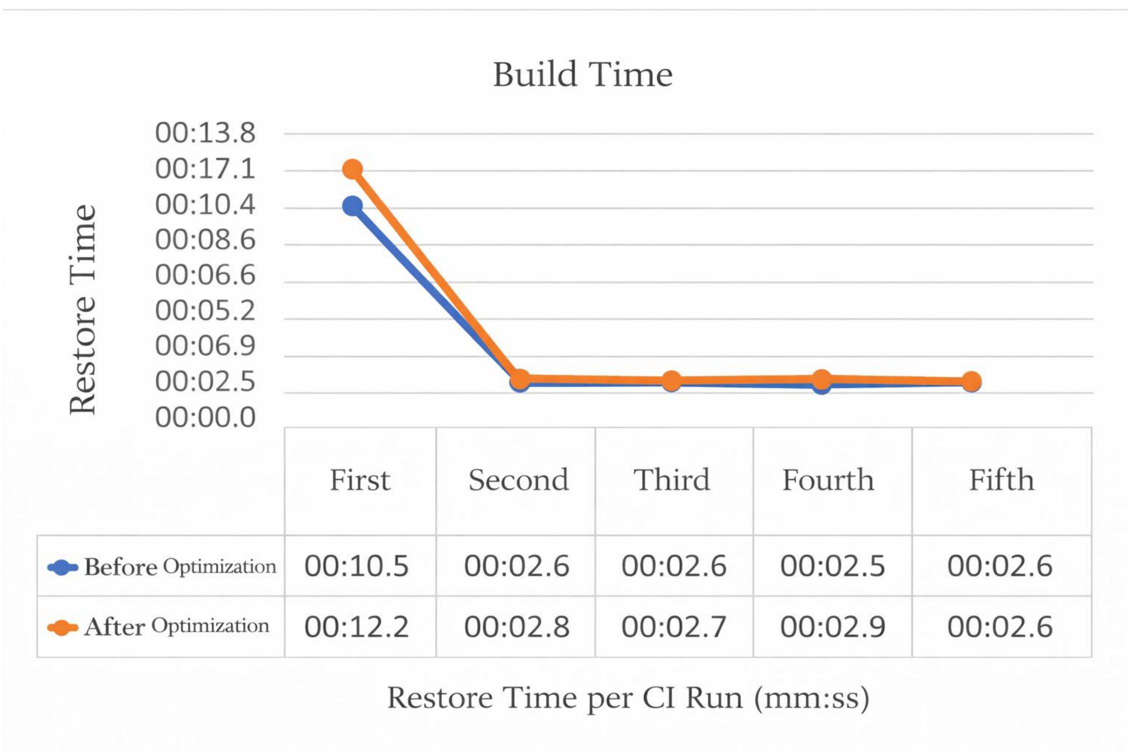


*Figure 5. Number of Runs and Restore Time.*

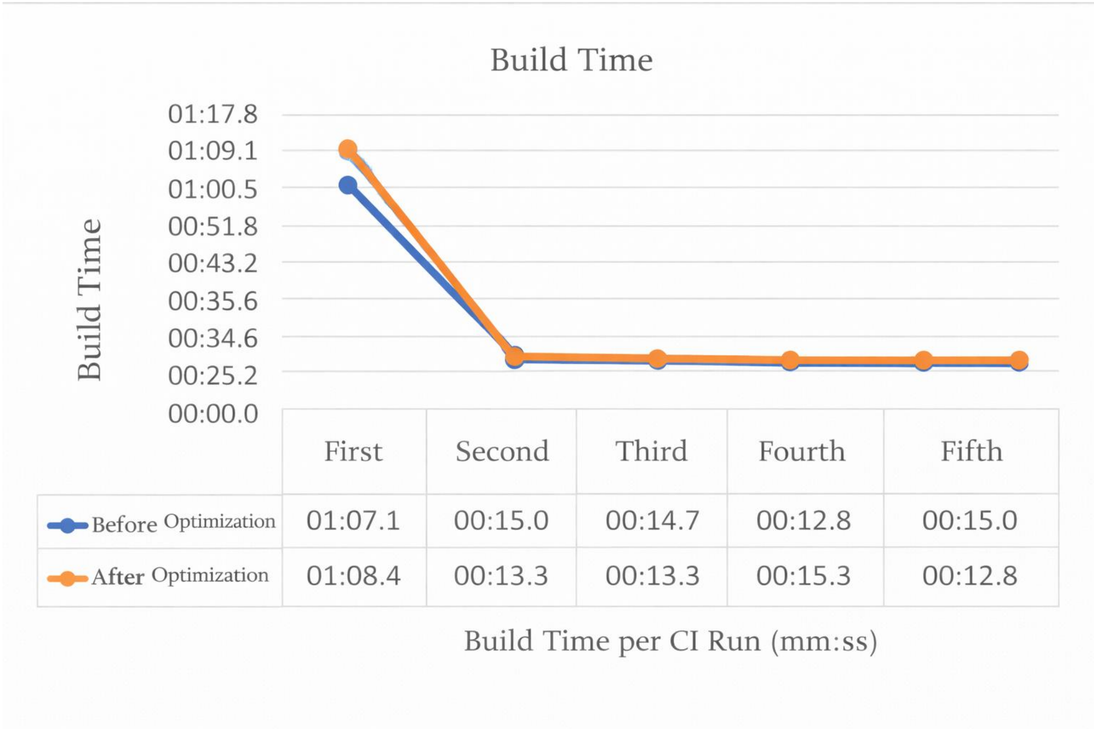
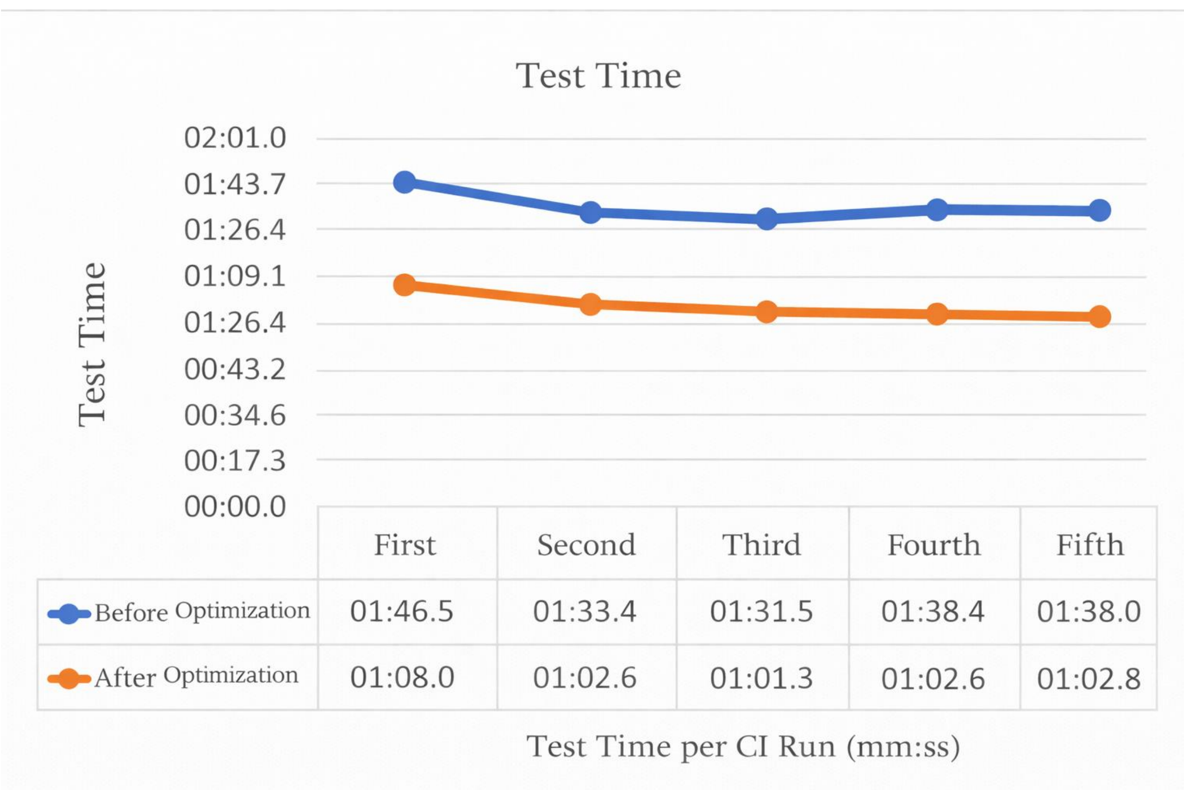



*Figure 6. Number of Runs and Test Time.*



Case Study Results

This section analyzes the impact of implementing the proposed optimization on the company's real continuous integration pipeline deployed in Azure DevOps.

Reduction in Build Time:

Following the analysis of simulations conducted in a virtual machine environment and the evaluation of the obtained results, the proposed optimization strategy was implemented in the CI pipeline after consultation and agreement with the company. By strategically incorporating the --no-restore and --no-build parameters, the objective was to eliminate redundant restore and build processes in order to achieve a faster pipeline in a real production environment.

According to the workflow presented in the literature review, the development process follows a structured sequence. The process begins with a planning phase, during which team members meet to discuss and define the tasks to be completed. Following this discussion, a two-week sprint is initiated. It should be noted that the company employs a hybrid model, using GitHub for version control and Azure DevOps for managing continuous integration and deployment processes.[1]

*Table 1. Comparison of Execution Times of Existing and Optimized Scripts.*

| Run | Unoptimized Script Execution Time | Optimized Script Execution Time |
|---|---|---|
| Run 1 | 03:04.0 | 02:28.6 |
| Run 2 | 01:51.0 | 01:18.7 |
| Run 3 | 01:48.8 | 01:17.4 |
| Run 4 | 01:53.7 | 01:20.8 |
| Run 5 | 01:55.6 | 01:18.1 |
| Average | 02:06.6 | 01:32.7 |
| | 02:06.6 | 01:32.7 |
| Percentage of Execution Time | | 26.77807085 |

Once a developer completes a task, the source code undergoes isolated build/compilation and testing processes. According to records obtained from Azure analytics, the average execution time of the continuous integration process over the past 30 days is approximately 10 minutes (Figure 7). Given the size of the project comprising more than 100 libraries and applications, Based on statistical data, the CI process is expected to take an average of 10 to 20 minutes. It is observed that 42.9% of this time is consumed by the build stage, while the remaining time is allocated to source code retrieval, integration, unit testing, and dependency caching (Figure 8).

*Figure 7. Average Pipeline Execution Time on Azure DevOps.*

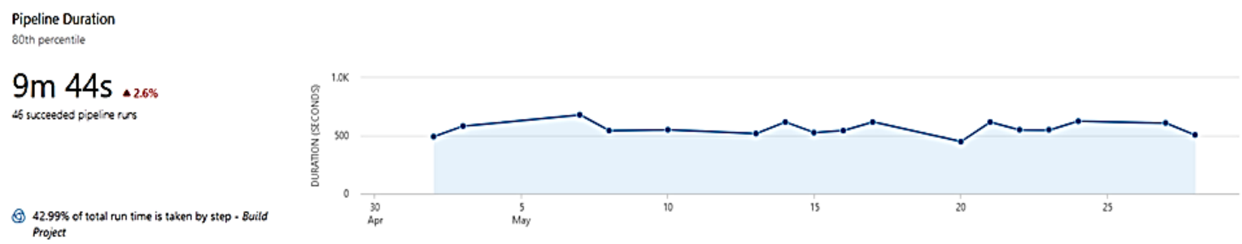
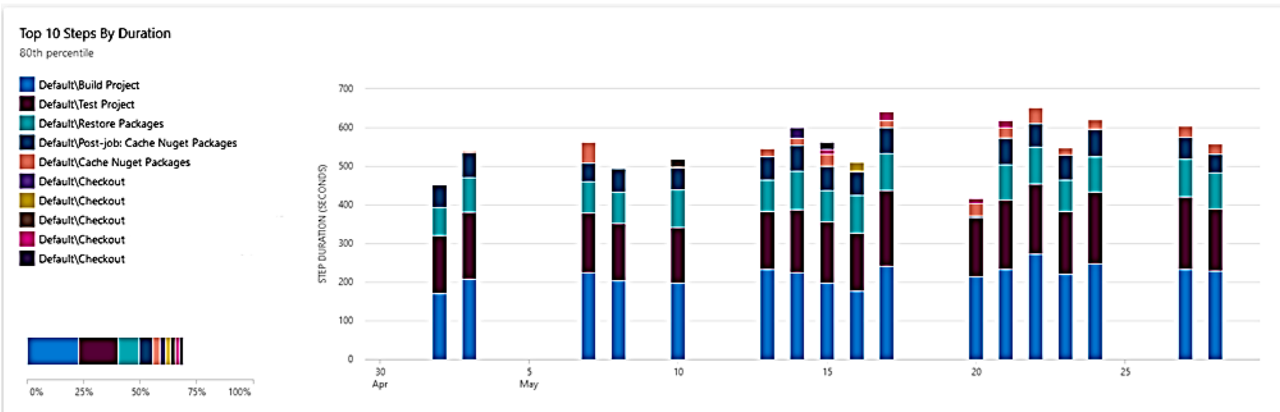
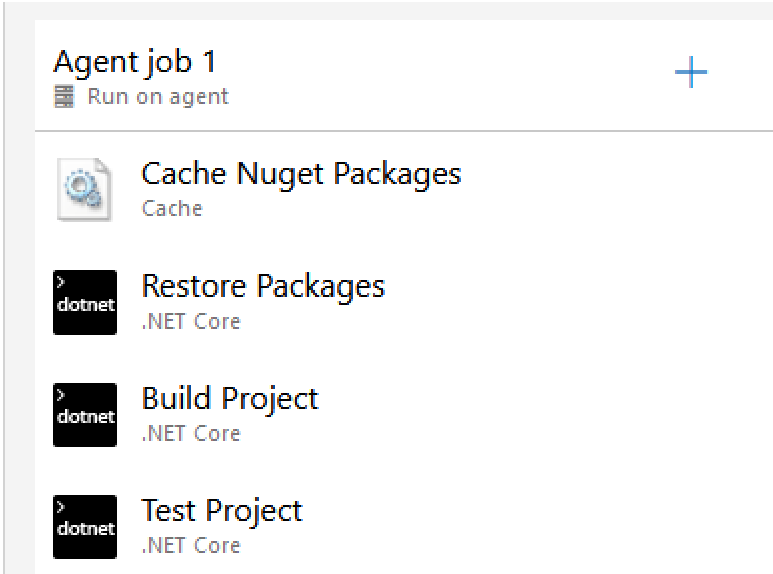



*Figure 8. Pipeline execution times by step in Azure DevOps.*




Based on the findings of our study, we decided to propose CI pipeline optimization to the company, with the specific goal of reducing CI process execution time. To analyze the process in greater detail, the complete workflow of the existing pipeline is considered, including source code retrieval, dependency caching, dependency restoration, project build, and test execution (Figure 9).

*Figure 9. Continuous Integration Stages in Azure DevOps.*




The changes were implemented in agreement with the company through a series of meetings and consultations. Following the introduction of these modifications, the first several pipeline runs resulted in a time savings of approximately 2–3 minutes (Figure 10). To fully assess the broader impact of the applied changes, it is necessary to observe the pipeline performance over the course of the following days.

*Figure 10. Execution Time of Processes After Optimization*

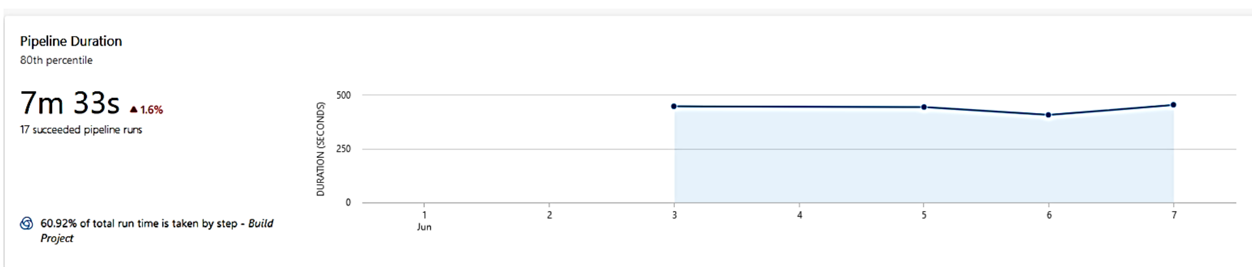
| | | | |
|---|---|---|---|
| windows-2022 | May 30 at 6:50 PM | <1s | 6m 37s |
| windows-2022 | May 30 at 6:39 PM | <1s | 5m 37s |
| windows-2022 | May 30 at 6:30 PM | <1s | 5m 31s |
| windows-2022 | May 30 at 6:22 PM | <1s | 5m 57s |

After a seven-day observation period, Azure analytics indicate that the average execution time decreased from 10 minutes to 7.33 minutes. This reduction suggests that further improvements may be achievable over time. To evaluate the effectiveness of the implemented changes on the CI process, the formula developed in this study is applied as follows:

$$\text{Improvement Percentage} = \frac{[584-453]}{584} * 100 = 22.6\%$$

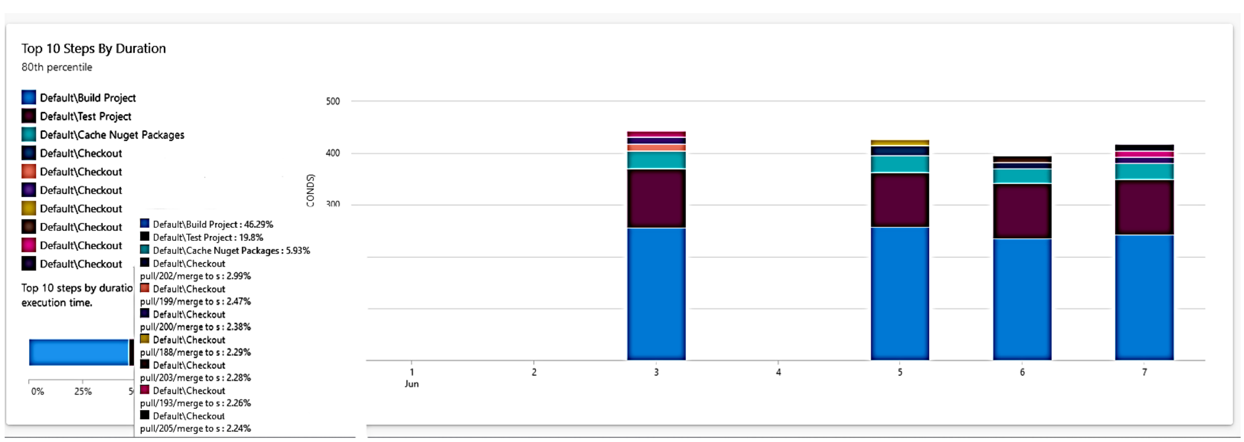
The findings of the case study demonstrate that the applied optimization strategy successfully reduced build time (Figure 11). Following the implementation of the changes, the average execution time of the CI process over the last seven days is presented in Figure 11.

*Figure 11. Average CI Process Execution Time Over the Last 7 Days.*




It is evident that the majority of the execution time is allocated to the build stage (Figure 12). Redundant restore processes were eliminated through the use of the --no-restore and --no-build parameters, as demonstrated in the optimized script example. [1,2]

*Figure 12. Time Distribution of Executed Stages in the CI Process.*




The experimental evaluation demonstrates a clear improvement in CI/CD pipeline performance following the implementation of optimization strategies. Comparative analysis of execution metrics indicates a consistent reduction in average build and integration times across multiple test scenarios. Results obtained from virtual machine experiments show that eliminating redundant restore operations significantly reduces pipeline execution duration. In Azure DevOps pipelines, optimized configurations resulted in shorter build stages and more predictable execution behavior. These findings confirm that systematic pipeline optimization enhances overall CI/CD efficiency while maintaining functional correctness and test coverage.

The findings of this study highlight the critical impact of internal pipeline efficiency on DevOps performance. Redundant operations, often overlooked during pipeline design, introduce unnecessary delays and resource consumption. By addressing these inefficiencies, organizations can achieve measurable performance gains without additional infrastructure investment. The results further indicate that optimization strategies must be context-aware and framework-specific. Generic optimization approaches may yield limited benefits, whereas tailored configurations aligned with the underlying build frameworks and tools produce significant improvements. These observations reinforce the importance of continuous monitoring and iterative optimization in CI/CD environments.[3,4]

## 4. Conclusion

This study presents an empirical investigation into CI/CD pipeline performance optimization through the elimination of redundant operations. The results demonstrate that targeted optimization strategies can substantially reduce execution time and improve software delivery efficiency. The proposed approach offers practical guidance for DevOps teams seeking to enhance pipeline performance while preserving software quality and system reliability.[2] Future research may extend this work by exploring adaptive optimization techniques, chaos engineering experiments, and machine learning-based approaches for dynamic pipeline optimization.

