

Agentic-DVC: Operationalizing Decision Efficiency for Autonomous Software Risk Mitigation via Multi-Agent Orchestration

Sadman Sakib^{*[0009-0002-9920-614X]}, Aritri Roy Priota^{**[0009-0003-9576-140X]},
Hossain Mohammad Sami^{***[0009-0004-5659-5821]}, and Abhijit
Bhowmik^{†[0000-0002-9166-347X]}

American International University-Bangladesh (AIUB), Dhaka, Bangladesh
{23-50636-1, 23-51531-1, 23-51354-1}@student.aiub.edu, abhijit@aiub.edu

Abstract. Software development in resource-constrained environments often encounters structural risks associated with unmet requirements and unfamiliar technologies, leading to project failures. These risks are conventionally countered by human-centric review boards, which create verification bottlenecks and significantly reduce Continuous Integration (CI) pipeline performance. This research paper presents an architectural multi-agent framework called Agentic-DVC which is characterized by a formal nature to help emulate and implement the key risk measures of the old Dynamic Verifier Core (DVC) and New Unproven Technology (NUT) through the use of Large Language Models (LLMs). The framework analyzes local telemetry to compute a normalized net decision efficiency score (ED_i), mathematically balancing functional alignment and structural risk against API costs and labor overhead. Through a deterministic routing algorithm called the Deviation Gate, the system leverages ED_i to autonomously merge pull requests, issue refactoring suggestions, or launch an Autonomous NUT agent to synthesize documentation for unfamiliar APIs. The theoretical feasibility of this routing logic is established using a localized fault-injection simulation matrix. A deterministic fail-safe protocol is implemented to halt any autonomous process and notify human reviewers in case of failure to meet the multi-agent consensus threshold.

Keywords: Software Risk Management · Multi-Agent Orchestration · Continuous Integration · Dynamic Verifier Core · Training team for NUT · Risk impact areas · IT Industry · Identify risk

1 Introduction

In resource-limited environments, the software development life cycle continuously faces the challenge of balancing speedy implementation with reliability.

^{*} Corresponding author.

^{**} Equal contribution.

^{***} Equal contribution.

[†] Corresponding author.

Research studies carried out in the field of software engineering have revealed an important fact: Project failures rarely occur due to individual programming mistakes. They happen as a result of improperly managed structural risks at higher levels [4]. For example, a recent comprehensive baseline study focusing on the Bangladesh IT industry identified severe socio-technical bottlenecks: communication failures constitute 68.2% of project failures, architectural flaws account for 43.4%, and the adoption of untested technologies without proper training leads to 63.6% of failures [25]. While localized, these statistics mirror global CI/CD friction points.

Over the years, the IT industry of Bangladesh has attempted to cope with such complex risks by making improvements in Software Development Life Cycle (SDLC) techniques. Initially, studies were conducted on the transformation of the IT industry from the conventional plan-driven framework to the agile approach in order to cope with changes in technicalities [7, 11]. Over time, however, research moved into focusing on finding efficient option choices that could be adopted at various developmental stages of software organizations to come up with hybrid models like Scrum-XP, among others [5, 19]. Despite advancements in the improvement of the SDLC process, such approaches still heavily rely on people-oriented ceremonies.

The conventional risk management framework depends almost completely on human labor in handling the risks. The traditional risk management process entails having physical risk identification and assessment panels that are based on a process consisting of four main processes: Risk Identification, Risk Measurement, Risk Assessment, and Risk Mitigation [25]. Though the process might be possible to execute, there will be many practical problems faced during the implementation of the process in an ongoing development process [26]. Stopping the entire development process just to conduct meeting sessions to assess risks will be a waste of valuable engineering personnel time.

Though the recent progress made in LLMs has included the use of multi-agent systems for code writing and simple peer review, existing systems mainly deal with fixing syntax issues and local bug finding. They are not designed to evaluate the socio-technical structural risks dynamically or determine operational efficiency in terms of costs and, ultimately, major responsibility for risk-based routing of pipelines has to rest on the shoulders of humans. To solve these problems of pipeline friction, this paper proposes the Agentic Dynamic Verifier Core (Agentic-DVC).

This framework creates an independent and cost-aware pre-triage verification system based on multi-agent coordination [12]. It does not intend to do away with human intervention completely; rather, it introduces a continuous evaluation process for tackling deterministic problems automatically but escalating non-deterministic cases for human panelists.

The architecture employs the Orchestrator Agent for interception of inbound payloads and their asynchronous distribution among three parallel verification nodes:

- A Requirements Agent addressing the risk of requirements alignment.