## REFERENCES

1. G. Kuchava, S. Gotsiridze, M. Mantskava, and N. Momtselidze, "Analyzing the performance of DevOps integral components continuous integration/continuous deployment (CI/CD) modules and investigating the functionality and possibilities for software improvement," in *Research Collaboration, Consolidation, Friendship*, pp. 53–54, 2024.
2. S. Gotsiridze, "Analysis of the efficiency of CI/CD modules as integral components of DevOps and opportunities for software improvement," M.S. thesis, Business and Technology University, 2024 (in Georgian).
3. N. Forsgren, J. Humble, P. Debois, J. Willis, and G. Kim, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2021.
4. Z. Kavtaradze, "Increasing software quality and deployment speed through CI/CD in DevOps," M.S. thesis, Business and Technology University, 2024 (in Georgian).
5. N. Demchenko, "Optimizing DevOps performance using chaos engineering," M.S. thesis, Business and Technology University, 2025 (in Georgian).
6. G. Kuchava, G. Dvalishvili, and I. Kartvelishvili, "Optimizing the quality and speed of software delivery in DevOps with CI/CD," Georgian Technical University, pp. 72–78, 2024 (in Georgian).
7. G. Kuchava, I. Kartvelishvili, and S. Vashalomidze, "Using chaos engineering to enhance the resilience of microservice architecture: An analysis of the 'Online Boutique' case," Georgian Technical University, pp. 469–473, 2025 (in Georgian).
8. L. Lwakatare, "DevOps in practice: A multiple case study of five companies," *Information and Software Technology*, 2019.
9. N. Forsgren and E. Freeman, *DevOps for Dummies*. For Dummies, 2019.
10. J. Hyman, *Microsoft Azure for Dummies*. For Dummies, 2023.
11. N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps – Building and Scaling High Performing Technology Organizations*. IT Revolution, 2024.

# Leadership Dynamics as Determinants of Project Success

Nino Lolashvili

*PhD (Engineering Sciences), Professor, Georgian American University, Tbilisi, Georgia.*

***Email:*** *nino.lolashvili@gau.edu.ge*

Anastasia Bajiashvili

*PhD (Engineering Sciences), Professor, Georgian American University, Tbilisi, Georgia.*

***Email:*** *nata.bajiashvili@gau.edu.ge*

## Abstract

The research examines the impact of leadership on project success given the increasingly complex and collaborative frameworks of projects today. While it is imperative to have technical competence and a detailed plan, this study underlines the importance of leadership as one fundamental element of successful outcomes. This research looks at how specific leadership behaviours influence the different aspects of project success such as achievement of goals, delivery on time, budget compliance, stakeholder satisfaction, and most importantly team cohesion.

A meaningful investigation of the research issue was conducted using a quantitative survey of more than 200 project practitioners involved in a variety of international projects using collaborative environments. The survey captured perceptions of leadership behaviours and their associated dimensions of project performance. In addition to descriptive data, the study employed regression analysis in order to identify support for strong relationships between transformational leadership characteristics and high levels of team cohesion and overall project success.

The findings from this article also describe the value of adaptive, motivator, and communicator leadership, in the context of dynamic project environments. This research contributes to the wider understanding of leadership as a strategic resource in project management and offers organizations practical implications to improve project deliverables through excellent leadership.

**Keywords**: Importance of Leadership, Project Success, Team Cohesion, Transformational Leadership

## Introduction

In today's increasingly dynamic and more complex business environment, organizations are even more reliant on project work to meet strategic objectives, enhance innovation, and sustain competitive advantage. Most of the past literature has focused on project management tools, techniques, and methodologies, which are not sufficient for project success. But in a context where organizations are chronically confronted with tighter deadlines, changing priorities, and increasingly cross-disciplinary teams, the human aspect, particularly leadership, becomes an essential dimension for coping with complexity and ultimately ensuring project success. We can bring evidence from a variety of sectors, from local initiatives to international programs, even the most well-resourced and collectively smart projects fail without leadership. Within corporate settings, strong, effective, and consistent leadership is often the distinguishing feature between projects that have an impact and those that do not. This research examines leadership not as a fair-weather influence but as a fundamental feature of success in contemporary project work contexts.

Despite significant developments in project management frameworks with processes where scheduling, budgeting, and risk are managed with increasing company instructions, many projects continue to fall short of their objectives. These frameworks have tended to focus on procedural and technical aspects of projects and have tended not to manage human aspects of projects well, including the most important one - leadership. Leadership is regarded as a supplementary skill, and it is not identified as a core strategic requirement to successfully manage teams, engage with stakeholders, and deal with the complexities of successfully leading organizations. Projects will get done, but fail in critical success factors such as working together in teams or stakeholder engagement and relationships. The gap between leadership practices and project execution represents an important challenge for organizations to succeed within what is becoming an increasingly complex, fast-paced, and competitive business environment. Closing the gap is vital to utilizing the full potential of project-based work and improving the overall performance of the project.

## Materials and Methods

Project success depends not only on tools like Gantt charts but also on leadership—the human engine driving teamwork and accountability (PMI, 2021). Projects are temporary endeavors with unique goals, where success is measured by meeting stakeholder expectations (Bryde, 2005), delivering business value (Zwiebel & Smyrk, 2012), and increasingly, by sustainability and social impact (Gareis et al., 2013). The rise of Agile methods such as Scrum and Kanban (Highsmith, 2009) and the spread of global, virtual teams (Lipnack & Stamps, 2000) highlight the need for adaptable leadership. Leaders are now expected to be strategic motivators rather than administrators (Bass & Avolio, 1994; Crawford & Cooke-Davies, 2005). Effective leadership blends centralized, decentralized, or hybrid coordination (Schwaber & Sutherland, 2020) and prioritizes supportive practices such as empathy, conflict resolution, empowerment, and emotional intelligence (Goleman, 1995; Edmondson, 1999). Success also depends on organizational culture and a leader's ability to shape it (Schein, 2010; Kotter, 2012).

Leadership remains difficult to define, with perspectives ranging from "influence" (Maxwell, 1993) to "mobilizing others for shared aspirations" (Kouzes & Posner, 1995). Summerfield (2014) proposed a simpler view: leadership is "making things better," emphasizing action over position.

Leadership theories have evolved across several eras (King, 1990; Turner, 2009). The Trait School emphasized that leaders are born with qualities such as drive and confidence (Kirkpatrick & Locke; Marshall, 2003), while the Behavioral School focused on balancing structure and relationships, identifying styles such as autocratic, democratic, and laissez-faire (Stogdill &

Coons, 1957; Lewin et al., 1939). The Contingency School introduced the idea that effectiveness depends on context, through models such as Fiedler's theory, Situational Leadership, and Path-Goal Theory (Fiedler, 1967; Hersey & Blanchard, 1969; Turner, 2009). Later, the Visionary School distinguished transactional leaders, who rely on rewards and control, from transformational leaders, who inspire and guide change (Burns, 1978; Bass, 1985; Keegan & Den Hartog, 2004). More recent perspectives include the Emotional Intelligence School, which highlights self-awareness, empathy, and adaptability as key to leadership (Goleman, 1995; Goleman et al., 2002), and the Competence School, which frames leadership effectiveness in terms of measurable competencies across EQ, MQ, and IQ (Turner, 2009; Dulewicz & Higgs). Together, these perspectives illustrate a shift from viewing leadership as innate traits to integrative models that combine behaviors, context, and competencies.

Research confirms leadership's role in project performance. Yang, Huang, and Wu (2010) found both transformational and transactional styles positively influence outcomes, mediated by teamwork. Dainty et al. (2005) identified twelve behavioral competencies crucial for success, while Rehan, Thorpe, & Heravi (2024) highlighted relationship management, self-management, and interpersonal sensitivity.

In IT, Stevenson & Starkweather (2010) and Gruden & Stare (2018) found soft skills such as communication, teamwork, and assertiveness more valued than technical expertise, emphasizing leadership's behavioral dimension.

Future projects will require leaders skilled in cultural intelligence (Earley & Ang, 2003), virtual team management (Hoch & Kozlowski, 2014), and ethical leadership. Research should explore servant leadership (Greenleaf, 1977; Liden et al., 2008), soft skills development, and innovative training such as simulations and mentoring (Yukl, 2012). Greater focus is also needed on micro-level leadership behaviors and informal leadership roles in projects.

The following hypotheses were developed based on established leadership theories and prior empirical studies that demonstrate how leadership behaviors influence team performance and project outcomes.

Hypothesis 1: Leadership should be seen as action-oriented, not role-bound.

Hypothesis 2: The effectiveness of leadership styles is contingent on situational and follower factors.

Hypothesis 3: Leadership theory has evolved from traits toward behaviors, situational context, and competencies.

Hypothesis 4: No single leadership style is universally effective; success depends on context.

This study explores the influence of leadership on project success in international and collaborative projects. A quantitative approach was employed using a structured online survey with 5-point Likert scale questions to capture perceptions of leadership effectiveness and project outcomes. Purposive sampling targeted 211 experienced project professionals across various roles, industries, team sizes, and project durations to provide diverse insights (Etikan, Musa, and Alkassim, 2016).

Data collected covered demographics, leadership behaviors, and project success factors. Responses were analyzed using descriptive statistics, ANOVA, and regression modeling to examine relationships between leadership practices and project outcomes. Descriptive statistics summarized participant characteristics and overall perceptions, ANOVA tested differences across demographic groups, and regression predicted outcomes like motivation, team effectiveness, and stakeholder satisfaction based on leadership behaviors.

The survey included 25 questions: demographics (Q1–6), leadership perceptions (Q7–15), project complexity and context (Q16–20), project performance (Q21–22), and interpersonal/communication skills (Q23–25). Questions were linked to four hypotheses concerning leadership vision, style effectiveness, and evolving leadership theory (see Table 2).

The study ensured anonymity, informed consent, and voluntary participation. Ethical principles followed Kant's (2001) categorical imperative, treating participants as ends in themselves, enhancing the study's reliability and trustworthiness.

## Results and Discussion

This part presents results from the survey examining leadership's role in project success across international and collaborative contexts. Transformational and participative leadership styles were found to significantly enhance project performance, team cohesion, and goal achievement. Leadership effectiveness varies by demographics, role, and experience, as supported by ANOVA and regression analyses.

Multiple regression showed that leadership behaviors and project characteristics explained 39.1% of variance in team motivation ($R = 0.626$, $F = 18.66$, $p < 0.001$). The strongest positive predictor was Encouraging Creative Thinking ($\beta = 0.232$, $p < 0.001$), followed by Stakeholder Diversity ($\beta = 0.207$, $p < 0.001$) and Technical Complexity ($\beta = 0.135$, $p = 0.002$). Variables like developing team strengths, prioritizing well-being, and building relationships were not significant. These findings suggest intellectual stimulation and diverse, challenging projects drive motivation, whereas relational or well-being-focused behaviors are less directly motivational.

Regression analysis ($R^2 = 0.380$, $F = 25.09$, $p < 0.001$) indicated that Project Progress ($\beta = 0.426$, $p < 0.001$) was the strongest predictor of stakeholder satisfaction, followed by Communication Skills ($\beta = 0.138$, $p = 0.030$), Technical Complexity ($\beta = 0.121$, $p = 0.020$), and Stakeholder Diversity ($\beta = 0.118$, $p = 0.040$). Task efficiency had a marginal effect. Leadership and project characteristics collectively influence stakeholder satisfaction, with progress and communication being key drivers.

Regression results ($R^2 = 0.260$, $F = 14.43$, $p < 0.001$) showed that Managing Interpersonal Relationships ($\beta = 0.220$, $p = 0.01$) and Fostering Collaboration ($\beta = 0.178$, $p = 0.03$) significantly improved team cohesion. Other behaviors, including prioritizing well-being, supporting individuals, or passive intervention, were not significant. These results emphasize the importance of proactive, relational, and collaborative leadership for cohesive and effective teams.

Based on the results we can say that leadership behaviors—particularly those encouraging creativity, fostering collaboration, and managing relationships—combined with project factors such as complexity and stakeholder diversity, are key drivers of motivation, stakeholder satisfaction, and team cohesion in international projects.

Overall, this research adds considerable evidence to the importance of leadership in terms of project management and project success in both international and collaborative work environments. Across all three models, team motivation, stakeholder satisfaction and team cohesion, leadership behaviors demonstrated statistical significance. Specifically, inspiring creative thinking, engaging with diverse stakeholders, and changing how people engage with technical complexity are three significant factors having a positive influence on team motivation. For stakeholder satisfaction, nothing had a larger impact than a project's timeline and effective communication. Team cohesion is significantly influenced by leadership behaviors in more general ways, aligning with other literature to understand leadership as a focus on relationships alongside task-oriented leadership. Overall, the variances explained in each population (39.1 percent for motivation, 38 percent for stakeholder satisfaction; and 26 percent for cohesion) demonstrate moderate to strong explanatory power in terms of leadership and project-specific variables, and the overall importance of leadership when leading and managing teams, especially in complex, multicultural projects. These findings add some reassurance as to the importance of leadership in project management and the requirement for improving the leadership skill set required for any given project to be successful.

## Conclusion

This study examined the roles of leadership behaviors and their relationship to project success, particularly with respect to motivating and creating cohesive teams in international and collaborative projects. The research was based on data from 211 participants and adopted a transformational leadership framework.. The findings show that leadership behaviors like stimulating creative thought, handling the technical complexities of projects, and managing the relationships of various stakeholders have potential in improving team performance and engagement.

On the other hand, other supportive behaviors, such as modeling concern for the team, or developing the strengths of individuals, rarely contributed to key project outcomes, which indicates that, in settings that are fast-paced and high-stakes, team members pay more attention to leaders who provide intellectual stimulation or strategic direction, than who provide only emotional support.

This study raises questions around adaptive leadership, where inspiring innovation, building inclusive teams, and leading through complexity are heightened leadership demands. The findings from this study provide important considerations for leadership development and project design and should work toward developing a cadre of leaders that inspire, stimulate, and lead teams made up of diverse members toward success.

Based on the research findings, it is recommended that leadership development programs in international and collaborative project settings emphasize skills that foster creativity, adaptability, and inclusivity. Leaders should be trained to stimulate innovative thinking within teams and to embrace technical challenges and stakeholder diversity as opportunities for motivation and cohesion. Organizations should prioritize assigning leaders who can manage complexity and facilitate dynamic team environments, rather than focusing solely on interpersonal or emotional support behaviors. Additionally, project design should incorporate elements that intellectually engage team members and encourage collaboration across diverse groups. Future leadership strategies should adopt a situational approach, enabling leaders to adjust their style based on project demands and team composition for optimal outcomes. Finally, institutions managing youth, Erasmus+, or globally distributed teams should consider these leadership factors when planning training or assigning responsibilities to maximize team engagement and overall project success.

## REFERENCES


Bass, B. M. (1985). *Leadership and performance beyond expectations*. Free Press.

Bass, B. M., & Avolio, B. J. (1994). *Improving organizational effectiveness through transformational leadership*. Sage.

Bryde, D. J. (2005). Methods for managing project success. *International Journal of Project Management, 23*(3), 182–190

Burns, J. M. (1978). *Leadership*. Harper & Row.

Crawford, L., & Cooke-Davies, T. (2005). Project management in the contemporary organization. *Project Management Journal, 36*(2), 5–15.

Dainty, A., Cheng, M., & Moore, D. (2005). Competencies of construction project managers. *Journal of Construction Research, 6*(1), 1–12.

Edmondson, A. (1999). Psychological safety and learning behavior in work teams. *Administrative Science Quarterly, 44*(2), 350–383. https://doi.org/10.2307/2666999

Earley, P. C., & Ang, S. (2003). *Cultural intelligence: Individual interactions across cultures*. Stanford University Press.

Fiedler, F. E. (1967). *A theory of leadership effectiveness*. McGraw-Hill.

Field, A. (2013). *Discovering statistics using IBM SPSS statistics* (4th ed.). Sage.

Gareis, R., Huemann, M., & Martinuzzi, A. (2013). Project management and sustainable development. *International Journal of Project Management, 31*(6), 761–771.

Geoghegan, L., & Dulewicz, V. (2008). Do project managers' leadership competencies contribute to project success? *Project Management Journal, 39*(4), 58–67.

Goleman, D. (1995). *Emotional intelligence*. Bantam.

Goleman, D., Boyatzis, R., & McKee, A. (2002). *Primal leadership: Realizing the power of emotional intelligence*. Harvard Business School Press.

Greenleaf, R. K. (1977). *Servant leadership: A journey into the nature of legitimate power and greatness*. Paulist Press.

Hersey, P., & Blanchard, K. H. (1969). Life cycle theory of leadership. *Training and Development Journal*, *23*(5), 26–34.

Highsmith, J. (2009). *Agile project management: Creating innovative products* (2nd ed.). Addison-Wesley.

Hoch, J. E., & Kozlowski, S. W. (2014). Leading virtual teams: Hierarchical leadership, structural supports, and shared team leadership. *Journal of Applied Psychology, 99*(3), 390–403.

Keegan, A., & Den Hartog, D. (2004). Transformational leadership in a project-based environment: A comparative study. *International Journal of Project Management, 22*(8), 609–617.

King, A. W. (1990). Leadership theories and historical perspectives. *Journal of Leadership Studies, 2*(2), 45–57.

Kirkpatrick, S., & Locke, E. A. (1991). Leadership: Do traits matter? *Academy of Management Executive, 5*(2), 48–60. https://doi.org/10.5465/ame.1991.4274679

Kotter, J. P. (2012). *Leading change*. Harvard Business Review Press.

Lewin, K., Lippitt, R., & White, R. K. (1939). Patterns of aggressive behavior in experimentally created social climates. *Journal of Social Psychology, 10*, 271–299. https://doi.org/10.1080/00224545.1939.9713366

Lipnack, J., & Stamps, J. (2000). *Virtual teams: Reaching across space, time, and organizations*. John Wiley & Sons.

Marshall, G. (2003). *The concise Oxford dictionary of sociology* (3rd ed.). Oxford University Press.

Maxwell, J. C. (1993). *Developing the leader within you*. Thomas Nelson.

Pallant, J. (2020). *SPSS survival manual* (7th ed.). McGraw-Hill.

Project Management Institute. (2021). *A guide to the project management body of knowledge (PMBOK® guide)* (7th ed.). Author.

Rehan, R., Thorpe, R., & Heravi, G. (2024). Leadership competencies for successful international projects. *International Journal of Project Management, 42*(1), 15–28.

Schwaber, K., & Sutherland, J. (2020). *The Scrum guide*. Scrum.org.

Schein, E. H. (2010). *Organizational culture and leadership* (4th ed.). Jossey-Bass.

Stevenson, M., & Starkweather, J. (2010). Leadership behaviors and project success. *Project Management Journal, 41*(2), 38–49.

Turner, R. (2009). *Handbook of project-based management* (3rd ed.). McGraw-Hill.

Turner, R., & Müller, R. (2005). The project manager's leadership style as a success factor on projects: A literature review. *Project Management Journal, 36*(2), 49–61. https://doi.org/10.1177/875697280503600206

Yang, L., Huang, C., & Wu, K. (2010). The association among project manager leadership style, teamwork, and project success. *International Journal of Project Management, 28*(3), 265–277.

Zwiebel, M., & Smyrk, J. (2012). Project success: A value-based perspective. *Project Management Journal, 43*(6), 74–86.

# Performance and Efficiency Evaluation of GitOps-Based Deployment Workflows

Giorgi Tseradze

*MSc (Information Systems), Business and Technology University, Tbilisi, Georgia.*

***Email:*** *giorgi.tseradze.1@btu.edu.ge*

Giorgi Kuchava

*PhD (Engineering of Informatics), Associate professor, Georgian Technical University, Tbilisi, Georgia.*

***Email:*** *kuchavagiorgi08@gtu.ge*

Teimuraz Sturua

*Candidate of Technical Sciences, Associate professor, Georgian Technical University, Tbilisi, Georgia.*

***Email:*** *t.sturua@gtu.ge*

Ana Khinchikashvili

*BSc (Information Technologies), Business and Technology University, Tbilisi, Georgia.*

***Email:*** *ana.khinchikashvili.1@btu.edu.ge*

Ekaterine Somkhishvili

*BSc (Information Technologies), Assistant, Business and Technology University, Tbilisi, Georgia.*

***Email:*** *ekaterine.somkhishvili.1@btu.edu.ge*

Ioseb Kartvelishvili

*Candidate of Technical Sciences, Professor, Georgian Technical University, Tbilisi, Georgia.*

***Email:*** *s.kartvelishvili@gtu.ge*

Maka Mantskava

*PhD (Biology), Professor, European University, Tbilisi, Georgia.*

***Email:*** *maia.mantskava@eu.edu.ge*

Shalva Chokhonelidze

*MSc (Computer Science), Senior Scientist, I. Beritashvili Center of Experimental Biomedicine, Tbilisi, Georgia.*

***Email:*** *tchokhonelidze.shalva@atsu.edu.ge*

Nana Momtselidze

*PhD (Biology), Senior Scientist, I. Beritashvili Center of Experimental Biomedicine, Tbilisi, Georgia.*

***Email:*** *nana.momtselidze@eu.edu.ge*

**Abstract**

GitOps represents a modern approach to infrastructure and software deployment that leverages Git to enable continuous, automated, and controlled operational processes. This study examines the core functional mechanisms of GitOps, including declarative configuration, the reconciliation loop, and the pull-based deployment model. Special attention is given to Argo CD and Flux as leading GitOps tools, with an emphasis on their integration with Kubernetes. The paper analyzes real-world operational workflows, architectural components, and security mechanisms associated with GitOps-based systems. The efficiency of GitOps is investigated across diverse application domains, ranging from technology startups to highly regulated industries, including banking and healthcare. A comparative analysis is conducted between GitOps-based approaches and traditional CI/CD models. The concluding section outlines future perspectives for the evolution and expansion of GitOps, including standardization efforts, scalability challenges, and the integration of emerging technologies such as artificial intelligence, serverless architectures, and edge computing. This work serves as a practical guideline for organizations seeking to adopt GitOps practices and improve the efficiency and reliability of their operational processes [1].

**Keywords:** GitOps, CI/CD, Kubernetes, Argo CD, Flux, DevOps, YAML

## 1. Introduction

GitOps represents a modern approach to the automated management of infrastructure and software systems, with Git serving as the single source of truth. This paradigm is built upon tools and processes already familiar to developers, such as Git repositories, pull requests, and code review mechanisms. The primary objective of GitOps is to minimize the boundary between development and operations practices while enhancing the transparency, traceability, and reliability of infrastructure management processes. Historically, GitOps was first popularized by Weaveworks in 2017 as a response to the challenges associated with rapid service deployment and management in cloud-native environments. Weaveworks introduced GitOps as an "operations model driven by Git," establishing a new standard based on declarative configurations stored in Git repositories and continuous reconciliation between the desired state defined in Git and the actual state of the cluster.

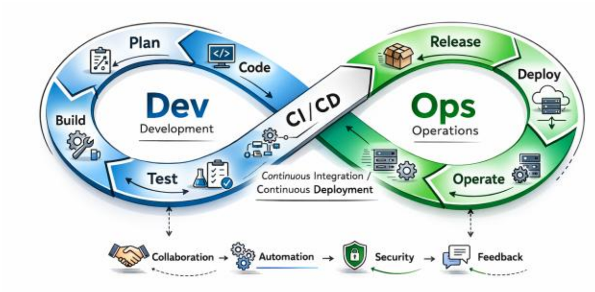
The GitOps paradigm builds upon earlier concepts such as Infrastructure as Code (IaC); however, unlike traditional IaC practices, GitOps tightly integrates IaC with continuous integration and continuous deployment (CI/CD) workflows. In this model, every infrastructure change is executed through a Git pull request, enabling comprehensive version control, historical traceability, and auditability not only for application code but also for infrastructure configurations. At present, GitOps is most widely adopted within Kubernetes-based environments; nevertheless, its underlying principles are technology-agnostic and can be adapted to other systems and platforms. Ongoing technological advancements—including the maturation of CI/CD tooling and the widespread adoption of declarative infrastructure—have facilitated the integration of GitOps practices across a broad range of industries, from early-stage technology startups to large-scale global enterprises.

Prior to the emergence of GitOps, the technology landscape had already undergone significant transformation through the adoption of DevOps and CI/CD methodologies. These approaches arose as responses to long-standing challenges, including inadequate communication between development and operations teams, unstable deployment processes, and limited capability for rapid software delivery. In traditional operational models, developers produced code that was subsequently handed off to operations teams, often resulting in conflicts, delays, and systemic inefficiencies. The DevOps movement (fig. 1) addressed these issues by promoting the integration of development and operations workflows, shared responsibility, and the adoption of continuous delivery practices.

Complementing this cultural shift, CI/CD mechanisms were introduced to automate code testing, build processes, and deployment pipelines, thereby improving software quality, delivery speed, and operational consistency. Over time, the demand for container orchestrators such as

Kubernetes increased, as these platforms manage containerized environments and enable the definition of an application's desired state. However, it was precisely within this cloud-native ecosystem that the need emerged for a more structured, version-controlled, and transparent deployment model, which subsequently led to the development of GitOps.

*Figure 1. DevOps Lifecycle*




DevOps, as both a cultural and technical practice, aims to improve coordination between development (Dev) and operations (Ops) teams. Its core principles include continuous integration, rapid release cycles, infrastructure automation, and transparent collaboration. GitOps represents a logical extension of DevOps and one of its practical implementations, building upon the same foundational values while reinforcing them through Git-managed infrastructure. Whereas DevOps constitutes a broader philosophy encompassing organizational culture, processes, and tooling, GitOps focuses on the practical realization of these principles within cloud-native and container-based environments. It operationalizes DevOps concepts through the adoption of concrete technical standards, such as declarative configurations, automated reconciliation, and pull-based deployment mechanisms.

GitOps incorporates the following fundamental components of the DevOps paradigm:

Infrastructure as Code (IaC): GitOps builds upon the IaC approach while extending its capabilities by ensuring that all infrastructure changes are fully version-controlled within Git repositories. In this model, developers manage infrastructure using the same mechanisms applied to application code management, such as pull requests, version history, and code review workflows.

Continuous Delivery (CD): GitOps enhances continuous delivery practices by ensuring that deployments are triggered exclusively by validated changes committed to Git repositories. In this approach, deployment occurs only when the repository state has been verified, thereby improving deployment reliability and traceability.

Collaboration and Review: GitOps assigns the same level of importance to operational workflows as to code review processes, ensuring that infrastructure changes are subject to pull request–based review. This approach enhances transparency, accountability, and overall system quality.

GitOps represents a significant advancement in that Infrastructure as Code is no longer confined to system administrators or DevOps engineers but becomes a shared operational workflow for developers as well. As a result, GitOps enables a developer-centric DevOps environment in which service management, updates, and monitoring are performed through familiar and integrated toolchains. In today's landscape, where many organizations are transitioning to Kubernetes-based infrastructures and deployment environments are becoming increasingly complex, GitOps has emerged as an integral component of DevOps strategies. It supports teams in achieving core objectives, including accelerated innovation, operational stability, automation, and enhanced security.

*GitOps and Infrastructure as Code (IaC)*

Infrastructure as Code (IaC) is a practice in which infrastructure is defined and managed in the form of code. The primary objective of IaC is to ensure that system configuration, deployment, and operational processes are executed using the same principles and tools applied in application development, including version control, code review, testing, and automation. This approach enables organizations to manage complex infrastructures more rapidly and efficiently.

GitOps is a practice built upon this conceptual foundation but represents a more advanced and structured evolution of IaC. Both approaches rely on a declarative model, in which the desired state of the system is specified rather than the procedural steps required to reach that state. GitOps strengthens this model in several key ways:

Git as the Single Source of Truth: While IaC practices may involve a variety of tools and code artifacts, GitOps consolidates all infrastructure definitions within a single platform, designating Git as the authoritative source of truth for all infrastructure resources.

Push vs. Pull Deployment Model: In traditional IaC workflows, infrastructure updates are often applied using push-based mechanisms, where scripts or tools such as Terraform, Chef, or Ansible actively enforce changes. GitOps adopts a pull-based model, in which an agent (e.g., Argo CD or Flux) continuously compares the desired state stored in Git with the actual system state and performs reconciliation as needed.

Version Control and Auditability: GitOps provides a comprehensive audit trail for infrastructure changes, precisely documenting who introduced a change, when it was made, and for what purpose. While IaC may partially support such traceability, GitOps enforces it in a systematic and consistent manner.

Simplified Rollback Mechanisms: Owing to Git's inherent version history, GitOps enables straightforward rollback operations. Reverting the system to a previous state is as simple as restoring an earlier commit within the Git repository.

In summary, although GitOps and IaC are closely intertwined and often discussed together, GitOps can be regarded as an evolutionary extension of IaC. IaC provides the foundational principles, whereas GitOps builds upon this foundation to deliver an integrated, automated, and secure operational model for modern cloud-native environments [2].

## 2. Methodology

Following the overview of GitOps functionality, it is essential to analyze its practical effectiveness in order to determine the extent to which it fulfills its intended purpose within real-world engineering, operational, and strategic contexts. Although GitOps can be successfully implemented from a technical perspective, its overall effectiveness is influenced by factors such as architectural complexity, team capabilities, and organizational culture. The objective of the effectiveness analysis is to assess whether GitOps exerts a positive impact on system stability, the speed of change implementation, error reduction, the facilitation of team collaboration, and the optimization of operational costs in practice. To address these research questions, a mixed-methods research methodology is employed, combining both quantitative and qualitative analyses. The quantitative analysis is based on well-defined DevOps metrics, which enable objective measurement of outcomes, including the following: Deployment Frequency, Lead Time for Changes, Mean Time to Recovery (MTTR), Change Failure Rate, Operational Cost Variation, Developer Productivity Metrics. The qualitative analysis encompasses interviews, observational data, and feedback from development and operations teams, enabling the assessment of not only technical aspects but also the influence of human and organizational factors on the effectiveness of GitOps. The following primary criteria are used to assess GitOps effectiveness (Table 1):

*Table 1. Primary DevOps Metrics and Descriptive Definitions*

| Criteria | Description |
|---|---|
| Deployment Frequency | The number of successful deployments conducted within a given timeframe. |
| Lead Time for Changes | The commit-to-deployment duration for implementing a specific change. |
| Change Failure Rate | The percentage of deployments that result in production incidents. |
| Mean Time to Recovery (MTTR) | The average time taken to restore system functionality following a failure. |
| Operational Cost Variation | The change in operational costs compared to a baseline period. |
| Developer Productivity Metrics | Indicators of the team's productivity and responsiveness to operational challenges. |

Based on these criteria, GitOps is evaluated with consideration of technical, process-oriented, and human aspects. The adoption of GitOps across various industries presents distinct challenges and advantages, as its effectiveness depends on factors such as organizational size, technological architecture, and regulatory requirements.

In the financial sector, banks and insurance companies particularly value GitOps for its capabilities in change auditing and security. Git commit–based control mechanisms ensure transparency and support regulatory compliance, while automated rollback processes reduce the risk of downtime and support business continuity. Small and medium-sized technology startups leverage GitOps to automate DevOps processes and accelerate development cycles; however, limited resources and insufficient expertise often hinder full-scale adoption. Large enterprises adopt GitOps as a strategic tool for infrastructure standardization and multi-cluster management. In documented cases, enterprises supported by technical consulting have successfully addressed challenges related to Argo CD and single sign-on integration, resulting in more stable deployment processes.

GitOps is particularly effective in cloud-native environments, where configurations are managed using declarative formats such as YAML and Kubernetes simplifies tool integration. In contrast, adoption in legacy infrastructures may require additional abstraction layers and process reengineering, increasing implementation complexity and cost. The benefits therefore vary depending on organizational context: enhanced compliance in finance, flexibility in startups, governance and standardization in enterprises, and technical efficiency in cloud-native infrastructures. Nevertheless, effective adoption in all cases requires alignment with team capabilities, technological architecture, and organizational culture.

To assess overall effectiveness, GitOps must be compared with traditional CI/CD approaches. Although both aim to enable rapid and reliable deployment, their architectural principles and levels of automation differ significantly [2,3]. The use of internationally recognized DORA metrics—deployment frequency, lead time for changes, change failure rate, and mean time to recovery (MTTR)—provides an objective basis for comparing GitOps-based workflows with traditional deployment models (Table 1).

*Table 2. A Comparative Analysis of GitOps and Traditional CI/CD Using Core DevOps Metrics*

| DevOps Metrics | Traditional CI/CD | GitOps |
|---|---|---|
| **Deployment Frequency** | Deployment, typically manual, occurring 1–2 times per week in many organizations. | Deployment frequency generally higher, facilitated by automation through Git-based pipelines. |
| **Lead Time for Changes** | Requires complex approval chain, often involving multiple steps before deploying changes to production. | commit-to-deploy much faster by using declarative manifests and Git review automation. |
| **Change Failure Rate** Manual deployment processes and complex pipelines often increase the likelihood of production errors. | Manual deployment processes and complex pipelines often increase the likelihood of production errors. | Declarative YAML configuration, Git review and automation reduce likelihood of deployment failures. |
| **Mean Time to Recovery (MTTR)** | Rollback processes often require manual intervention and configuration of CI pipeleline components. | rollback through Git commit-based approaches that restore previous configurations automatically. |

Comparison with Traditional CI/CD: Traditional CI/CD approaches often require the integration of multiple tools, complex approval chains, and active involvement from operations teams. In many organizations, deployments are typically performed once or twice per week, while rollback procedures are frequently non-standardized and inefficient. In contrast, GitOps automates the entire process by applying changes directly from Git commits to the cluster through declarative YAML manifests, enabling deployments to be executed multiple times per day with minimal manual intervention. Comparative tables clearly demonstrate the advantages of GitOps across key metric indicators. GitOps reduces lead time from commit to deployment, decreases the number of failures due to the deterministic nature of declarative configurations, accelerates rollback through Git-based version restoration, and enhances overall process transparency and consistency. Comparison with Script-Based and Manual Deployment Approaches: In many small or traditionally structured teams, deployments are still performed using Bash scripts, command-line interfaces (CLI), or graphical tools. Such approaches require a high level of manual involvement, increase the risk of human error, and complicate rollback procedures (Table 3) and auditability. By contrast, GitOps enables developers to focus exclusively on Git-based workflows, while deployment is fully automated through dedicated agents. This shift reduces operational overhead and enhances overall productivity by minimizing manual intervention and standardizing deployment processes. The comparative data indicate that GitOps provides greater predictability, reduced error rates, and a more streamlined approach to propagating changes. The use of YAML-defined system states, version control, and built-in auditability contributes to stable and reliable IT operations [4].

*Table 3. A Comparative Analysis of GitOps and Script/Manual Deployment with Respect to Productivity and Security*

| Criteria | Script/Manual Deployment | GitOps |
|---|---|---|
| Deployment Process | Script-based through CLI commands and tools. | YAML configurations and version control |
| Ease of Rollout | Prone to human error with manual steps in deployment. | commit-to-cluster automated rollout using declarative manifests. |
| Rollback Efficiency | Rollback often complex, requiring manual intervention. | rollback = Git commit-based version restoration. |
| Auditability and Consistency | Limited audit trails and consistency checks due to decentralized management. | Versioned declarative configurations provide full audit trail and consistency. |

Before-and-After Analysis Within the Same Organization: For a reliable comparison, one of the most robust approaches is to analyze organizational performance metrics before and after the adoption of GitOps within the same organizational context. Case studies from Intuit and Codefresh demonstrate that the introduction of GitOps led to substantial reductions in MTTR, failure rates, and deployment time. Intuit: MTTR was reduced from approximately 45 minutes to less than 5 minutes, while deployment cycles decreased from days to minutes. Codefresh: Improvements were observed in code quality, deployment frequency increased, and overall system downtime was reduced [5].

*Table 4. Evaluating the Effects of GitOps on MTTR, Deployment Frequency, and System Stability: Insights from Intuit and Codefresh*

| Organization | Metric | Before GitOps | After GitOps |
|---|---|---|---|
| **Intuit** | MTTR (Mean Time to Recovery) | ~45 minutes | < 5 minutes |
| | Deployment cycle duration | From days | To minutes |
| **Codefresh** | Failures per deployment day | ~24 hour mean time | Significantly reduced |
| | Failures per deployment day | Irregular, about twice per week | 40%+ increase |
| | Deployment frequency | Irregular, about twice per week | 40%+ increase |
| **Codefresh** | Occasional outages and instability | Occasional outages and instability | Markedly reduced |

Developer and Operations Team Experience with GitOps: GitOps enables developers to manage infrastructure changes directly through pull request–based workflows, thereby increasing control and reducing dependency on operations teams. The versioning of changes in Git, the availability of visual feedback through tools such as the Argo CD user interface, and the use of YAML within familiar Git-centric processes collectively lower the learning barrier and facilitate adoption. The primary positive characteristics include: Improved change control: Every modification is preceded by peer review, which increases accountability and reduces the likelihood of unintended changes. Reduced dependency on operations teams: Developers manage deployments directly through Git-based workflows, thereby decreasing waiting times and operational bottlenecks. Predictability and simplified rollback: Restoring the system to a previous state can be achieved by reverting to an earlier Git commit, enabling rapid recovery from failures. According to data reported by Weaveworks, following the adoption of GitOps, 75% of developers reported increased change security and improved process transparency. GitOps and Onboarding Efficiency: GitOps accelerates the onboarding process by providing versioned, standardized environments in which documentation is embedded directly within declarative YAML configurations. New team members leverage Git repository history, prior pull requests, and predefined examples as learning resources, enabling faster contextual understanding of both infrastructure and operational workflows. According to a 2024 study by Humanitec, teams adopting GitOps experienced an average 35–45% (Table 5) reduction in onboarding time, with the first pull request typically submitted within 1.8 days [6].

*Table 5. Technical Process Comparison: Traditional Deployment Models versus GitOps*

| Metric | Traditional Onboarding | GitOps Onboarding |
|---|---|---|
| Onboarding duration | 7–10 days | 3–5 days |
| Fully productive duration | 2–3 days | < 1 day |
| First Pull Request submission | 3–4 days | 1–2 days |
| Key onboarding resources | Internal documentation | Git history, pre-defined examples (Recent PRs) |

Reduction of Operations Team Workload: GitOps facilitates the transformation of operations (Ops) team responsibilities by shifting the focus away from routine incident handling and configuration-related issues toward strategic platform improvement. Since deployments are executed exclusively based on changes committed to Git repositories, response times are reduced, configuration errors are minimized, and uncoordinated operational interventions are significantly decreased. According to the Accelerate State of DevOps Report, high-performing teams adopting GitOps or similar disciplines achieved, on average, a 2.5-fold reduction in change lead time and an almost threefold decrease in failure rates. Furthermore, a 2024 report by Octopus Deploy indicates that 63% of organizations reported improved deployment stability and a reduction in incidents following the adoption of GitOps.

Evaluating the GitOps concept solely through theoretical discussion is insufficient to fully capture its practical value. A more comprehensive understanding emerges through the analysis of real-world implementation outcomes. This section presents a case study of a medium-sized software development company in Georgia, where the adoption of GitOps fundamentally transformed both infrastructure management practices and the dynamics of development workflows. Prior to GitOps adoption, infrastructure management and application deployment within the organization were conducted using semi-manual processes, resulting in frequent incidents, configuration errors, and communication breakdowns between teams. Systemic complexity—stemming from the simultaneous management of legacy and modern services deployed across heterogeneous infrastructure environments—further increased the likelihood of human error and hindered process standardization.

The primary objective of this research is to analyze the impact of GitOps adoption on organizational and technical dynamics within a specific Georgian company. The study focuses on the transformative changes introduced by GitOps implementation, ranging from reductions in failure rates to improvements in team communication and operational transparency. The core research questions address prior challenges, phased implementation strategies, technical and organizational changes, and the evolution of team workflows and responsibility distribution. The scope encompasses both technical infrastructure transformation—such as YAML structuring, Argo CD adoption, and pull-based deployment processes—and human factors, including onboarding practices and change management culture [7].

The case study employs a mixed-methods research approach combining quantitative data analysis with qualitative interviews. Interviews were conducted with five employees representing key roles, including DevOps, development, project management, and technical leadership. Additional data sources included Argo CD synchronization logs, Git commit histories, CI/CD pipeline records, incident archives, and GitHub pull request metrics [8]. Documentation updates and YAML configuration changes were also analyzed. Data collection was conducted between September 2024 and February 2025, following the completion of GitOps implementation in October 2024.

Within several months of adoption, measurable improvements were observed across multiple dimensions of operational efficiency. Evaluation was conducted using predefined metrics designed to compare traditional deployment models with the GitOps-based approach (Table 6).

*Table 6. Comparative Analysis of Key Operational Metrics Before and After GitOps Adoption*

| Metric | Description | Before GitOps | After GitOps | Data Source |
|---|---|---|---|---|
| **Deployment Success Rate** | Percentage of successful deployments relative to total attempts | **76%** | **94%** | CI/CD logs |
| **Mean Time to Recovery (MTTR)** | Average time required to restore service after failure | **2.5 hours** | **30 minutes** | Incident tracking system |
| **Configuration Drift Incidents** | Number of detected configuration inconsistencies between desired and actual state | **15 incidents** | **2 incidents** | Argo CD audit logs |
| **Rollback Duration** | Time required to revert a failed deployment to a stable state | **~1 hour** | **~5 minutes** | Git commit history |
| **Pull Request Review Time** | Average time required to review and approve pull requests | **45 minutes** | **~1 minute** | GitHub Insights |
| **New Team Member Onboarding Time** | Time required for a new developer to become operationally productive | **2.8 working days** | **~1 working day** | Internal organizational metrics |

*Key Findings*

Improved predictability: System changes are reviewed in advance, semantically defined, and executed exclusively through Git commits, ensuring controlled and reproducible deployments. Reduction in incidents: The number of configuration drifts decreased significantly, indicating improved synchronization between the desired and actual system states. Faster rollback: Previously, rollback procedures required both code inspection and direct system intervention; under GitOps, restoring the system to a stable state is achieved simply by reverting to a previous Git commit. Structured documentation: The modularization of YAML files and the integration of operational guidelines within README files enhance clarity, traceability, and reliability. In addition, GitOps enabled a standardized operational model in which each project maintains an independent configuration and repository-specific governance policy, preventing unintended cross-project impact and strengthening system stability.

*Limitations and Analysis*

Although multiple benefits were identified, several limitations must be acknowledged. Methodological limitations include the small interview sample size and a relatively short observation period of approximately 4–5 months. Technical and process-related limitations involve legacy infrastructure constraints, tooling-related barriers such as the Kubernetes expertise required for Argo CD management, and cultural inertia during the formalization of change management processes. Despite initial resistance, pull request–based workflows were eventually institutionalized as standard practice.

*Key Observations and Startup Case Analysis*

Deployment success rates and MTTR improved significantly, indicating greater operational stability. Developers assumed greater ownership of change management, operations teams shifted toward strategic initiatives, and workflow transparency increased through structured YAML configurations and pull request governance.

A complementary case study of a Georgian technology startup further illustrates these outcomes. Prior to GitOps adoption, deployments were semi-manual and prone to inconsistencies across environments. Following implementation—leveraging Argo CD, declarative YAML configurations, Kustomize overlays, encrypted secret management via Mozilla SOPS, and GitHub Actions automation—operational dynamics improved substantially. Automation reduced delays, enhanced environmental consistency, and increased deployment reliability [9]. Overall, the case demonstrates GitOps's effectiveness not only as a deployment mechanism but also as a catalyst for broader operational and cultural transformation (Table 7).

*Table 7. Comparison of Key Operational Metrics Before and After GitOps Adoption*

| Metric | Before GitOps | After GitOps | Observation |
|---|---|---|---|
| **Deployment Frequency** | 1–2 times per week | 2–3 times per day | Significant increase |
| **Mean Time to Recovery (MTTR)** | Approximately 2 hours | 10–15 minutes | Substantial improvement |
| **Change Failure Rate** | ~30% | ~5% | Reduced error rate |
| **Configuration Consistency** | Low | High | Improved environment alignment |

## 3. Discussion

Despite the positive outcomes, the team encountered several challenges:

YAML configuration complexity: Proper structuring and management of YAML files required additional training and skill development.