- A Security and Architecture Agent solving the risk of structural failure.
- A Risk Scoring Agent monitoring the consumption of computational tokens and labor by humans.

Instead of subjective estimation by humans, the system uses telemetry from localized agents to synthesize anomalies via an automated Anomaly Synthesis Node. This enables the calculation of the precise normalized net surplus decision efficiency formula, adapted from baseline risk theory [25] by applying a normalization coefficient (ω) to transform raw operational costs (C_k) into a dimensionless utility scale, aligning it with abstract risk metrics:

$$ED_i = \sum_{k=1}^{\text{all actions}} \left[(RRA_Net)_k + OBA_k - (OICA_Net)_k - (\omega \cdot \hat{C}_k) \right] - GICA$$

The calculated efficiency rating (ED_i) leads to the creation of an Automated Deviation Gate, where the code updates will be carried out based on mathematical thresholds. The decision-making logic of the framework entails that pull request payloads having a confirmed net efficiency surplus and no deviations in terms of structure are considered for automated merge validation. Commits with any conflict in the architectural configuration are automatically processed to generate line-by-line refactoring guidelines. Furthermore, in situations when the framework determines the existence of an unknown technology stack, the Autonomous NUT Agent is employed to generate customized documentation packages.

Last but not least, to guarantee operational safety without fail, the system incorporates an overriding fail-safe approach. In the event that high-complexity structural anomalies result in the aggregate detection certainty level dropping below a strict, empirically defined safety threshold, the system immediately overrides autonomous operations and escalates the payload to a traditional human review board.

1.1 Primary Contributions

- **Formal Architectural Blueprint for Risk Operationalization:** We transform static, committee-based formulas for software risk into an explicit multi-agent process which operates on pull requests from repositories.
- **Autonomous Cost-Benefit Switchboard:** An automated verification gate is introduced, relying on absolute net surplus calculations, to balance the cost of computation and refactor in terms of man-hours spent against core feature implementation.
- **Proactive Unproven Stack Mitigation:** To overcome high failure rates associated with unproven stacks, we inject context-sensitive documentation in real-time to developers having issues with unknown dependencies.
- **Deterministic Safety Override:** Our solution incorporates a deterministic safety limit that automatically overrides the process and forces manual intervention when multi-agent detection certainty falls below an empirical threshold.

1.2 Research Questions

To test the effectiveness and feasibility of the proposed approach, this research investigates the following key research questions:

- **RQ1:** How effective is the use of the Agentic-DVC approach in optimizing the mathematical balance among operational costs, structural risks, and features, relative to static analysis approaches?
- **RQ2:** How effectively does the Autonomous NUT Agent’s localized documentation generation reduce the simulated structural risk of integrating unproven technology stacks?
- **RQ3:** What is the quantitative effect of the multi-agent consensus threshold on the latency of CI/CD verification? How does it reconcile pipeline throughput with codebase stability?

2 Related Work

Approaches aimed at reducing software risk as well as approaches that seek to automate code testing were two separate endeavors for quite some time. Recently, these efforts have started overlapping, since LLMs have been introduced into CI pipelines as well.

2.1 Traditional Software Risk Management

Traditionally, risk management required manual and staged assessments of probable threats. The foundation for threat assessment was established by Boehm to predict threats and ensure consistency in operations throughout the duration of the project [4]. Modern approaches use empirical information to point out that changing requirements, instability of architecture, and the use of untested technologies are the major causes of failure. The base assessment conducted by Rahman et al., which focuses on Bangladesh’s IT industry, reveals that the use of untested technology contributes to 63.6% of failures within localized software projects [25]. Traditional solutions to such problems involved using physical task forces. An introduction of the theoretical principles of incorporating an audit function known as the Dynamic Verifier Core (DVC) was made by Khatavakhotan and Ow [14]. This theoretical construct suggests using manual remonitors of risks during software development. While it is an effective idea, its implementation requires significant logistics [26].

2.2 LLMs in Automated Code Verification

In order to solve problems related to human verification, Large Language Models have been employed in automated code review processes. Initial work done by Li et al. illustrated how parametric fine-tuning helps models to conduct code reviews and corrections independently, without any involvement from humans [18]. Further work done by Tufano et al., and later by İçöz & Biricik, took

this ability forward by introducing the concept of RAG (Retrieval-Augmented Generation) to make models capable of self-reliance in identifying structural and security risks [10, 29].