Increased merge conflicts: Merge conflicts became more frequent when multiple team members concurrently modified configurations for different environments.

Initial resistance to restricted CLI access: Some team members initially expressed resistance to the reduced use of direct CLI access, perceiving it as a loss of operational flexibility.

## 4. Conclusion

The adoption of GitOps led to a substantial improvement in the company's technical stability and process transparency. Automation and version-controlled infrastructure significantly simplified change management, particularly under conditions of limited resources. Early adoption of GitOps is therefore recommended for startups where speed, reliability, and sustainable growth are critical success factors. At the same time, strong emphasis should be placed on team training and cultural adaptation to ensure that technological changes are implemented effectively.

Evaluation of GitOps: Strengths, Challenges, and Future Perspectives

GitOps represents a logical evolution of modern DevOps practices by integrating version control, infrastructure automation, and systemic transparency within a unified architectural model. Its strengths are multifaceted and manifest across technical, procedural, and organizational dimensions. The use of Git version history as a rollback mechanism enables rapid response and enhances system stability. The pull-based architecture reduces security risks by ensuring that deployment control is restricted to internal GitOps agents, while external systems are not granted direct access to the cluster. Standardization of YAML structures and modularization of repositories mitigate configuration sprawl and improve team collaboration. In addition, comprehensive audit logs establish transparent and accountable workflows, which are particularly valuable in highly regulated environments.

Despite these advantages, the practical adoption and long-term maintenance of GitOps require careful consideration of several challenges. One of the most critical issues is secret management—the secure and correctly versioned handling of sensitive data—which often necessitates additional tools, resources, and procedural discipline. Managing merge conflicts and approval workflows is another delicate aspect, especially in multi-team environments. Git-centric workflows may occasionally delay urgent deployments if fine-grained ownership models or well-defined branching policies are not in place. The learning curve also represents a significant barrier: effective GitOps adoption demands solid expertise in Kubernetes, YAML, Git, and CI/CD systems simultaneously. For organizations that have not yet embraced cloud-native approaches, this can constitute a substantial obstacle. Furthermore, excessive automation—particularly in production environments—may introduce risks. Highly automated deployments without sufficient approval gates or staged validation can result in unintended failures.

Nevertheless, both empirical research and practical case studies indicate that GitOps provides a robust framework for building resilient, secure, and predictable systems. Its effectiveness is not determined solely by tooling; rather, it depends on organizational readiness, cultural transformation, and the strategic integration of security practices. Future perspectives are closely linked to the evolution of GitOps standards—such as the OpenGitOps initiative—the simplification of the tooling ecosystem, and the extension of Git functionality in the areas of security, auditing, and system integration. In this context, GitOps demonstrates high potential when adopted thoughtfully, strategically, and in alignment with the specific needs of an organization.

## REFERENCES

1. Tseradze G., Kuchava G., Mantskava M., and Momtselidze N., “Evaluating GitOps Functionality: Efficiency Analysis and Software Enhancement Opportunities,” (in Georgian), Business and Technology University, pp. 175–176, 2025.
2. Tseradze G., “A Study of GitOps Functionality: Efficiency Analysis and Software Improvement Perspectives,” (in Georgian), Business and Technology University, 50 p., 2025.
3. C. Davis, *GitOps and Kubernetes: Continuous Deployment with Argo CD, Flux, and Friends*. Sebastopol, CA, USA: O’Reilly Media, 2021.
4. G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland, OR, USA: IT Revolution Press, 2016.
5. C. Davis, *GitOps: Continuous Delivery for Kubernetes*. O’Reilly Media, 2021.
6. T. Winters, T. Manshreck, and H. Wright, *Software Engineering at Google: Lessons Learned from Programming Over Time*. O’Reilly Media, 2020.
7. Kuchava G., Kartvelishvili I., and Vashalomidze S., “Using Chaos Engineering to Enhance the Resilience of Microservice Architecture: An Analysis of the ‘Online Boutique’ Case,” (in Georgian), Georgian Technical University, pp. 469–473, 2025.
8. C. Davis, *Cloud Native Patterns: Designing Change-Tolerant Software*. Shelter Island, NY, USA: Manning Publications, 2019.
9. Khinchikashvili A., Kuchava G., and Gaprindashvili A., “Local Area Network Navigation System Software and Hybrid Algorithms for Functionality,” (in Georgian), Business and Technology University, pp. 1–7, 2025.

# AI-Assisted Programming in Practice Conceptual Foundations and Data-Informed Observations

Besiki Tabatadze

*PhD (Applied Mathematics), Professor, European University, Georgian American University, Tbilisi, Georgia.*

***Email:*** *tabatadze.besik@eu.edu.ge*

## Abstract

The increasing use of large language models in programming practice has transformed how software is designed, implemented, and validated. While existing studies largely focus on performance, productivity, or tool ecosystems, less attention has been paid to the role of requirement specification and interpretation in AI-assisted programming. This study examines how differences in task specification influence the behavior of humans and artificial intelligence during program construction.

The analysis is based on a comparative, practice-informed observation of programming tasks executed under two contrasting conditions: an incompletely specified task and a fully, explicitly specified task. Using a controlled programming context and the same execution environment, the study investigates how algorithmic assumptions, interpretative choices, and code structure emerge in each scenario for both human developers and AI-generated solutions. The focus is placed not on execution speed or optimality, but on interpretative variability, stability across executions, and qualitative properties of the resulting program code, including structural complexity and readability.

The observations show that incompletely specified tasks create a wide interpretative space in which AI-generated solutions exhibit substantial variability and, in some cases, exceed the intended scope of the task through functionally abundant or overly complex implementations. In contrast, explicit specification significantly reduces interpretative divergence and leads to conceptual convergence between human-written and AI-generated code. However, even under fully specified conditions, differences remain in qualitative aspects of code structure, with AI-generated solutions tending toward higher structural complexity.

These findings highlight specification as a critical boundary condition in AI-assisted programming and emphasize the evolving role of humans from code production toward interpretation, validation, and refinement in modern programming workflows.

**Keywords:** AI-Assisted Programming, Large Language Models, Task Specification, Interpretive Differences, Program Synthesis, Human-in-the-Loop

## Introduction

In recent years, the use of artificial intelligence in programming practice has expanded significantly. The development of large language models (LLMs) has made it possible to generate program code directly from natural language descriptions, whereby artificial intelligence no longer functions merely as a supporting tool but actively participates in the process of solving programming tasks (Chen et al., 2021; Li et al., 2022; Nijkamp et al., 2022). Against this background, contemporary programming increasingly takes the form of a hybrid process in which human interpretation and machine-generated code coexist within a shared working environment.

In parallel with this trend, artificial intelligence is widely applied both in software systems and across various types of computational tasks, indicating its emergence as a general-purpose computational instrument rather than solely a means of automating specific operations. Research shows that the integration of AI technologies into system architectures and workflows substantially transforms the logic of task formulation, execution, and control, particularly in contexts where processes rely on textual, structural, or semantic interpretation and require human decision support (Chen et al., 2021; Tabatadze, 2024). At the same time, neural-network-based approaches are increasingly considered as alternative computational possibilities for formally defined problems that have traditionally been addressed using strictly algorithmic or numerical methods, including programming tasks and scientific computing (Li et al., 2022; Tabatadze, 2025). A common characteristic of these trends is that the behavior of artificial intelligence depends fundamentally on the quality of task specification and on the constraints that define the boundaries of interpretation in both system-level and computational contexts.

In practical programming processes, tasks are rarely formulated through fully exhaustive and formal specifications. Developers frequently work with partially specified requirements, whose interpretation is grounded in experience, conventions, and contextual judgment. When such tasks are delegated to artificial intelligence, the level of specification becomes a decisive factor in shaping the resulting programmatic solution. Existing empirical studies indicate that incompletely specified tasks increase the variability of AI-generated code and lead to multiple possible implementations of the same task, differing both in internal logic and structural complexity (Gottlander & Khademi, 2023; Ramasamy, 2024; Chen et al., 2021).

In contemporary practice, this tendency is often reflected in prompt-based, exploratory programming approaches, where task formalization is minimal and outcomes depend strongly on interpretation (Ray, 2025; Taulli, 2024). While such approaches enhance flexibility and lower entry barriers, they simultaneously intensify interpretative uncertainty and complicate the assessment of stability and readability of the resulting programmatic solutions.

Research on code generation demonstrates that clear and detailed task specification reduces interpretative variability and promotes conceptual convergence among solutions (Li et al., 2022; Nijkamp et al., 2022). Nevertheless, even under fully specified conditions, AI-generated code frequently differs from human-written code in terms of structural organization, level of abstraction, and readability (Ray, 2025). These observations indicate that specification constitutes not merely a technical prerequisite but a critical boundary factor that governs interpretative freedom and the distribution of responsibility between humans and artificial intelligence within AI-assisted programming processes.

Within this context, the present study aims to examine how different levels of task specification influence the behavior of humans and artificial intelligence in the process of solving programming tasks. Through a practice-informed comparative analysis involving incompletely specified and fully specified formulations of the same task, the study focuses on interpretation, stability across repeated executions, and qualitative properties of program code, rather than on execution speed or algorithmic optimality. In doing so, the paper seeks to more clearly articulate the transformation of the human role in contemporary AI-assisted programming practice, from direct code authorship toward interpretation, validation, and simplification.

## Conceptual Foundations of AI-Assisted Programming

AI-assisted programming represents a conceptual transformation in software development, within which program construction emerges from continuous interaction between human intent and probabilistic inference performed by machines. Unlike traditional programming paradigms, where algorithms are explicitly formulated, precisely specified, and directly implemented by human developers, AI-assisted programming relies on large language models that generate program code based on textual prompts, contextual constraints, and learned probability distributions. As a result, the programming process increasingly shifts from direct code creation toward the management, constraint, and interpretation of machine-generated solutions (Chen et al., 2021; Li et al., 2022).

One of the distinguishing characteristics of large language models is their reliance on probabilistic inference rather than deterministic rule execution. Code generated by AI does not constitute the execution of a formally defined algorithm, but rather a prediction of plausible continuations grounded in statistical regularities of code and natural language. Accordingly, code generation involves an interpretative process in which the model fills underspecified requirements, introduces implicit assumptions, and selects among multiple possible algorithmic realizations. This feature fundamentally differentiates AI-assisted programming from classical automation tools and elevates specification from a secondary technical detail to a central conceptual element.

Within this context, task specification functions as a constraint space that defines the boundaries of interpretative freedom available to artificial intelligence. Incompletely specified tasks leave portions of this space unconstrained, allowing AI systems to introduce additional operations, abstractions, or structural generalizations that are not explicitly required by the task description. In contrast, clearly and explicitly defined specifications restrict the interpretative space and increase the likelihood of convergence toward a particular algorithmic structure (Nijkamp et al., 2022). Importantly, the relationship between specification and outcome is not binary; even minor changes in task formulation may lead to qualitatively different programmatic realizations, even when functional correctness is preserved.

Prompt-based programming practices have become increasingly widespread and follow a clearly positive trend. Vibe coding refers to an intuitive, loosely formalized interaction with AI systems, where priority is given to rapid iteration and outcome generation rather than detailed requirement formalization (Ray, 2025). While this approach lowers entry barriers and increases flexibility, it simultaneously amplifies interpretative variability and shifts responsibility toward the human participant as the final authority for evaluation and control (Taulli, 2024). At a conceptual level, this indicates that AI-assisted programming is not limited to accelerating code production, but also reshapes the internal cognitive and organizational structure of programming practice.

Similar tendencies can be observed in the broader application of artificial intelligence within software systems and computational tasks. The integration of AI technologies into system architectures and decision-support processes demonstrates that neural-network-based models increasingly operate within existing information systems, with their role in everyday practice steadily expanding (Tabatadze, 2024). Likewise, the application of deep learning in scientific computing shows that neural networks may be regarded as alternative solution paradigms even for strictly formalized problems, further underscoring the role of specification across different computational contexts (Tabatadze, 2025).

Thus, AI-assisted programming can be understood as an interpretative process shaped by the interaction between task specification, probabilistic inference, and human oversight. The quality, structure, and stability of AI-generated code are determined not only by model capabilities, but also by how requirements and constraints are formulated. From this perspective, specification functions as a boundary condition that governs the distribution of roles between humans and artificial intelligence in contemporary AI-assisted programming workflows.

## Research Methodology

The present study adopts a qualitative, practice-oriented methodological approach aimed at analyzing how different levels of task specification influence the behavior of humans and artificial intelligence in programming contexts. The study does not seek to evaluate model performance in terms of benchmarks, execution speed, or quantitative accuracy metrics; instead, it focuses on interpretative behavior as it emerges during the formation of programmatic solutions under varying conditions of requirement formalization.

The methodological design is comparative and observational. The same programming task is executed under two distinct conditions: an incompletely specified task and a fully, explicitly specified task. Both human developers and artificial intelligence operate within the same programming language and execution environment, ensuring that observed differences are primarily attributable to task specification and interpretative processes rather than technological factors. In this context, artificial intelligence is not treated as an experimental subject in a strict sense, but as an active participant in the programming workflow.

The study is grounded in programming practice rather than controlled experimentation. Its objective is not to test hypotheses related to optimal performance, but to observe how assumptions, interpretations, and structural decisions emerge during the process of program construction. This practice-informed perspective makes it possible to analyze qualitative characteristics of program code, such as structural complexity, interpretative scope, and stability across repeated executions.

Interpretation is regarded as the central analytical component of the study. Programmatic solutions are evaluated in terms of how requirements are interpreted, which operations are included or excluded, and how consistently similar solutions are produced under identical task formulations. Particular attention is given to differences between human-written code and AI-generated code with respect to levels of abstraction, structural organization, and alignment with task requirements.

## Data-Informed Observations from Programming Tasks with Different Levels of Specification

This section presents data-informed observations drawn from programming practice, focusing on the execution of programming tasks under different levels of specification, both with and without the use of artificial intelligence. The aim of the study is to examine how the roles of humans and artificial intelligence, the nature of interpretation, and the resulting outcomes in modern programming change depending on the degree of task formalization.

Two contrasting cases are examined:

1. A programming task that is incompletely specified and does not include all necessary details;
2. The same task formulated in a fully and explicitly specified manner.

The focus of the analysis is not on execution speed or code optimality. Instead, attention is directed toward the process of requirement interpretation: how it differs between humans and artificial intelligence, what kinds of interpretative assumptions emerge in each case, and how these assumptions affect the final outcome.

The task examined within the study belongs to the class of text data preprocessing (input preprocessing), which is widely used in programming practice and is conceptually simple. Nevertheless, tasks of this type are interpretatively sensitive, as their outcomes depend significantly on the level of requirement specification and the logic of interpretation applied during implementation.

To ensure the comparability of observations, the programmatic solution of the task was implemented using the same programming language and execution environment. The task was implemented in the Python programming language, version 3.11, which is widely established in text data processing practice. The choice of language is motivated by its high readability, its rich ecosystem for text manipulation, and the fact that it is equally common in both human-written and AI-generated program code.

In this context, the objective of the observation is not to evaluate specific syntactic features or language constructs. Python is used as a neutral medium that minimizes the influence of technological differences and allows the analysis to focus on the structural and conceptual properties of the programmatic solution.

*Incompletely Specified Task: Interpretation and Outcomes*

In the first research scenario, a deliberately incompletely specified programming task is employed, with the aim of revealing differences in requirement interpretation between humans and artificial intelligence when task specifications are provided at a minimal level. In both cases, task execution involves the creation of program code that automatically performs text normalization on the given data; manual text processing or editing is not part of the observation.

The programming task is defined as follows: given several paragraphs of textual data, develop a programmatic solution (an algorithm and the corresponding sourse code) that performs text normalization in such a way that the data become uniform and suitable for subsequent automated processing.

The task description does not define specific text normalization rules, applicable operations, or mechanisms for handling exceptions. The requirements do not specify how the processing of textual structural elements, special characters, whitespace, or letter case should be reflected in the program code; nor are formal criteria for evaluating the result provided. Consequently, the task does not prescribe a single unambiguous algorithmic standard and leaves room for multiple, equally plausible programmatic interpretations.

The textual material used for observation consists of paragraphs with differing structural characteristics, containing ordinary sentences as well as punctuation and ellipses, mixed letter case (e.g., combinations of upper- and lower-case letters), redundant whitespace, and, in some cases, special characters and digits. This heterogeneity implies that textual “uniformity” can be achieved through different algorithmic strategies, and the choice of a particular strategy is reflected in the program code produced by the executor.

Such incomplete formalization creates an interpretative space in which the programmatic solution depends on the assumptions and design decisions implemented in the code. For this reason, the unit of comparative analysis in this scenario is not execution speed or syntactic style, but rather which algorithmic assumptions are reflected in the code during the definition of the normalization process and how stably or variably these assumptions manifest in programmatic solutions generated by humans and artificial intelligence.

Accordingly, the evaluation is conducted based on the conceptual profile of the solutions, which involves identifying the following aspects:

1. which operations are implemented in the program code as essential components of normalization (e.g., case normalization, whitespace consolidation, handling of punctuation or special characters);
2. how the processing of textual structural elements is reflected in the code (preservation or merging of paragraphs and lines);

3. how exceptions are handled within the algorithmic logic (e.g., empty lines, non-standard characters, mixed formats);
4. how consistent the program logic generated by the same executor remains under repeated executions.

This analytical framework allows human-created programmatic solutions to be interpreted as experience-based and relatively stable algorithmic interpretations, whereas program code generated by artificial intelligence can be viewed as a spectrum of possible programmatic realizations of the same requirements, which may vary even when the task formulation itself remains unchanged.

The theoretical considerations are further reinforced by empirical observations. When humans construct a programmatic solution, a clear and coherent interpretative framework emerges, grounded in commonly accepted though informal practices of text normalization. Within the program logic, normalization is understood not as a transformation of textual content, but as a form of formal stabilization aimed at ensuring a uniform representation suitable for subsequent automated processing.

More specifically, the programmatic strategy adopted by humans is implemented through operations that standardize the formal characteristics of the text (such as letter case, whitespace, and structural boundaries), while deliberately avoiding algorithmic steps that would alter the content level of the text or the form of individual words. This choice indicates that normalization, as encoded in the program logic, is defined as a minimal yet sufficient transformation.

Observation of task execution by artificial intelligence reveals a different pattern. Under the same task formulation, programmatic solutions generated by artificial intelligence differ from one another in terms of the number of normalization operations implemented in the code, as well as in their content, ordering, and interpretative scope. The observed generations do not converge toward a single algorithmic interpretative framework and, in some cases, extend beyond the boundaries of formal normalization as a purely technical task.

More specifically, in some generations artificial intelligence interprets normalization at the programmatic level as a minimal technical operation, and the corresponding code implements only those steps required for superficial textual stabilization. In other generations, however, the generated code includes additional algorithmic operations that exert a content-level influence on the processed text. Such changes manifest in the program logic through alterations in word order within sentences, reformulation of expressions, the use of synonym substitutions, as well as structural simplification or merging of sentences. Although these transformations often preserve the general meaning of the text, they modify its formulation and semantic emphasis and therefore exceed the scope of normalization understood as formal standardization.

It is particularly important that these content-related effects emerge even in cases where the task description does not require text rewriting or reformulation. This indicates that, during the process of program code generation, artificial intelligence supplements the task requirements based on its internal patterns and language modeling mechanisms. As a result, normalization in some generations algorithmically transforms into a process of text reformulation, increasing the risk of content distortion and reducing the predictability of the resulting programmatic solution.

This interpretative variability is directly reflected in the structure of the generated program code. Observations show that, despite functional correctness, code generated by artificial intelligence often exhibits relatively high structural complexity. Such code frequently introduces multiple logical layers, additional helper constructs, and generalized structures that are not strictly required by the task context but emerge as a consequence of an expanded interpretation of the requirements.

Consequently, reading and comprehending the generated program code requires greater effort, particularly when the task itself is incompletely specified and interpretative assumptions are already substantial. The internal logic of the code is often not directly aligned with the concise task description, increasing the cognitive load during subsequent analysis and validation. From this perspective, comparison is no longer limited to whether the code fulfills the required function, but also encompasses code comprehensibility namely, how easily it can be read, understood, and modified by a third party.

Overall, the observations indicate that AI-generated programmatic solutions frequently exhibit an approach oriented toward so-called "functional abundance," which, in the context of tasks of limited scope, results in excessive structural complexity. This phenomenon is particularly pronounced in incompletely specified tasks, where interpretation is already challenging. Under such conditions, overly complex and difficult-to-read program code reduces the practical effectiveness of AI-assisted programming and increases the role of the human not only in interpretation, but also in understanding, refactoring, and simplifying the generated code.

*Fully (Explicitly) Specified Task: Interpretation and Outcomes*

In the second research scenario, the same programming task is presented in a fully and explicitly specified form, with the aim of minimizing interpretative freedom and clearly defining the required operations, their order, assumptions, and constraints. In this scenario, the task formulation constitutes an unambiguous algorithmic specification within which the programmatic solution must be constructed.

The following programming task is defined (Fully Specified Programming Task):

Several paragraphs of textual data are given (either a list of strings or one large text). Write a program or a function normalize_text(paragraphs) that applies the normalization steps listed below to each paragraph in the exact order specified and returns the normalized paragraphs in the same order.

The input may be:

list[str] — a list of paragraphs; or  str — a single text in which paragraphs are separated by a blank line.

The output must be list[str] — a list of normalized paragraphs. Paragraphs that become empty after normalization must not be included in the output.

The program must implement the following normalization steps in the exact order given:

1. Remove leading and trailing whitespace from each paragraph;
2. Convert any repeated whitespace into a single space;
3. Convert the text to lowercase;
4. Apply Unicode normalization using the NFKC form;
5. Normalize punctuation (standardize dashes and ellipses);
6. Normalize quotation marks;
7. Remove control characters;
8. Ensure exactly one space after punctuation marks;
9. Remove empty paragraphs.

In addition, the following constraints are specified to rule out content-level transformation of the text:

1. Do not change the order of words;
2. Do not change word forms;
3. Do not remove numbers or delete punctuation (except for the normalization defined above);
4. The programmatic solution must be idempotent, meaning that repeated execution must not change the result.

Under conditions where the task is fully and explicitly specified, the development of a programmatic solution by humans primarily constitutes a step-by-step implementation of the stated requirements. The space for interpretative choice is reduced to a minimum, as each operation is defined both in terms of content and exact execution order, while the stated constraints further delimit the task. In this context, the human working process is oriented toward tracing the specification: steps are followed sequentially, such that each code block can be directly mapped to a corresponding item in the task description, and subsequent verification (e.g., compliance with idempotency or the rule of removing empty paragraphs) can be performed through straightforward logical checks.

In human-written program code, a minimalist structure typically emerges. A single main function is implemented, supplemented, if necessary, by a limited number of helper operations that serve the realization of the explicitly listed rules (such as Unicode NFKC normalization, consolidation of whitespace into a single space, or punctuation standardization). This structure enhances code transparency, reduces unnecessary generalization, and simplifies later modification or testing, as each requirement of the specification is represented in the code in a visible and localized manner.

Notably, in this scenario, solutions produced by humans exhibit a high degree of consistency under repeated execution. Detailed requirements establish a stable framework within which the code is less susceptible to variation: any differences that do arise are typically limited to stylistic aspects (such as the organization of function blocks or the use of helper functions), while the functional content and the order of operations remain virtually unchanged.

Observation of task execution by artificial intelligence under fully specified conditions shows that a clear specification significantly reduces interpretative variability. Program code generated by AI generally implements the required steps correctly and, under the given constraints, avoids content-level transformations of the text (such as reordering words or altering word forms). Moreover, explicitly defined test criteria—most notably idempotency—act as a critical boundary condition that enables AI to verify logical consistency and detect obvious conflicts among the rules.

Nevertheless, despite functional correctness, AI-generated program code often continues to exhibit relatively high structural complexity. Observations indicate that AI frequently produces a more builder-style solution, introducing additional helper functions, generalized configurations, or redundant logical branches “for safety,” even when the specification does not explicitly require them. As a result, the code may be functionally correct and idempotent, yet its readability and rapid validation demand greater cognitive effort than that of human-written code.

In the fully specified task scenario, comparative analysis demonstrates that detailed specification ensures conceptual convergence of programmatic solutions: code produced by both humans and artificial intelligence is oriented toward the precise execution of the same rules, and interpretative differences are reduced to a minimum. Under these conditions, artificial intelligence exhibits a clear advantage in the operational dimension of task execution, rapidly generating functionally correct and specification-compliant solutions that satisfy the idempotency requirement.

At the same time, differences remain clearly visible in qualitative aspects of code. Human-created programmatic solutions are typically shorter, more direct, and easier to read, whereas AI-generated code often includes additional structural layers and helper constructs. Consequently, in fully specified tasks, the role of the human shifts away from interpretation toward the evaluation, simplification, and optimization of the generated code.

## Conclusion

This paper has examined AI-assisted programming as an interpretative and collaborative process in which the formation of programmatic solutions depends not only on the capabilities of the applied model, but also on the quality and logic of task specification. The study demonstrates that the integration of large language models into programming practice fundamentally transforms the process of program construction, shifting the focus from direct code writing toward the interpretation of requirements, the formulation of constraints, and the evaluation of generated outcomes.

Practice-informed observations indicate that incompletely specified tasks create a broad interpretative space in which AI-generated programmatic solutions are characterized by high variability, functional abundance, and frequently excessive structural complexity. Under such conditions, artificial intelligence tends to supplement task requirements with implicit assumptions, increasing the risk of semantic deviation and reducing code readability. In contrast, human-developed solutions in the same context typically rely on more stable interpretative frameworks and exhibit clearer, more comprehensible algorithmic logic.

In cases where tasks are fully and explicitly specified, the observations reveal a conceptual convergence between solutions produced by humans and artificial intelligence. Detailed specification significantly reduces interpretative divergence and ensures a high level of functional alignment. Nevertheless, even under these conditions, AI-generated code often retains relatively high structural complexity, while human-written code is more strongly oriented toward evaluation, simplification, and optimization.

Based on these findings, task specification in AI-assisted programming can be understood not merely as a technical requirement, but as a boundary condition that defines the limits of interpretative freedom and shapes the distribution of roles between humans and artificial intelligence in the programming process. In contemporary practice, the human role is increasingly less confined to direct code authorship and more focused on interpretation, validation, and responsibility for the resulting programmatic solutions.

Thus, AI-assisted programming should not be viewed as a direct replacement for traditional programming, but rather as a new conceptual framework in which programmatic outcomes emerge through dynamic interaction between human judgment and probabilistic machine inference. The study underscores that the quality of this interaction ultimately depends on how clearly, consistently, and thoughtfully the task is formulated, and on the degree to which its specification is detailed and complete.

## REFERENCES

Chen, M., Tworek, J., Jun, H., Yuan, Q., Ponde de Oliveira Pinto, H., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., … Zaremba, W. (2021). *Evaluating large language models trained on code. arXiv*. https://doi.org/10.48550/arXiv.2107.03374

Fan, G., Liu, D., Zhang, R., & Pan, L. (2025). The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: a comparative study with traditional pair programming and individual approaches. *International Journal of STEM Education*, *12*(1), 16.

Gottlander, J., & Khademi, T. (2023). The Effects of AI Assisted Programming in Software Engineering.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., ... & Vinyals, O. (2022). Competition-level code generation with alphacode. *Science*, *378*(6624), 1092-1097.

Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., ... & Xiong, C. (2022). Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Ramasamy, D. (2024). *Towards AI-Assisted Data Science Development: Decoding, Visualising, and Enhancing Human-AI Collaboration for Data Science Workflows in Practice* (Doctoral dissertation, University of Zurich).

Ray, P. P. (2025). A Review on Vibe Coding: Fundamentals, State-of-the-art, Challenges and Future Directions. *Authorea Preprints*.

Tabatadze, B. (2024). Prospects of Using AI Technologies in Open Journal Systems (OJS). *Journal of Technical Science and Technologies*, *8*(2), 68-74.

Tabatadze, B. (2025). Solving partial differential equations with deep neural networks. *In Proceedings of the International Conference on Global Practice of Multidisciplinary Scientific Studies II.* 763.

Taulli, T. (2024). *AI-assisted programming: Better planning, coding, testing, and deployment*. O'Reilly Media, Inc..

Wang, Y., Yu, P. D., & Tan, C. W. (2025). Future-Proofing Programmers: Optimal Knowledge Tracing for AI-Assisted Personalized Education. *arXiv preprint arXiv:2509.23996*.

# Use of Artificial Intelligence in IT Project Management

Anastasia Bajiashvili

*PhD (Engineering Sciences), Professor, Georgian American University, Tbilisi, Georgia.*

***Email:*** *nata.bajiashvili@gau.edu.ge*

Nino Lolashvili

*PhD (Engineering Sciences), Professor, Georgian American University, Tbilisi, Georgia.*

***Email:*** *nino.lolashvili@gau.edu.ge*

## Abstract

With the rapid development of technologies, project management is becoming increasingly complex, and new management approaches and organizational cultures are emerging. Companies seek to adopt as many innovations as possible in order to gain a competitive advantage in the market. The constantly changing environment presents continuous challenges for IT project managers, who must find new and innovative ways to manage and execute their activities more effectively. This article examines the benefits of using artificial intelligence (AI) in project management and explores how AI can support IT project managers in their daily activities and help prevent project-related issues.

**Keywords:** IT, AI, Artificial Intelligence, Project Management, Information Systems, ChatGPT

## Introduction

The rapid advancement of digital technologies has fundamentally transformed the way organizations operate, compete, and deliver value [1]. In this dynamic environment, information technology (IT) projects have become increasingly complex, involving distributed teams, tight deadlines, evolving requirements, and heightened expectations for quality and efficiency. Traditional project management approaches, while still valuable, often struggle to cope with the pace of change and the growing volume of data that modern projects generate. As a result, organizations are seeking innovative tools and methods to enhance decision-making, optimize resource utilization, and improve project outcomes [1].

Artificial intelligence (AI) has emerged as one of the most promising technologies capable of reshaping project management practices. By leveraging machine learning, natural language processing, and data-driven analytics, AI-powered systems can automate routine administrative tasks, support risk assessment and forecasting, and provide actionable insights to project managers [2],[6]. This shift allows project leaders to focus more on strategic activities, stakeholder engagement, and team development, while delegating time-consuming and repetitive tasks to intelligent systems.

In the context of IT project management, the potential of AI is particularly significant. IT projects are characterized by rapid technological change, uncertainty in requirements, and a high degree of interdependence between technical and organizational factors. The integration of AI-based tools, such as intelligent assistants and analytics platforms, offers new opportunities to enhance planning accuracy, monitor project performance in real time, and support timely and informed decision-making [2, 6].

This thesis explores the role of artificial intelligence in IT project management, examining how AI technologies can support project managers in their daily activities, improve efficiency, and reduce risks throughout the project lifecycle. By analyzing current applications of AI and illustrating their practical use through real-world examples, the study aims to demonstrate how AI can contribute to more effective, adaptive, and resilient project management practices in contemporary organizations.

## Project and Project Management

A project is a temporary endeavor undertaken to create a unique product, service, or result. An IT project is an effort aimed at developing a new information system or infrastructure and/or upgrading and improving existing ones. Project management is the process that helps organize and manage work effectively. It involves the application of approaches, methods, and practices designed to address specific problems and their solutions in order to achieve successful outcomes [1].

Every IT project has a project manager—a professional who organizes, plans, and implements activities within project constraints (budget, resources, and time). Project managers lead the team, define project objectives, communicate with stakeholders, and are able to anticipate the final project outcomes upon project closure [4].

## Artificial Intelligence

Artificial intelligence (AI) is a broad and rapidly developing field within computer science. Researchers aim to develop AI using various computational systems (Nilsson). In simple terms, AI is a system or machine capable of imitating human behavior in order to perform assigned tasks and use accumulated information for self-improvement [2, 6].

The development of AI is important for several reasons:

- It can reduce human errors;
- It replaces routine tasks;
- It is available at any time;
- It works faster than humans;
- It plays a significant role in social media;
- It supports learning and education [2, 6].

There are different ways to build AI systems depending on intended goals and expected outcomes. AI systems are commonly categorized into three main types: narrow, general, and superntelligence [5, 6].

**Artificial Narrow Intelligence (ANI)** is designed to perform specific tasks within a limited domain. Such systems can execute thousands of calculations per second but are restricted to predefined variables set by designers and developers [2, 6].

**Artificial General Intelligence (AGI)** is considered human-level intelligence. The idea is that AGI would be capable of performing any intellectual task that a human can do, including planning, reasoning, and learning from experience. Unlike ANI, AGI is not limited to a single domain and can generalize across tasks [5].

**Artificial Super Intelligence (ASI)** is a hypothetical form of intelligence in which machines surpass human intelligence in virtually all areas, from scientific creativity to social skills. Although purely theoretical at present, ASI is often discussed in the context of future AI research and ethics [3, 5].

Studies have shown that project managers spend more than half of their time on administrative tasks such as managing checks and updates. Artificial intelligence systems can easily handle these relatively simple but time-consuming tasks. As a result, project managers can devote more time to the most important aspects of their work. In other words, they can focus more on complex processes that go beyond routine management activities. They can also focus more on team members, contribute to staff development, and help identify and develop further skills. Few factors slow down a project as much as a project manager who simply does not have enough time to address the needs and requirements of each team member. Freeing up part of the working time during the day is not only an effective way to ensure smoother project progress, but it also helps create a more supportive work environment in which employees feel supported and know that they will always have access to appropriate resources [2, 6].

Another important potential benefit is **risk assessment**. It is a fact that every project is prone to risks. AI can accurately identify defects or assess product quality. An AI-based system is capable of comparing current progress with the planned schedule and, based on data, can perform the following actions:

- Alert the team about potential delays;
- Indicate that the team is deviating from KPI (key performance indicator) targets;
- Provide recommendations on how the team can maintain alignment with the original project plan.

The use of such models enables teams to quickly identify potential problems and take timely corrective actions.

## AI in Project Management (ChatGPT Case)

A good example of demonstrating the capabilities of artificial intelligence is **ChatGPT**. It is an AI-powered chatbot developed by OpenAI that operates in a conversational mode with users: it receives input, processes it, and returns responses in a human-readable format. It has been trained on a wide range of internet-sourced texts, enabling it to generate easily understandable responses to a broad spectrum of requests.

When considered in the context of project management, ChatGPT also has a wide range of applications. For example, ChatGPT can be integrated into project management tools to provide conversational interfaces for project managers and team members, allowing them to quickly obtain information, updates, and notifications. In addition, ChatGPT can be used to automate repetitive tasks, such as sending status updates, thereby freeing project managers' time to focus on more strategically important activities.

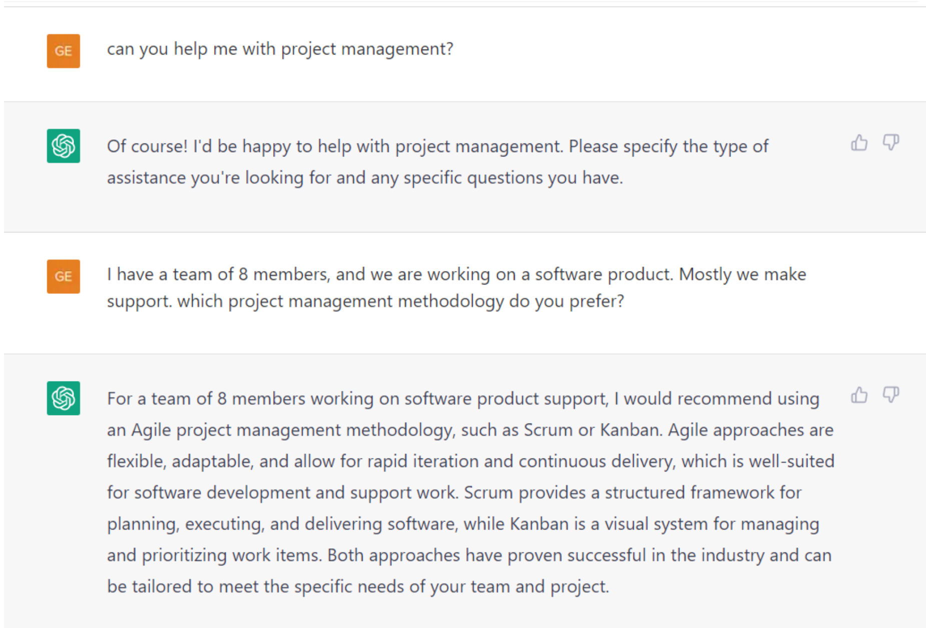
Furthermore, ChatGPT can be used to generate reports such as project progress summaries, resource utilization reports, and risk overviews, helping project managers make more informed decisions. The model's ability to understand and produce natural language makes it an ideal tool for natural language processing and text-based analysis [7].

**Demonstrative short dialogue:**

— *"Can you help me with project management?"*

— *"Of course! I would be happy to help you with project management. Please specify what type of assistance you are looking for and ask any specific questions you may have."*
— *"I have a team of 8 members and we are working on a software product. Most of our work is related to support. Which project management methodology would you recommend?"*
— *"For a team of 8 members working on software product support, I would recommend an Agile methodology, such as Scrum or Kanban. The Agile approach is flexible, adaptive, iterative, and focused on continuous delivery, which is well suited for software development and ongoing support. Scrum provides a structured framework for planning, execution, and delivery of software, while Kanban is a visual system for managing and prioritizing work items. Both approaches are widely used in the industry and can be tailored to the specific needs of your team and project."*

*Figure 1 – Dialogue with ChatGPT on a simple case*




The response provided by ChatGPT fully aligns with the appropriate solution. It correctly selected the project management methodology and provided well-reasoned arguments explaining the working style of each approach [1, 4]. While this question may have been relatively simple for the model, it is worth examining how it performs when asked to provide advice on a specific case supported by concrete figures. The response is presented below:

— *"I have a team of 15 members working on a software product. The deadline is approaching, and only 20% of the work has been completed. What can we do?"*

— *"If you have a team of 15 members working on a software product, the deadline is approaching, and only 20% of the work has been completed, it is important to take immediate action to bring the project back on track. Below are several steps you should consider:*

1. **Assess the situation:** Identify the causes of low progress and evaluate your team's resources and capabilities.

2. **Prioritize tasks:** Determine which tasks are most critical before the deadline and prioritize them accordingly.

3. **Re-plan the schedule:** Based on the prioritized tasks, revise the schedule to ensure that the most important tasks are completed on time.

4. **Assign clear roles and responsibilities:** Ensure that each team member clearly understands their roles and responsibilities.

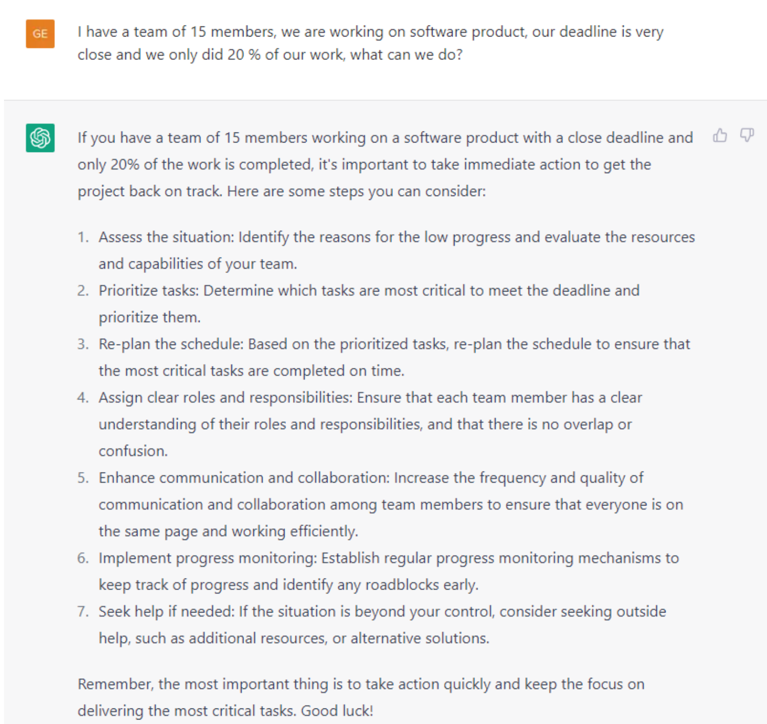
5. **Strengthen communication and collaboration:** Increase the frequency of communication and collaboration among team members to ensure alignment and effective teamwork.

6. **Monitor progress:** Establish regular progress monitoring mechanisms to track progress and identify blockers in a timely manner.

7. **Seek support if necessary:** If the situation is beyond your control, consider seeking external support, such as additional resources or alternative solutions.

*Remember, the most important thing is to act quickly and maintain focus on completing the most critical tasks. Best of luck!"*

*Figure 2 – Dialogue with ChatGPT on a specific case*



GE

I have a team of 15 members, we are working on software product, our deadline is very close and we only did 20 % of our work, what can we do?

If you have a team of 15 members working on a software product with a close deadline and only 20% of the work is completed, it's important to take immediate action to get the project back on track. Here are some steps you can consider:

1. Assess the situation: Identify the reasons for the low progress and evaluate the resources and capabilities of your team.
2. Prioritize tasks: Determine which tasks are most critical to meet the deadline and prioritize them.
3. Re-plan the schedule: Based on the prioritized tasks, re-plan the schedule to ensure that the most critical tasks are completed on time.
4. Assign clear roles and responsibilities: Ensure that each team member has a clear understanding of their roles and responsibilities, and that there is no overlap or confusion.
5. Enhance communication and collaboration: Increase the frequency and quality of communication and collaboration among team members to ensure that everyone is on the same page and working efficiently.
6. Implement progress monitoring: Establish regular progress monitoring mechanisms to keep track of progress and identify any roadblocks early.
7. Seek help if needed: If the situation is beyond your control, consider seeking outside help, such as additional resources, or alternative solutions.

Remember, the most important thing is to take action quickly and keep the focus on delivering the most critical tasks. Good luck!

As evident from ChatGPT's response, the model fully understood the problem at hand and provided recommendations in the correct sequence [7].

## Conclusion

In conclusion, some of the most complex aspects of IT project management involve managing the project budget, schedule, risks, and quality [1, 4]. These areas are closely interrelated and are often affected by uncertainty and changing requirements. The use of artificial intelligence in project management can improve management processes by supporting data-driven decisions, enabling early risk detection, and automating routine tasks [3, 6]. This allows project managers to focus more on strategic coordination and team leadership, increasing efficiency and helping ensure that projects are completed on time and within budget.

## REFERENCES


1. Seymour, T. and S. Hussein, "The History of Project Management," 2014.
2. Oracle, "What Is Artificial Intelligence (AI)?," Oracle. [Online]. Available: https://www.oracle.com/cis/artificial-intelligence/what-is-ai/. [Accessed: May 29, 2023].
3. G. Gurkaynak, I. Yilmaz, and G. Haksever, "Stifling artificial intelligence: Human perils."
4. R. Siles, "The Roles of the Project Manager," 2018. [Online]. Available: https://www.pm4dev.com/pm4dev-blog/entry/the-role-of-the-project-manager.html. [Accessed: May 29, 2023].
5. B. Goertzel, *Aspects of Artificial General Intelligence*.
6. IBM, "What Is Artificial Intelligence?," IBM Cloud. [Online]. Available: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence/.
7. OpenAI, "ChatGPT," 2023. [Online]. Available: https://chat.openai.com/.

# When Digital Attacks Turn Physical: National Security in the Age of Cyber Threats

Akaki Shekeladze

*PhD (Management), Associate Professor, Georgian American University, Tbilisi, Georgia.*

***Email:*** *akaki.shekeladze@gau.edu.ge*

## Abstract

Cyber attacks are usually aimed at compromising personal, business or state sensitive information and IT systems, either for gaining access to data, finance or performing online sabotage. The world has seen various examples of cyber attacks affecting everyday life, from delaying or cancelling multiple flights to the destruction of physical infrastructure. The paper explores the growing interconnection between cyberspace and the physical world, emphasizing how cyber incidents targeting critical infrastructure, such as healthcare facilities, energy systems or transportation networks can directly endanger citizens' safety and disrupt national resilience. The reader will have an overview of the key challenges in the cyber space, which is an alarm for the national security. The text analyses various cyber attacks which managed to affect citizens' physical security and pose serious threats to the overall security of the countries. Emphasis will be made on the nature of the modern cyber space and how malicious actors adapt to changes in the world, demanding both private and state sector to be cyber aware and alert to avoid becoming victims. The research highlights the current vulnerabilities and makes prognosis for the future, how and why cyber attacks can cause even more danger to everyday life. Moreover, information will be provided about the possible solutions, based on the foreign experience, research data and statistics. The study concludes that safeguarding national and citizen security in the 21st century requires not only technological adaptation, but also strategic foresight and coordinated governance at all levels, in all sectors.

**Keywords:** Cyber Security, Cyber Attack, Sabotage, Hackers, National Security, Civil Safety

## Introduction

The recent history of the world includes large-scale cyberattacks carried out in the 21st century, most of which are attacks against states. Attacks are carried out using such tools and technical means as: social engineering, phishing, DoS and DDoS attacks, malicious programs including Ransomware, Spyware, as well as Zero-Day exploits, etc. In the wake of the development of technologies, hackers try to strengthen their capabilities, which includes the development of their human resources, as well as the refinement of tactics, and the strengthening of software tools in order to be able to successfully achieve the malicious goals they set.

This paper discusses the most powerful and well-known cyberattacks carried out against the states. It is about those software tools and technical potential that create the basis for alarm in modern cyberspace and represent the greatest risk for the information security system of the world governments. The results obtained as a result of the research concern not only information systems, but also physical infrastructure. Innovative means of preventing the mentioned threats are proposed and further stages of the research are set to assess their practicality, applicability, feasibility and effectiveness.

## Main Body

1. Destruction of Physical Infrastructure Using Cyber Means

Today, software and non-software means are distinguished from the types of cyberattacks. Even in the last century, there were cases when not only information systems were damaged by software and hardware, but also physical, real infrastructure.

Below are some notable examples of these cases, depending on the means used.

1.1. Trojan.

It is well known that in the 21st century, opposing countries no longer use classical methods of war to achieve their malicious goals. However, in as early as the 80s of the last century, there was a case when cyber activity damaged physical infrastructure.

The first such incident was on the Urengoy-Chelyabinsk gas pipeline in June 1982, where an explosion with a capacity of three kilotons of TNT equivalent occurred using a Trojan. It is noteworthy that no more powerful non-nuclear explosion has been recorded to date. The Trojan Horse software, which was intended to automate technological processes on the gas pipeline, secretly contained a so-called "logic bomb". Its essence was that the program correctly executed several hundred thousand cycles, and then changed the output parameters, which, in turn, increased the pressure in the gas pipeline twice [11].

1.2.Virus.

On August 5, 2008, an explosion occurred on the Turkish section of the Baku-Tbilisi-Ceyhan oil pipeline near Refahiye. According to information security experts, Russian special services were responsible for the explosion. The pipeline network was allegedly hacked through a surveillance camera. The hackers disabled alarm systems and increased pressure, causing an explosion. As a result, 30,000 barrels of oil were spilled, causing BP and its partners to lose $5 million and Azerbaijan to lose $1 billion in revenue [12].

1.3.Virus (Worm) and Zero-Day Exploit.

Most of the public is familiar with the rather large-scale and powerful cyberattack carried out in 2009-2011 by the worm “Stuxnet”. A malicious program called Stuxnet was introduced into the internal network of the uranium enrichment plant in Natanz (Iran). Experts call it a Zero-Day attack, since the worm used a vulnerability in the Siemens Step7 software to infect the programmable logic controller (PLC) [8]. The virus increased the number of revolutions of the centrifuges. As a result, 1,368 out of 5,000 centrifuges failed. Due to economic sanctions, it would have taken years to restore the centrifuges, which forced Iran to begin negotiations on the cessation of nuclear weapons production. As a result, the Iranian government shut down its nuclear weapons program and began negotiations to lift economic sanctions [10].

1.4.Ransomware.

In 2017, the so-called WannaCry attack swept across the globe, causing significant financial losses to victims and disrupting critical services (including healthcare) worldwide [1]. According to Clear Insurance, it was among the 10 largest cyberattacks in history [9]. The WannaCry attack was an example of ransomware. When such software is installed on a victim's device, it blocks access to or encrypts files. The subtype of software that encrypts files is called crypto-ransomware. The attacker offers the victim a decryption key in exchange for a certain ransom, which is usually paid in cryptocurrency. The WannaCry attack demanded a ransom of $300 in Bitcoin (a cryptocurrency). The attackers later increased the ransom to $600. Although some victims paid the "ransom," it is still unknown whether their files were restored. The attack also disrupted critical services, including Spanish mobile phone company Telefonica. Hospitals in the UK were also affected: ambulances were diverted, calls were not answered or were delayed, and in some cases, even fatal. 19,000 doctor visits were canceled, and the healthcare sector suffered a total of £92 million in losses. Ultimately, the ransomware affected 230,000 computers in 150 countries, causing $4 billion in damage worldwide. Research by the reputable Symantec company and others has shown that a North Korean group was behind the Lazarus attack.

1.5.Attack on the US Colonial Pipeline.

On May 9, 2021, the US President declared a state of emergency, the reason for which was a large-scale attack on the largest pipeline in the US, the Colonial Pipeline [3]. The pipeline originates in Texas and supplies fuel to the southeastern part of the US. On May 6, 2021, 100 gigabytes of information were stolen from the company. And on May 7, a powerful cyberattack was carried out. The type of attack was a blackmail/ransomware program. According to the Federal Bureau of Investigation (FBI), the hackers demanded 75 bitcoins (4.4 million US dollars), which the company paid [2]. As a result, they received the appropriate means from DarkSide to restore the system, although this took quite a long time. In addition, the price of a gallon increased by $3.04, which is a historical maximum in the last 7 years. Numerous gas stations were left without fuel in at least 4 states. 87 of the stations in the US capital were empty. People were collecting fuel in various containers from available stations. Flights were delayed from several airports. Panic ensued. [14]

2. Supply chain attacks.

At the end of 2020, the company FireEye released information that an unidentified hacking group had “hacked” the company’s computer system and stolen confidential tools intended for testing the security of information systems [4]. The company, like about 300,000 clients in many countries around the world, uses software from the company SolarWinds to manage computer networks. The hacking group disguised the malware as another update on the SolarWinds update server. Fireeye’s information system automatically updated Solarwinds, as a result of which it was infected [7]. The investigation revealed that more than 18,000 clients worldwide downloaded the malicious Solarwinds update. In the US, the victims of the cyber incident included federal agencies, including the Department of Homeland Security, the Pentagon, the State Department, the Nuclear Nonproliferation Agency, the Treasury Department, the National Institutes of Health, the Department of Energy, and the Department of Finance and Commerce. From the private sector, Microsoft itself became a victim of the cyber incident. The authoritative American publication “The Washington Post”, citing an anonymous source from the investigative bodies, states that the cyber group of the Russian Foreign Intelligence Service APT29, also known as “Cozy Bear”, was behind the operation. The Russian side, traditionally, calls this accusation baseless. [5] The described cyber operation is unique in terms of the selection of the target, planning and flawless execution. The type of cyberattack itself is well known in the field of information security and is called a "Supply Chain Attack."

3. Recent Alarming Cyber Attacks against Critical Services

3.1. Water supply

In February 2021, a hacker infiltrated the control software of the water treatment plant in Oldsmar, Florida and briefly raised the concentration of sodium hydroxide (lye) in the water supply from the normal 100 parts per million to a dangerous 11,100 ppm. The intrusion was made via remote-access software (TeamViewer) on an outdated, poorly secured system — vulnerabilities experts say are common in under-resourced municipal utilities. A plant operator spotted the malicious activity in real time and reverted the change before any contaminated water reached the town, preventing a public-health disaster. Because drinking-water systems are part of a country's critical infrastructure, such cyber-intrusions are not just a criminal matter — they represent a significant threat to public health, economic stability, and national security. Experts have warned the incident should act as a wake-up call: many water and wastewater utilities lack basic cyber-hygiene, proper access controls, or robust operational technology (OT) defenses, making them dangerously exposed.

### 3.2.Travel

In 2025 a major cyber-attack disrupted operations at several European airports when the software of Collins Aerospace — responsible for check-in and boarding systems at dozens of hubs — was hit by ransomware. As a result, automated check-in, baggage drop and boarding systems went offline at key airports such as Heathrow Airport (London), Brussels Airport and Berlin Brandenburg Airport. Airlines and airport staff reverted to manual check-in and boarding procedures, leading to widespread flight delays, cancellations and long passenger queues. This incident has been widely viewed as a wake-up call about the vulnerability of critical infrastructure in the aviation sector — exposing how dependency on third-party digital systems can create systemic risks for national and international travel.

### 3.3.Healthcare

In early August 2023, Prospect Medical Holdings — which runs dozens of hospitals and over 150 clinics across multiple U.S. states — suffered a large-scale ransomware attack by the group Rhysida. The attackers reportedly exfiltrated more than 1 TB of data (including a 1.3 TB

SQL database) containing sensitive personal and health-care information belonging to patients, employees and dependents — full names, Social Security numbers, driver's licenses, medical records, diagnoses, lab results, treatment/insurance data, and more. As a result of the breach, many hospitals under the group had to shut down emergency rooms, redirect ambulances, suspend elective surgeries and outpatient services, and revert to using paper records. The company notified affected individuals, offered credit-monitoring and identity-protection services, and launched investigations in cooperation with law-enforcement — but the incident underscored serious vulnerabilities in cybersecurity in the U.S. health-care sector.

## 4. Discussion

The listed software and non-software tools represent the greatest challenge for the cybersecurity of states, i.e. for national security today. Even the Stuxnet case made several things clear. First of all, after the attack, public awareness of cybersecurity issues increased. Cybersecurity strategies of a number of states were created, changed and updated. According to the report of the Cyber Defense Project "Hotspot Analysis: Stuxnet",

- The state should cooperate with the private sector in the process of protecting critical infrastructure;
- The state should have plans for dealing with cyber attacks such as Stuxnet;
- Countries should have cybersecurity standards for infrastructure assets.

Despite a number of measures, in today's reality, 100% efficiency by states in the fight against cyber threats has not been achieved. It is impossible to completely eliminate technical means, since this reflects the capabilities and signature techniques of hackers; however, depending on the source of the threat, various countries have developed mechanisms that do not completely eliminate, but at least reduce information security risks to some extent.

High awareness is essential. Judging by the cases discussed, criminals often use as a means of attack those people who directly work with information systems. Despite the fact that hundreds of thousands of people are employed in the public sector, agencies should work on raising information security awareness, using training courses, lectures, e-learning platforms, phishing simulations, and other tools. The issue of mandatory training is also important. For some civil servants, information technologies are not of particular interest, which is why they are not interested in security training voluntarily and remain vulnerable to cyber threats. Therefore, it is very important to cover the public sector completely with mandatory retraining programs, especially since threats in cyberspace are increasing every day and the severity of their harmful consequences is also growing. Attracting qualified personnel is another challenge. As in many foreign countries, there is a lack of qualified personnel in the field of information security in our country. One of the reasons for this is the novelty of the field, and another is the difficulty of obtaining academic education. In addition, remuneration in the public sector is less attractive for qualified personnel in this field, since employers in the private sector offer much higher remuneration.

Taking into account cyber threats in the state procurement process is also essential. As discussed above, the so-called Supply Chain Attack represents a complex issue that includes a number of aspects, starting with the legal framework and ending with thorough testing of software and hardware. Most countries' current procurement legislation does not fully take into account cyber threats and cannot completely avoid risks such as the purchase of computer equipment and software from malicious suppliers. To minimize the risks of Supply Chain Attack, it is vital to develop a special procedure for the procurement of cyber technologies as specific goods and services, where the reliability and security of the product become determining factors.

## Conclusion

Thus, the examples discussed in the paper have made clear the importance of cybersecurity as a cornerstone of national security. In most cases, a quantitative method is used to assess the scale and severity of a cyberattack, namely the financial losses suffered by the victim (individual, organization, or state) as a result of a specific action. According to cybersecurity experts, by 2025 the damage caused to the global economy by cybercrime will reach 10.5 trillion US dollars per year. This means that every minute cyberattacks will cause 20 million US dollars in damage to the world economy. As a result of the research, the company "Specops Software" has compiled a list of countries ranked by the number of significant cyberattacks based on data from 2006–2020 [13]. A significant cyberattack is considered an attack on a country's government structures or companies, the losses of which are equal to or exceed one million US dollars. It is noteworthy that the USA is not only the leader in this list, but the number of attacks on it significantly exceeds those recorded in other countries. This is despite the fact that the US Cyber Command receives billions of dollars in funding annually and the country ranks first in the world in the GCI (Global Cybersecurity) Index, with a score of 100 out of 100 [6].

The relevance of the issue of developing cyber capabilities is confirmed by the statement of US President Joe Biden in May 2021. Even the leader of such a country openly acknowledged that the country's critical infrastructure cannot be protected to the appropriate degree and that more work is needed to strengthen cyber capabilities. All this is of great concern to any developed or developing country, as it should be emphasized that although the efforts of countries to combat cyber threats are substantial, they are not sufficiently effective. In the modern world, countries must create opportunities for specialists in the field to combat information security risks through cooperation with the public and private sectors, as well as with the rest of the world, and provide support in order to develop new, innovative, technical and non-technical ways to combat cyber threats.

## REFERENCES

1. L. Johnson, "WannaCry: Ransomware attacks show strong links to Lazarus group," Broadcom, 2017. [Online]. Available: https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=b2b00f1b-e553-47df-920d-f79281a80269&tab=librarydocuments
2. Osborne, "Colonial Pipeline ransomware attack: Everything you need to know," *ZDNet*, 2021. [Online]. Available: https://www.zdnet.com/article/colonial-pipeline-ransomware-attack-everything-you-need-to-know/
3. Duckett, "Incremental improvements are not enough as Biden signs order boosting US cyber posture," *ZDNet*, 2021. [Online]. Available: https://www.zdnet.com/article/biden-signs-order-boosting-us-cyber-posture-saying-incremental-improvements-are-not-enough/
4. Congressional Research Service, "Cybersecurity: Selected cyberattacks, 2012–2021," Congressional Research Service, 2021. [Online]. Available: https://crsreports.congress.gov/product/pdf/R/R46974
5. E. Sanger, N. Perlroth, and E. Schmitt, "Scope of Russian hacking becomes clear: Multiple US agencies were hit," *The New York Times*, Dec. 14, 2020. [Online]. Available: https://www.nytimes.com/2020/12/14/us/politics/russia-hack-nsa-homeland-security-pentagon.html
6. International Telecommunication Union, "Global Cybersecurity Index 2020," ITU, 2020. [Online]. Available: https://www.itu.int/dms_pub/itu-d/opb/str/D-STR-GCI.01-2021-PDF-E.pdf
7. Jibilian and K. Canales, "The US is readying sanctions against Russia over the SolarWinds cyber attack," *Business Insider*, 2020. [Online]. Available: https://www.businessinsider.com/solarwinds-hack-explained-government-agencies-cyber-security-2020-12
8. K. Zetter, "An unprecedented look at Stuxnet, the world's first digital weapon," *WIRED*, Nov. 2014. [Online]. Available: https://www.wired.com/2014/11/countdown-to-zero-day-stuxnet/
9. L. Carter, "10 biggest cyber attacks in history," Clear Insurance, 2022. [Online]. Available: https://clearinsurance.com.au/10-biggest-cyber-attacks-in-history/
10. M. Baezner and P. Robin, "Stuxnet," ResearchGate, 2018. [Online]. Available: https://www.researchgate.net/publication/323199431_Stuxnet
11. RISIDATA, "CIA Trojan causes Siberian gas pipeline explosion," RISIDATA Database. [Online]. Available: https://www.risidata.com/index.php?/Database/Detail/cia-trojan-causes-siberian-gas-pipeline-explosion
12. RISIDATA, "Baku–Tbilisi–Ceyhan pipeline explosion," RISIDATA Database. [Online]. Available: https://www.risidata.com/Database/Detail/baku-tbilisi-ceyhan-pipeline-explosion
13. Specops Software, "The countries experiencing the most 'significant' cyber-attacks," Specops Software Blog, 2023. [Online]. Available: https://specopssoft.com/blog/countries-experiencing-significant-cyber-attacks/

14. "Colonial Pipeline ransomware attack," *Wikipedia*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Colonial_Pipeline_ransomware_attack

# PM Methodology Implementation in IT Management

Anastasia Bajiashvili

*PhD (Engineering Sciences), Professor, Georgian American University, Tbilisi, Georgia.*

***Email:*** *nata.bajiashvili@gau.edu.ge*

Nino Lolashvili

*PhD (Engineering Sciences), Professor, Georgian American University, Tbilisi, Georgia.*

***Email:*** *nino.lolashvili@gau.edu.ge*

## Abstract

The implementation of project management methods in IT management is one of the most important factors in the effective planning and resolution of contemporary managerial tasks. It contributes to the development and adoption of appropriate approaches and methodologies for addressing governance challenges within organizations, supports the formation of a new work culture, and ultimately creates a new value chain that provides competitive advantage in the market. Accordingly, this article discusses the importance of having a well-defined strategy for addressing managerial tasks, including the critical role of a properly structured IT governance framework and the existence of IT departments and a Project Management Office (PMO) based on that framework.

**Keywords:** IT, Project Management, Information Systems, IT Governance, SDLC

## Introduction

Business management is inconceivable without the use of information technologies. This is particularly relevant for both small-scale and large enterprises, such as holdings and corporations. Managing such organizational entities is not a simple task regulated solely by theoretical rules. Despite the extensive use of technologies, it is essential to select an appropriate management policy that complies with the legal environment, meets modern management requirements, and enables the organization to maintain competitiveness in the market[1, 5].

Meeting these requirements highlights the importance of aligning the organization's future vision with IT development strategies. This, in turn, necessitates the establishment of new policies and processes, for which the implementation of effective and well-structured management methods is required [2].

The management of information technologies falls under the full responsibility of the organization's IT departments and constitutes an integral and high-level component of overall organizational management. Within IT departments, all operational activities related to information technologies are carried out in the form of individual programs and/or projects. Consequently, the success of an organization's overall management is directly proportional to the effectiveness of its IT strategy, IT governance, and the successful implementation of IT projects, which may be managed in accordance with internationally recognized methodologies and standards [2, 3, 5].

## IT Governance and Its Importance

IT governance consists of leadership, structures, and processes that enable organizations to make decisions ensuring the development, management, and achievement of their IT strategies and objectives [1, 2, 5]. It is an integral part of organizational governance and is essential for the effective management of both large organizations and small and medium-sized enterprises. This requires a clear understanding of organizational goals and objectives in order to appropriately design and implement IT strategies [2].

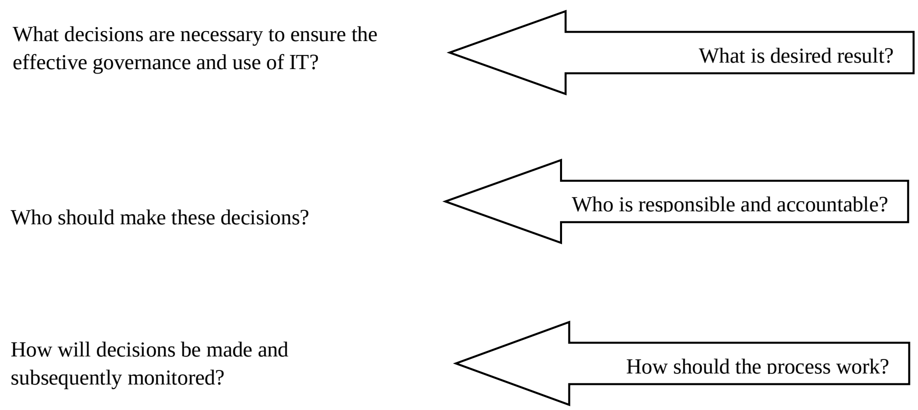
IT governance ensures that IT-related decisions are focused on:

- The appropriate use and effective management of IT departments;

- Monitoring IT department performance to achieve defined objectives and improve processes;
- IT strategies and policies;
- Alignment of IT strategies with organizational goals [2, 5].

To establish proper and effective IT governance following questions should be answered:

*Figure 1 – Three key questions for establishing effective governance*




Within an effective IT governance structure, the following departments are typically included:

- **IT Operations** – responsible for managing IT operations;
- **System Administration** – responsible for the stability of server infrastructure;
- **Database Administration** – responsible for the reliable and secure operation of databases;
- **Network Administration** – responsible for the proper functioning of network devices and network connectivity;
- **Software Development** – responsible for developing and improving software products according to organizational needs;
- **IT Support** – responsible for providing technical support to company employees;

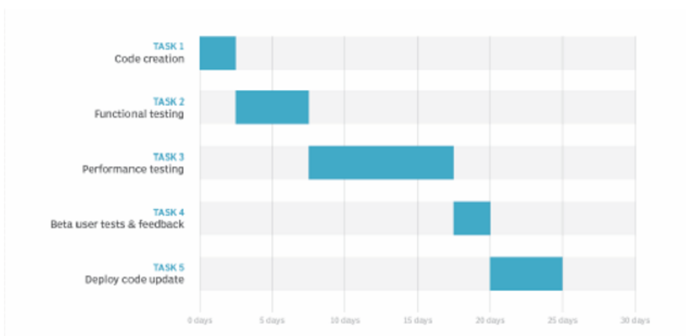
- **PMO (Project Management Office)** – responsible for managing projects and programs within the organization and supporting investment-related initiatives [1, 2, 5].

## PM Methodologies

### *Traditional Project Management Methodology*

Numerous project management methodologies exist in the international arena. For clarity, this paper focuses on two primary approaches: traditional and iterative (Agile) [3]. The traditional methodology consists of several sequential phases, each dependent on the previous one. The process is often planned using a Gantt chart, which visually represents tasks and their start and end dates [3].

*Figure 2- Traditional PM Phases*




The main phases of the traditional methodology include:

Initiation – This phase focuses on formally launching the project. It involves identifying key stakeholders, forming the project team, clarifying roles and responsibilities, and defining the project’s vision, goals, tasks, and overall scope. At this stage, initial feasibility is assessed, high-level risks are identified, and the project charter is often developed and approved to authorize the project [3].

Planning – During the planning phase, the project concept is elaborated in detail. This includes defining deliverables, breaking down work into manageable tasks (e.g., using a Work Breakdown Structure), estimating resources, budget, and timelines, and assigning responsibilities.

Scheduling tools such as Gantt charts and network diagrams (e.g., CPM/PERT) are commonly used. Risk management plans, communication plans, quality criteria, and procurement strategies are also developed to guide successful project execution [3].

Execution – This phase involves carrying out the planned project activities and producing the defined deliverables. The project team performs assigned tasks, coordinates resources, and collaborates with stakeholders. Project managers ensure effective communication, manage team performance, resolve issues, and implement quality assurance processes. Changes and emerging risks are addressed through established change management procedures [3].

Monitoring and Controlling (often integrated with Execution) – Throughout execution, project progress is continuously monitored against the plan. This includes tracking schedule, budget, scope, and quality indicators; managing risks and issues; controlling changes; and reporting performance to stakeholders. Corrective and preventive actions are taken to keep the project aligned with objectives [3].

Closure – The closure phase formalizes project completion. It includes validating and accepting deliverables, evaluating outcomes against objectives, documenting lessons learned and best practices, closing contracts, releasing resources, and preparing final reports. This phase ensures that knowledge gained is captured for future projects and that stakeholders formally acknowledge project completion [3].

In IT projects, project management phases are often aligned with the Software Development Life Cycle (SDLC), which provides a structured framework for developing, deploying, and maintaining information systems.

The SDLC typically consists of the following stages:

- Requirements analysis
- System design
- Implementation

- Testing
- Deployment
- Maintenance/Monitoring [1, 3].

### *Iterative (Agile) Project Management Methodology*

The iterative project management methodology, as expressed in the Agile Manifesto, was created by individuals who had previously worked with traditional project management approaches, including Waterfall. These individuals included project managers, analysts, developers, and other professionals involved in project delivery. According to them: "We are discovering better ways of developing software by doing it and helping others do it." [4].

As a result of this work, they formulated the following system of values:

I. We value **individuals and interactions** over processes and tools

II. We value **working software** over comprehensive documentation

III. We value **customer collaboration** over contract negotiation

IV. We value **responding to change** over following a plan [4].

Therefore, although we value the items on the right, we place greater emphasis on the items on the left. Based on the values outlined above, a framework was developed that is globally known as the Scrum Framework [4].

What is Scrum?

Scrum is a lightweight framework that helps individuals, teams, and organizations create value through adaptive solutions to complex problems [3, 4]. It consists of the following key events (processes):

- Sprint Planning
- Daily Scrum
- Sprint Review

- Sprint Retrospective [4].

When an iterative approach and the Scrum framework are chosen as the project management methodology, requirements are initially formulated as User Stories, which are then collected in the product backlog. It is important to note that the backlog is not fixed; user stories can be added or removed throughout the project lifecycle based on evolving project needs. Once the initial product backlog is prepared, the sprint can begin [3, 4].

A comparison table of traditional and iterative methodologies is presented below.

*Figuge 3 -Methodology Comparison Table*

| ASPECT | TRADITIONAL METHODOLOGY (WATERFALL) | ITERATIVE METHODOLOGY (AGILE / SCRUM) |
|---|---|---|
| APPROACH | Linear and sequential phases | Cyclical, incremental development |
| FLEXIBILITY | Low – changes are costly and difficult | High – changes are welcomed and expected |
| REQUIREMENTS | Defined upfront and fixed | Evolving requirements (User Stories) |
| CUSTOMER INVOLVEMENT | Limited, mostly at the beginning and end | Continuous customer/stakeholder involvement |
| DELIVERY | Single final delivery at project end | Frequent incremental deliveries (sprints) |

| | | |
|---|---|---|
| DOCUMENTATION | Heavy documentation | Lightweight, just-enough documentation |
| RISK MANAGEMENT | Risks identified early but realized late | Risks addressed early through iterations |
| PLANNING STYLE | Detailed long-term planning (Gantt charts) | Adaptive planning per sprint |
| TEAM STRUCTURE | Hierarchical, role-based | Cross-functional, self-organizing teams |
| BEST SUITED FOR | Stable, well-defined projects | Dynamic, innovative, uncertain projects |

There are several aspects that should be considered when selecting an appropriate project management methodology. First and foremost, the project itself must be analyzed in terms of its needs and requirements. Before choosing a suitable methodology, it is essential to clearly understand what requirements and objectives the project must fulfill.

By considering how strict the constraints are in terms of time and budget, it becomes possible to determine whether a traditional or an iterative approach is more appropriate for managing the project.

In addition, the competencies and capabilities of the project team should be assessed, including their level of expertise in the subject area and the extent to which team members are able to work as self-organizing individuals.

The size of the organization and its existing approach to project management should also be taken into account, including whether the organization is large or small and what practices have been applied in previously implemented projects [3, 4].

## Conclusion

In conclusion, organizations should begin by clearly defining their strategic goals, identifying business and technological needs, and formulating a coherent long-term vision before selecting the most appropriate project management methodology. The choice of methodology should not be driven solely by trends or industry popularity, but rather by a careful analysis of the organization's context, project characteristics, regulatory constraints, and maturity level in project and IT governance practices [2, 3, 5].

Both traditional and iterative methodologies offer distinct advantages and limitations, and their successful application depends on proper organizational readiness, leadership support, and cultural alignment. The implementation of any methodology requires a structured change management approach, including the development of appropriate policies, processes, and competencies across the organization. Without adequate preparation, training, and stakeholder engagement, even well-established methodologies may lead to misaligned expectations, resistance to change, and suboptimal project outcomes [3, 5].

Furthermore, the alignment of project management methodology with organizational strategy and IT governance frameworks plays a critical role in ensuring transparency, accountability, and value realization. When strategy, governance structures, and delivery methodologies are coherently integrated, organizations are better positioned to manage risks, optimize resources, and achieve sustainable and measurable project success. This integrated approach ultimately enhances organizational performance and strengthens long-term competitiveness in dynamic and technology-driven environments [2, 5].

## REFERENCES

1. AXELOS, *IT Service Management (ITIL®) V4*. Norwich, U.K.: AXELOS, 2019.
2. ISACA, *COBIT 5: A Business Framework for the Governance and Management of Enterprise IT*. Rolling Meadows, IL, USA: ISACA, 2012.

3. Project Management Institute (PMI), *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 7th ed. Newtown Square, PA, USA: PMI, 2021.
4. K. Beck et al., "Manifesto for Agile Software Development," 2001. [Online]. Available: https://agilemanifesto.org/
5. International Organization for Standardization, *ISO/IEC 38500:2008(E), Corporate Governance of Information Technology*, 1st ed. Geneva, Switzerland: ISO, 2008.

# Application of Shortest Path Algorithms in Telecommunication Channel Coding: A Comparative Study

Tamar Kamkhadze

*MSc (Telecommunications), Invited Lecturer, Georgian American University, Tbilisi, Georgia.*

***Email:*** *tamar.kamkhadze@gau.edu.ge*

## Abstract

Reliable communication over noisy channels requires efficient error-correcting mechanisms. Convolutional channel codes enable error correction by introducing redundancy, while decoding algorithms reconstruct the most probable transmitted sequence. The decoding process can be formulated as a shortest path problem in a weighted directed graph known as a trellis. The Viterbi algorithm performs maximum likelihood decoding by minimizing cumulative path metrics. This paper presents a mathematical formulation of convolutional decoding as a shortest path problem and compares the Viterbi algorithm with classical shortest path algorithms, including the Bellman–Ford algorithm and Dijkstra's algorithm. Structural similarities, complexity differences, and practical implications are analyzed.

**Keywords:** Channel Coding, Shortest Path Algorithm, Viterbi Algorithm, Bellman–Ford, Dijkstra, Trellis Graph

## Introduction

Digital communication systems transmit data across noisy channels, where received signals are corrupted by additive noise and interference [3]. Let the transmitted sequence be:

$$\mathbf{u} = (u_1, u_2, \dots, u_k)$$

After encoding, the transmitted coded sequence becomes:

$$\mathbf{x} = (x_1, x_2, \dots, x_n)$$

The received sequence is:

$$\mathbf{r} = \mathbf{x} + \mathbf{n} \qquad (1)$$

where **n** represents channel noise.

The decoding problem consists of finding the transmitted sequence $\hat{\mathbf{x}}$ that maximizes the conditional probability:

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} P(\mathbf{r} \mid \mathbf{x}) \qquad (2)$$

This optimization problem can be reformulated as a shortest path problem in a weighted graph.

### 1. Convolutional Coding Model

A convolutional encoder with memory $m$ can be modeled as a finite-state machine with:

$$S = 2^m$$

possible states [4].

Each input bit causes a state transition and generates output symbols according to generator polynomials.

The encoder can be represented as a state diagram, which when expanded over time forms a trellis graph.

### 2. Trellis as a Weighted Directed Graph

The trellis is a directed acyclic graph (DAG) defined as:

$$G = (V, E)$$

where:

$V$= set of states at each time step

$E$= transitions between states

Each edge $e \in E$ has an associated weight $w(e)$, defined by a distance metric between received and expected symbols.

For hard-decision decoding, the Hamming distance is used:

$$d_H(\mathbf{r}_t, \mathbf{x}_t) = \sum_{i=1}^{n} | r_i - x_i | \qquad (3)$$

For soft-decision decoding (AWGN channel), Euclidean distance is used:

$$d_E(\mathbf{r}_t, \mathbf{x}_t) = \sum_{i=1}^{n} (r_i - x_i)^2 \qquad (4)$$

The total path metric for path $P$ is:

$$D(P) = \sum_{t=1}^{T} w_t \qquad (5)$$

Decoding becomes:

$$\hat{P} = \arg\min_{P} D(P) \qquad (6)$$

This is equivalent to the shortest path problem.

3. **Trellis as a Weighted Directed Graph**

The Viterbi algorithm applies dynamic programming [1].

Let:

$M_t(s)$be the minimum path metric to reach state $s$ at time $t$.

Recursive update:

$$M_t(s) = \min_{s'} [M_{t-1}(s') + w(s', s)] \qquad (7)$$

Only the minimum (survivor) path is retained.

**Complexity**

If:

- $S$= number of states
- $T$= time steps

Then complexity is:

$$O(S \cdot T) \qquad (8)$$

Because the trellis is layered and acyclic, computation proceeds forward without revisiting states.

## 4. Classical Shortest Path Algorithms

### 4.1 Bellman–Ford Algorithm

The Bellman–Ford algorithm computes shortest paths from a source vertex $s$to all vertices in a graph $G = (V, E)$.

Initialization:

$$d(s) = 0, d(v) = \infty \text{ for } v \neq s$$

Relaxation step:

$$d(v) = \min\,(d(v), d(u) + w(u, v)) \qquad (9)$$

This process is repeated $|\, V \,| - 1$ times.

Time complexity:

$$O(|\, V \,||\, E \,|) \qquad (10)$$

Bellman–Ford can detect negative cycles.

### 4.2 Dijkstra's Algorithm

The Dijkstra's algorithm [2] operates under the constraint:

$$w(u,v) \geq 0$$

It selects the vertex with minimum temporary distance and relaxes adjacent edges. Time complexity (with priority queue):

$$O(|E| \log |V|) \qquad (11)$$

It cannot handle negative weights.

## 5. Comparative Mathematical Analysis

All three algorithms solve:

$$\min_{P \in \mathcal{P}(s,t)} \sum_{e \in P} w(e) \qquad (12)$$

where $\mathcal{P}(s,t)$ denotes all paths from $s$ to $t$.

However, Table 1 shows:

*Table 1-main distinctions between algorithms*

| **Property** | **Viterbi** | **Bellman–Ford** | **Dijkstra** |
|---|---|---|---|
| Graph type | Layered DAG | General graph | General graph |
| Weight sign | Non-negative | Any | Non-negative |
| Update rule | Eq. (7) | Eq. (9) | Greedy selection |
| Complexity | $O(ST)$ | $O(VE)$ | $O(E \log V)$ |

Key distinction:

- Viterbi exploits time-layered structure.
- Bellman–Ford iterates over entire edge set.
- Dijkstra applies global greedy optimization.

## 6. Engineering Implications

The trellis structure enables deterministic hardware implementation of the Viterbi algorithm with fixed memory and predictable computational complexity. This property makes it suitable for real-time communication systems, including mobile and satellite networks.

However, shortest path theory is not limited to channel decoding. In computer networking, routing protocols rely extensively on shortest path algorithms to determine optimal forwarding paths between nodes.

Distance-vector routing protocols such as Routing Information Protocol are based on the principles of the Bellman–Ford algorithm. Each router iteratively updates path costs based on information received from neighboring routers.

Similarly, link-state routing protocols such as Open Shortest Path First compute shortest paths using the Dijkstra's algorithm. Routers construct a global topology map and calculate optimal routes via greedy path selection.

This demonstrates that shortest path optimization forms a foundational principle across multiple layers of communication systems:

- TCP/IP Physical layer → Channel decoding (Viterbi)
- TCP/IP Network layer → Routing protocols (Bellman–Ford, Dijkstra)

Thus, shortest path theory serves as a unifying mathematical framework bridging digital communications and computer networking.

## Conclusion

This paper presented a mathematical formulation of convolutional code decoding as a shortest path optimization problem. The Viterbi algorithm was analyzed as a specialized dynamic programming technique operating on layered trellis graphs. A comparative study with the Bellman–Ford and Dijkstra algorithms revealed both structural similarities and computational distinctions, despite their shared objective of path metric minimization.

Beyond channel decoding, shortest path algorithms play a central role in computer networking, forming the theoretical basis of routing protocols such as distance-vector and link-state approaches. This cross-layer applicability demonstrates that shortest path theory constitutes a unifying mathematical framework spanning the physical and network layers of communication systems.

The results confirm that graph-based optimization is fundamental to modern digital communication and networking architectures, reinforcing the importance of algorithmic theory in practical engineering design.

## REFERENCES


1. A. J. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
3. S. Haykin, *Communication Systems*, 5th ed. New York, NY, USA: Wiley, 2009.
4. J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York, NY, USA: McGraw-Hill, 2008.

# Enhancing CI/CD Security with Artificial Intelligence: State of the Art, Challenges, and Integrated Approaches

Iveri Jajanidze

*PhD Candidate (Informatics), Doctoral Program in Informatics, Georgian Technical University, Tbilisi, Georgia.*

***Email:*** *jajanidze.iveri24@gtu.ge*

## Abstract

Continuous Integration and Continuous Delivery (CI/CD) pipelines have become a core component of modern software engineering, enabling rapid and automated deployment of applications. While these practices significantly improve development efficiency, they also introduce new and complex security risks, including supply chain attacks, configuration drift, and large-scale propagation of vulnerabilities through automated pipelines [6, 7, 8]. Traditional security mechanisms, which rely primarily on static rules and signature-based detection, are increasingly inadequate in highly dynamic DevSecOps environments [3, 12]. This paper investigates the application of Artificial Intelligence (AI) and Machine Learning (ML) techniques to enhance the security of CI/CD systems. The proposed approach integrates anomaly detection, supervised classification, Explainable AI (XAI), federated learning, and adversarial robustness mechanisms into a unified analytical architecture [2, 4, 5]. The study argues that explainability and privacy-preserving learning are critical for adoption in regulated and mission-critical environments [2, 4, 7]. A conceptual and architectural framework is presented, together with an experimental evaluation strategy, demonstrating how AI-driven security analytics can provide proactive, transparent, and scalable protection for modern CI/CD pipelines.

**Keywords:** CI/CD Security, DevSecOps, Artificial Intelligence, Machine Learning, Explainable Artificial Intelligence, Federated Learning, Adversarial Machine Learning

## 1. Introduction

Modern software development is increasingly dependent on Continuous Integration (CI) and Continuous Delivery/Deployment (CD) practices, which enable frequent code integration, automated testing, and rapid release cycles [12, 13]. These approaches form the foundation of DevSecOps, where development, operations, and security are expected to function as a unified and highly automated process [12]. While CI/CD pipelines significantly improve delivery speed and operational efficiency, they also expand the attack surface by introducing complex dependencies, extensive automation, and a high frequency of changes across infrastructure and application layers [6, 7].

A defining characteristic of CI/CD environments is the large-scale use of Infrastructure as Code (IaC), third-party libraries, containerization platforms, and automated orchestration tools [12, 13]. In such settings, a single misconfiguration, compromised dependency, or malicious code injection can propagate through the entire pipeline in a very short time, potentially resulting in large-scale security incidents, including supply chain attacks [3, 9]. Recent incidents and regulatory responses clearly indicate that pipeline security is no longer a peripheral concern but a central requirement for trustworthy software delivery [6, 7, 8].

Conventional security controls in software pipelines are typically based on static analysis, predefined rules, and signature-based detection mechanisms. Although these techniques remain useful, they struggle to cope with the dynamic and heterogeneous nature of modern CI/CD systems and often fail to detect previously unseen attack patterns [3, 12]. As a result, security must be embedded into the pipeline as an adaptive, data-driven, and continuously improving process rather than as a final checkpoint before deployment [12, 13].

In this context, Artificial Intelligence (AI) and Machine Learning (ML) provide promising capabilities for analyzing large volumes of heterogeneous data generated by CI/CD systems, including source code changes, build and deployment logs, configuration files, and runtime telemetry [3, 12]. AI-driven methods enable automated anomaly detection, risk classification, and behavioral analysis at a scale and speed that is not feasible with purely manual or rule-based approaches [3]. At the same time, modern threats increasingly leverage AI themselves, such as adaptive malware and automatically generated social engineering content, which further strengthens the need for AI-assisted defensive mechanisms [3, 11].

However, the adoption of AI in security-critical and regulated environments raises additional requirements. Decision transparency and auditability are essential, particularly in sectors subject to strict compliance frameworks such as ISO/IEC 27001:2022, NIST SP 800-53, and the European Union's Digital Operational Resilience Act (DORA) [6, 7, 8]. Explainable Artificial Intelligence (XAI) addresses this challenge by providing human-interpretable explanations of model outputs, making automated decisions more trustworthy and operationally useful [2, 5].

At the same time, data protection and regulatory constraints often limit the feasibility of centralizing sensitive operational data for model training. Federated learning offers a privacy-preserving alternative by enabling collaborative model training across multiple environments without sharing raw data [4]. Another important challenge is the growing relevance of adversarial attacks against ML models, including data poisoning and evasion techniques, which can be particularly dangerous in highly automated CI/CD pipelines [5, 12].

This paper presents an integrated approach to CI/CD security that combines anomaly detection, supervised classification, XAI, federated learning, and adversarial modeling into a unified architectural framework [2, 3, 4, 5]. The objective is to demonstrate how AI-driven security analytics can move CI/CD protection from a predominantly reactive posture to a proactive, explainable, and privacy-aware security model suitable for modern DevSecOps ecosystems [6, 7, 8].

## 2. Related Work and Problem Formulation

The security of CI/CD and DevSecOps pipelines has attracted increasing attention in recent years due to the growing number of supply chain attacks, configuration-based vulnerabilities, and large-scale automation failures [3, 9, 12]. Existing research and industrial practices emphasize the importance of integrating security controls directly into development and deployment workflows, rather than treating security as a separate, post-deployment activity [12, 13]. Standards and regulatory frameworks such as NIST SP 800-53, ISO/IEC 27001:2022, and the European Union's DORA further reinforce the need for systematic, auditable, and resilient security mechanisms in software delivery infrastructures [6, 7, 8].

A substantial body of work has explored the application of machine learning techniques for security monitoring, including malware detection, intrusion detection, and anomaly detection in operational data [3]. Surveys of ML-based malware analysis show that supervised and unsupervised learning methods can outperform traditional signature-based approaches, particularly in detecting previously unseen threats [3]. In CI/CD contexts, these techniques are increasingly applied to analyze build logs, configuration changes, and runtime telemetry in order to identify abnormal behavior patterns that may indicate compromise or misconfiguration [12, 13].

Explainable Artificial Intelligence has emerged as a critical research direction for safety- and security-critical applications [2, 5]. Methods such as LIME and SHAP have been proposed to provide local and global explanations of model decisions, enabling better human understanding and auditability of automated systems [2, 5, 17, 18]. In security operations, such transparency is essential for incident investigation, compliance verification, and trust in automated decision-making processes [2, 5]. Nevertheless, many existing ML-based security tools still operate as "black boxes," which limits their acceptance in regulated environments [2, 5].

Another important research direction addresses data privacy and governance in distributed infrastructures. Federated learning has been proposed as a solution that enables collaborative model training across multiple environments without sharing raw data [4]. Prior studies demonstrate that federated learning can preserve model performance while significantly reducing the risk of sensitive data exposure, which is particularly relevant for CI/CD ecosystems spanning multiple organizational or regulatory domains [4, 7, 8].

At the same time, the robustness of machine learning models against adversarial manipulation has become a major concern. Adversarial examples, data poisoning, and evasion attacks show that ML models can be intentionally misled with carefully crafted inputs or corrupted training data [5]. In highly automated CI/CD pipelines, such weaknesses are especially dangerous, because erroneous model decisions can propagate rapidly and affect large parts of the delivery process [12]. Although adversarial training and robustness optimization techniques can mitigate some risks, they do not provide a complete solution and require continuous evaluation and adaptation [5, 6].

Despite these advances, several gaps remain. Many existing solutions address only isolated aspects of CI/CD security without integrating explainability, privacy preservation, and robustness into a single coherent framework [2, 3, 4, 5]. Moreover, the operational integration of AI-based security analytics into CI/CD pipelines is often insufficiently addressed, particularly with respect to automated decision points and human-in-the-loop oversight [12, 13]. Finally, regulatory and compliance requirements are rarely treated as first-class design constraints in technical security architectures [6, 7, 8].

Based on these observations, the problem addressed in this paper can be formulated as follows: how to design an AI-driven security architecture for CI/CD pipelines that is effective in detecting complex threats, transparent in its decision-making, respectful of data privacy constraints, and resilient against adversarial manipulation. The proposed approach aims to fill this gap by combining anomaly detection, supervised classification, Explainable AI, federated learning, and adversarial modeling into an integrated DevSecOps-oriented framework, which is detailed in the following sections.

## 3. Methodology and Core Models

The proposed approach is based on the integration of several complementary analytical components that address different aspects of CI/CD security. The methodology combines data-driven anomaly detection, supervised classification, explainable decision support, privacy-preserving collaborative learning, and adversarial robustness mechanisms within a unified framework [2, 3, 4, 5]. This design follows the general principles of secure and auditable systems emphasized in contemporary standards and regulatory frameworks [6, 7, 8].

### *A. Data Sources and Preprocessing*

Security analysis in CI/CD environments relies on heterogeneous data originating from multiple sources, including version control systems, pipeline execution logs, Infrastructure as Code (IaC) configurations, container metadata, and authentication or network events [12, 13]. These data streams are characterized by high volume, partial structure, and strong temporal dynamics, which necessitates a systematic preprocessing stage [12]. This stage includes normalization, deduplication, temporal aggregation, and feature extraction, as commonly recommended in ML-based security analytics [3, 12].

Textual data, such as commit messages and log entries, are processed using natural language processing techniques, including tokenization and vectorization based on term-frequency or embedding representations [12]. The resulting feature vectors form the input to the machine learning models employed in subsequent stages, enabling both statistical learning and behavioral analysis across the CI/CD pipeline [12].

### *B. Supervised and Unsupervised Learning*

Two complementary learning paradigms are used to address different threat detection scenarios. Supervised learning models are applied when labeled examples of known threats are available, following established approaches in malware analysis and security event classification

[3]. Typical classifiers include logistic regression, decision trees, gradient boosting methods such as XGBoost, and neural networks, which have demonstrated strong performance in security-related classification tasks [3, 12].

In contrast, unsupervised learning is primarily employed for the detection of previously unseen or rare events, which is particularly important in dynamic CI/CD environments where new failure modes and attack patterns continuously emerge [3, 12]. Autoencoders and isolation-based methods are widely used for this purpose. Autoencoders are trained to reconstruct normal behavior patterns, and deviations from this learned representation are interpreted as potential anomalies [12]. A high reconstruction error therefore serves as an indicator of abnormal or suspicious activity, which has proven effective in large-scale security monitoring scenarios [12].

### *C. Explainable Artificial Intelligence*

In security-critical contexts, the interpretability of automated decisions is as important as their predictive accuracy [2, 5]. For this reason, the framework integrates Explainable Artificial Intelligence (XAI) techniques that provide human-understandable explanations for model outputs. Model-agnostic methods such as LIME and SHAP are used to estimate the contribution of individual features to a specific prediction and to explain both local and global model behavior [2, 5, 17, 18].

These explanations enable security analysts to verify whether a decision is based on meaningful indicators, to support incident investigation, and to satisfy audit and compliance requirements [2, 5]. By embedding explainability directly into the analytical pipeline, the system addresses a major limitation of many ML-based security tools, which often operate as opaque "black boxes" and therefore face resistance in regulated or mission-critical environments [2, 5].

### *D. Federated Learning and Privacy Preservation*

In many organizational settings, CI/CD data cannot be centralized due to regulatory, contractual, or operational constraints [4, 7, 8]. To address this limitation, the proposed methodology incorporates federated learning as a core training paradigm. In this setup, models are trained locally within separate environments, such as development, testing, and production domains, and only model updates are shared with a central coordinator [4]. The global model is obtained through aggregation of these local updates, which reduces the risk of sensitive data exposure while preserving the benefits of collaborative learning [4].

This approach is particularly suitable for CI/CD ecosystems spanning multiple administrative or regulatory domains, where data governance requirements make traditional centralized training impractical [4, 7, 8].

### *E. Adversarial Robustness Considerations*

The increasing use of machine learning in security systems has introduced new attack vectors, commonly referred to as adversarial attacks [5]. These include adversarial examples, data poisoning, and evasion techniques that aim to mislead models during inference or corrupt them during training [5]. In highly automated CI/CD pipelines, such weaknesses are especially critical, as erroneous model decisions can propagate rapidly through deployment workflows [12].

To mitigate these risks, the proposed framework considers adversarial training and robustness-oriented evaluation as integral parts of the model lifecycle [5]. Although no defense can guarantee complete protection, systematic robustness testing and continuous model adaptation significantly improve resilience against realistic attack scenarios [5, 6].

## 4. System Architecture and CI/CD Integration

The proposed security analytics system is designed as a modular, multi-layer architecture that supports scalable data collection, automated analysis, explainable decision-making, and operational integration within CI/CD environments [12, 16]. The architecture follows the principles of separation of concerns and microservice-based design in order to ensure extensibility, maintainability, and adaptability to different organizational and technological contexts [12, 14, 15].

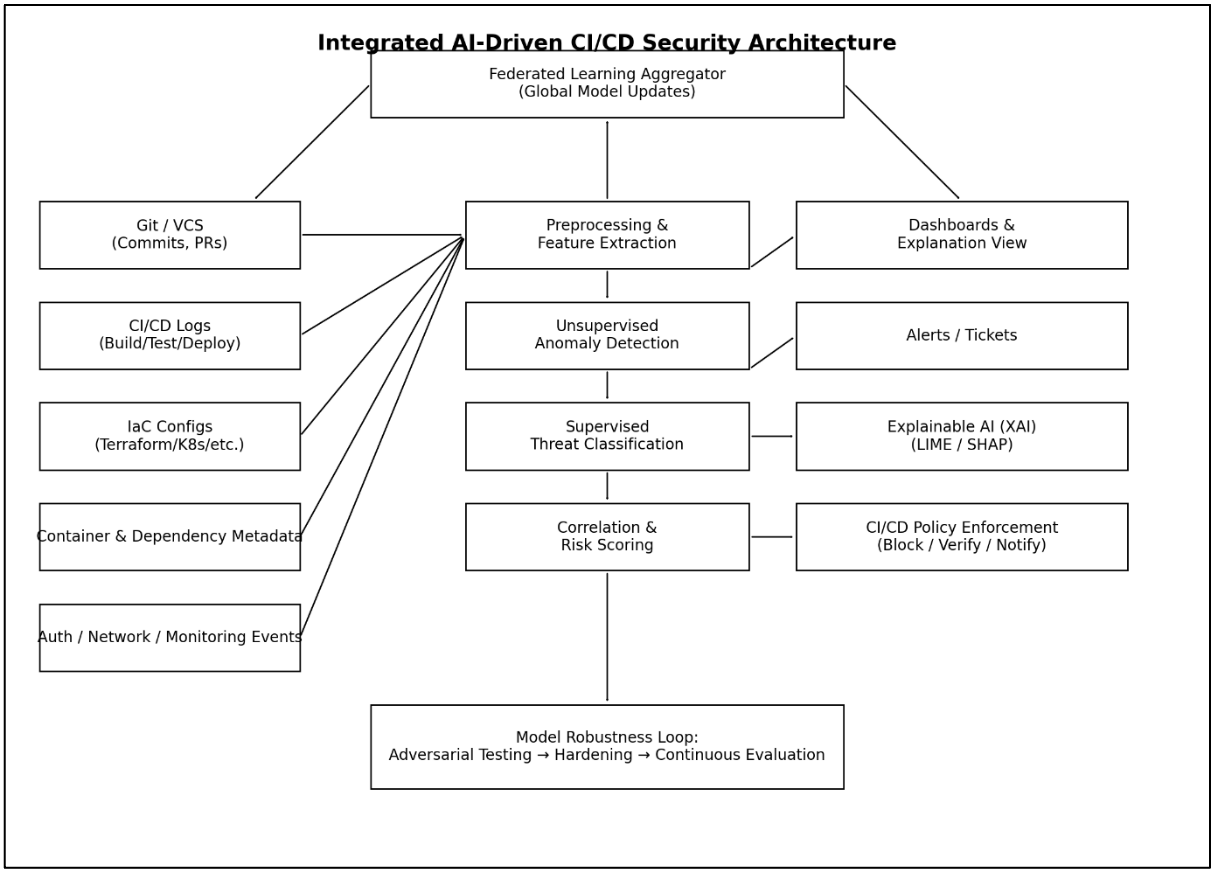
*A. Overall Architecture*

At the lowest layer, the system is responsible for collecting heterogeneous data from multiple CI/CD-related sources, including version control systems, CI/CD orchestrators, container platforms, and monitoring components [12, 13, 15, 16]. These data include source code changes, pipeline execution logs, Infrastructure as Code (IaC) configurations, container metadata, and security-relevant events [12, 13]. The collected information is normalized and forwarded to a central processing module, where preliminary filtering and feature preparation are performed [12].

The intermediate layer hosts the analytical components, including machine learning models for anomaly detection and classification, as well as Explainable AI modules for interpreting model outputs [2, 5, 12]. This layer constitutes the core of the decision-making process and is responsible for correlating events across different data sources in order to identify potential security incidents [3, 12]. In parallel, a federated learning service manages collaborative model updates across multiple environments, enabling continuous improvement of detection capabilities without violating data governance constraints [4, 7, 8].

The upper layer is dedicated to visualization and operational integration. It provides dashboards, metrics, and explanation views that support both real-time monitoring and forensic analysis of incidents [16]. In addition, this layer exposes interfaces for automated or semi-automated actions within the CI/CD pipeline, such as blocking a deployment, triggering additional verification steps, or notifying security personnel [12, 13]. Please refer to figure 1, Integrated AI-Driven CI/CD Security Architecture.

*Figure 1 - Integrated AI-Driven CI/CD Security Architecture*




*B. Data Management and Storage*

The system employs a centralized storage layer that supports both structured and semi-structured data, enabling efficient handling of logs, configuration files, and extracted feature vectors [12]. This storage layer maintains raw event data as well as processed representations used for model training and evaluation, which is essential for reproducibility and auditability of experimental results [6, 7].

Versioning of datasets and models is treated as a first-class requirement, following established MLOps practices for controlled experimentation and traceability [12]. This approach allows historical analyses to be repeated and supports compliance with regulatory expectations regarding accountability and evidence-based decision-making [6, 7, 8].

*C. Integration into CI/CD Pipelines*

A central design goal of the proposed architecture is seamless integration into existing CI/CD pipelines, so that security analysis becomes a natural and continuous part of the development and delivery process rather than an external or ad hoc activity [12, 13]. To achieve this, the system provides integration points at multiple stages of the pipeline, including code commit, build, test, and deployment phases [12, 14, 15].

At each stage, relevant data are collected and analyzed, and the results are fed back into the pipeline control logic. In critical cases, such as the detection of high-risk anomalies or policy violations, the pipeline can be automatically halted or redirected to additional validation steps [6, 7, 12]. In less severe scenarios, the system operates in an advisory mode, providing risk assessments and explanations to developers and operators while leaving the final decision to human oversight [12, 13].

This hybrid approach ensures a balance between automation and operational flexibility, which is essential for maintaining development velocity while enforcing robust security controls [12, 13].

### *D. Monitoring, Visualization, and Operational Use*

Operational effectiveness requires that analytical results be presented in a clear and actionable form. For this purpose, the architecture integrates monitoring and visualization components that aggregate metrics, analyze time series, and present security-relevant information through dashboards and reports [16]. These visualizations include performance indicators of the classification models, distributions of detected anomalies, and XAI-based explanations that highlight the factors influencing specific decisions [2, 5, 16].

Such capabilities support both immediate incident response and long-term strategic analysis of security trends within the CI/CD environment [6, 7]. By combining automated detection with explainable and auditable outputs, the system aims to enhance trust in AI-assisted security decisions and to facilitate their adoption in enterprise and regulated contexts [2, 5, 7].

## 5. Experimental Setup and Results Analysis

This section presents the experimental evaluation of the proposed AI-driven security architecture for CI/CD environments. The primary objective of the experiments is to validate the effectiveness of the selected methods and the proposed architectural integration under realistic operational conditions, including both normal operation and adversarial scenarios [3, 5, 12].

### *A. Experimental Environment*

The experimental setup was deployed in a hybrid environment that combines synthetic workloads with data derived from realistic CI/CD workflows [12, 14, 15]. The test infrastructure includes a version control system, a CI/CD orchestrator, container-based execution environments, and monitoring components, reflecting a typical DevSecOps pipeline architecture [12, 13, 15, 16].

The analytical components were executed in an isolated environment to avoid interference with operational processes and to ensure reproducibility of results, in line with recommended security and auditing practices [6, 7]. Data collection was performed both in near real time and from historical logs, enabling evaluation of the system in both online and offline analysis modes [12].

### *B. Dataset Description*

The dataset used in the experiments consists of several categories of data commonly observed in CI/CD environments. These include version control data such as commit histories and file change statistics, CI/CD pipeline logs covering build, test, and deployment stages, and Infrastructure as Code (IaC) configurations with both valid and intentionally vulnerable parameter settings [12, 13]. In addition, container metadata and dependency information were collected to reflect realistic software supply chain conditions [12, 15].

To evaluate robustness and detection capability, the dataset also includes synthetically generated threat scenarios, such as simulated supply chain attacks, adversarial inputs targeting the ML models, and examples of AI-assisted social engineering patterns [3, 5, 11]. All data were preprocessed using normalization and feature extraction procedures consistent with established ML security analytics practices [3, 12].

*C. Experimental Scenarios*

The evaluation was conducted using a set of representative operational and adversarial scenarios. These scenarios include normal pipeline execution with expected behavior, configuration errors introduced into IaC scripts, simulated supply chain compromises through modified dependencies, and adversarial inputs designed to test model robustness [3, 5, 9, 12]. In addition, social engineering-related scenarios were included to assess the system's ability to detect patterns associated with AI-assisted phishing and identity impersonation attempts targeting DevOps personnel [11, 12].

This scenario-based approach ensures that the evaluation reflects both routine operational conditions and high-risk threat situations commonly discussed in the literature on CI/CD and DevSecOps security [3, 9, 12].

*D. Evaluation Metrics*

The performance of the detection and classification components was assessed using standard metrics commonly applied in security-related machine learning studies, including precision, recall, F1-score, and ROC-based measures [3, 12]. For anomaly detection, reconstruction error distributions and threshold-based detection rates were analyzed to distinguish normal from abnormal behavior [12].

In addition to predictive performance, the quality of explanations generated by the XAI components was evaluated qualitatively, focusing on whether the highlighted features were consistent with domain knowledge and operational expectations [2, 5]. The impact of federated learning on model generalization and data privacy was assessed by comparing centralized and distributed training setups [4, 7, 8].

*E. Results and Discussion*

The experimental results indicate that the combined use of supervised and unsupervised models provides robust detection capabilities across a wide range of scenarios, including previously unseen anomalies [3, 12]. In particular, autoencoder-based anomaly detection proved effective in identifying configuration errors and unusual pipeline behaviors, while supervised classifiers achieved reliable performance in categorizing known threat types [3, 12].

The integration of Explainable AI methods, specifically LIME and SHAP, significantly improved the interpretability of model decisions, enabling analysts to trace detections back to meaningful features in logs, configurations, and metadata [2, 5, 17, 18]. This property is essential for operational adoption and compliance with auditing requirements emphasized in security standards and regulations [6, 7].

Federated learning experiments demonstrated that collaborative training across multiple environments can preserve detection performance while reducing the need to centralize sensitive operational data, thus supporting privacy and governance requirements [4, 7, 8]. Finally, robustness testing under adversarial conditions confirmed that adversarial training and systematic stress testing improve model resilience, although no approach can guarantee complete immunity to sophisticated attacks [5, 6].

Overall, the results support the feasibility of the proposed architecture as a proactive, explainable, and privacy-aware security solution for modern CI/CD pipelines, consistent with the objectives outlined in DevSecOps-oriented security frameworks [12, 13].

## 6. Conclusion

This paper presented an integrated, AI-driven approach to enhancing the security of CI/CD pipelines in modern DevSecOps environments. The proposed framework combines supervised and unsupervised machine learning, Explainable Artificial Intelligence, federated learning, and adversarial robustness mechanisms into a unified architectural model designed to address the dynamic and complex threat landscape of automated software delivery systems.

The analysis demonstrates that traditional, rule-based security controls are increasingly insufficient for highly automated and rapidly evolving CI/CD infrastructures. In contrast, data-driven methods enable scalable anomaly detection and risk classification across heterogeneous data sources, including code repositories, pipeline logs, configuration files, and container metadata. The integration of Explainable AI techniques ensures that automated decisions remain transparent and auditable, which is essential for operational trust and regulatory compliance in security-critical environments.

Furthermore, the adoption of federated learning addresses key data governance and privacy constraints by enabling collaborative model training without centralizing sensitive operational data. This property is particularly relevant for multi-environment and multi-organizational CI/CD ecosystems subject to diverse regulatory requirements. The inclusion of adversarial robustness considerations highlights the necessity of treating machine learning models as potential attack surfaces and of continuously testing and adapting them against evolving threats.

The experimental evaluation indicates that the proposed architecture can effectively detect a broad range of security-relevant events, provide meaningful explanations for its decisions, and maintain acceptable performance under privacy-preserving and adversarial conditions. Although no technical solution can guarantee complete protection, the results support the conclusion that AI-assisted, explainable, and privacy-aware security analytics represent a practical and forward-looking direction for strengthening CI/CD and DevSecOps security.

Future work will focus on large-scale industrial validation, deeper integration with organizational governance processes, and the exploration of advanced model lifecycle management techniques to further improve robustness, transparency, and operational reliability.

## REFERENCES

1. C. Cath *et al.*, "Artificial Intelligence and the 'Good Society': The US, EU, and UK Approach," *Science and Engineering Ethics*, vol. 24, no. 2, pp. 505–528, 2018.
2. D. Gunning *et al.*, "XAI—Explainable Artificial Intelligence," *Science Robotics*, vol. 4, no. 37, 2019.
3. D. Ucci, L. Aniello, and R. Baldoni, "Survey of Machine Learning Techniques for Malware Analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.
4. H. Yang *et al.*, "Federated Machine Learning: Concept and Applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.
5. F. Doshi-Velez and B. Kim, "Towards a Rigorous Science of Interpretable Machine Learning," *arXiv preprint arXiv:1702.08608*, 2017.
6. National Institute of Standards and Technology (NIST), *Security and Privacy Controls for Information Systems and Organizations*, NIST SP 800-53 Rev. 5, 2020.
7. ISO/IEC, *ISO/IEC 27001:2022 Information Security, Cybersecurity and Privacy Protection — Information Security Management Systems — Requirements*, 2022.
8. European Union, *Digital Operational Resilience Act (DORA)*, Regulation (EU) 2022/2554, 2022.
9. I. Kartvelishvili and G. Kuchava, "Optimization of Software Delivery Quality and Speed in DevOps Using CI/CD," in *Proc. Georgian Technical University Conference*, Tbilisi, 2024.
10. I. Kartvelishvili, M. Okhanashvili, and N. Chorkhauli, "Review and Analysis of Existing Methods for Network Attack Detection," in *Proc. International Scientific-Practical Conference*, Tbilisi, 2023.
11. A. Bichnigauri *et al.*, "Strengthening Cyber Defenses — The Crucial Role of Phishing Simulation in Modern Security Strategies," *Defence and Science*, no. 3, 2024, doi: 10.61446/ds.3.2024.8467.
12. Google Cloud, "MLOps: Continuous Delivery and Automation Pipelines in Machine Learning," Google Cloud Documentation, 2022.
13. Microsoft, "Secure DevOps Kit for Azure (AzSK)," Microsoft Documentation, 2023.
14. TensorFlow, "TensorFlow Extended (TFX) Documentation." [Online]. Available: https://www.tensorflow.org/tfx
15. Kubeflow, "Kubeflow Pipelines Documentation." [Online]. Available: https://www.kubeflow.org/docs/components/pipelines
16. Grafana Labs, "Grafana Documentation." [Online]. Available: https://grafana.com/docs/

17. M. T. Ribeiro, S. Singh, and C. Guestrin, "LIME: Local Interpretable Model-Agnostic Explanations." [Online]. Available: https://github.com/marcotcr/lime
18. S. Lundberg and S.-I. Lee, "SHAP: A Unified Approach to Interpreting Model Predictions." [Online]. Available: https://github.com/slundberg/shap
19. European Union, "Artificial Intelligence Act (EU AI Act)." [Online]. Available: https://artificialintelligenceact.eu/