2.3 Multi-Agent Orchestration in Software Engineering

Following this trend in the paradigm, multi-agent collaboration was established, thus offering the required technical base for autonomous task completion within projects. Concepts like MetaGPT and ChatDev have already proven that assigning different personas to particular agents successfully breaks down complex lifecycle processes into automated workflows [8, 23]. More recent approaches, such as RTADev and USEagent, prove the criticality of agent collaboration and cross-validation in terms of deep structural analysis [2, 17].

Although new multi-agent architectures for businesses, like Qodo [24], have emerged with the aim of incorporating automated PR routing and governance using the approach of compliance, and although certain solutions, such as QASec-Claw [1], employ cross-validation in order to minimize false positives, no apparent solution for socio-technical risk management can be seen. The key point here is that all current solutions are focused on the quality of the code or formal verification, but at the same time, they fail to provide any algorithmic balancing of risks according to the structural weaknesses amid the fiat API and associated costs of manual labor. This particular problem is solved by the suggested Agentic-DVC architecture via the implementation of deterministic algorithmic routing protocols in contrast to the historically human-based DVC and NUT committee approaches [25].

2.4 Mitigating Hallucinations in Autonomous Code Generation

However, even while adopting such an architecture, language models will continue to be susceptible to producing syntactically correct yet semantically flawed code [16]. State-of-the-art systems strive to resolve this issue via multiple strategies. The most common strategy is RAG, which confines the generation to a well-known repository setting [28]. Other methods include self-reflections in huge feedback loops where agents have to repeatedly run their generated code and revise it until no errors occur [6].

Another method of avoiding semantic issues is multi-agent debate (e.g., MARS), where different agents evaluate each other's claims to avoid inconsistencies in the results [31]. Nevertheless, utilizing self-reflections in infinite loops exponentially increases both token usage and processing time, directly conflicting with the time limitations of CI/CD environments. The proposed Agentic-DVC framework optimizes this trade-off. Instead of relying on costly internal reflections, it resorts to a tri-agent consensus mechanism (Requirements, Security, and Risk). In the event that consensus cannot be achieved above empirically established thresholds, the system halts automatic operations and escalates the payload data to human reviewers.

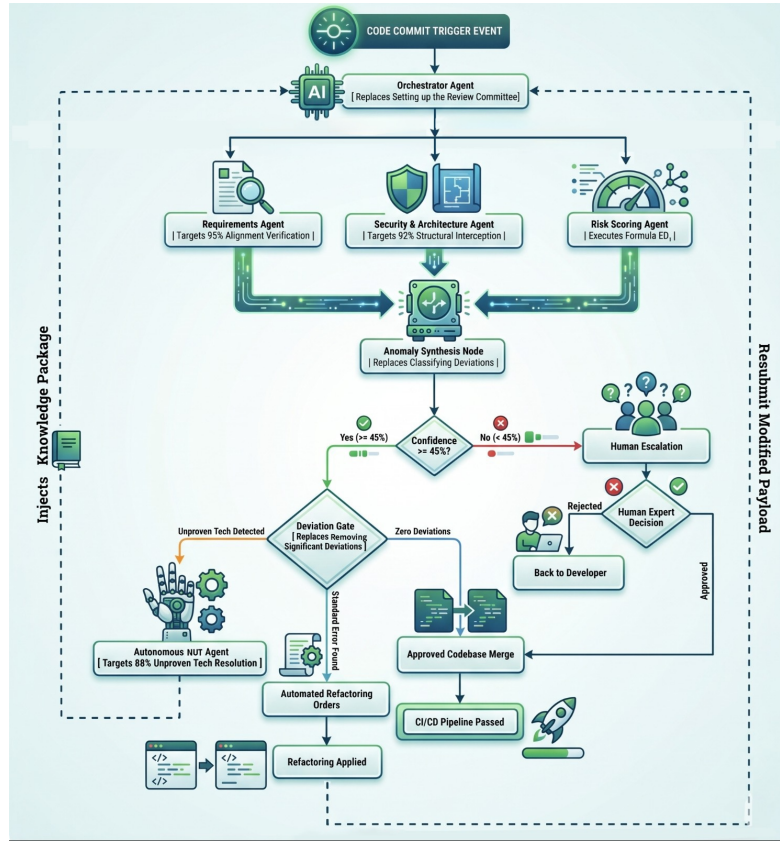


Fig. 1. Agentic-DVC Architecture Workflow and Verification Pipeline

3 Methodology: Agentic-DVC Architecture

The core innovation of the Agentic Dynamic Verifier Core (Agentic-DVC) architecture is the ability to convert static theories of risk mitigation [25] into a practical implementation of an automated, continuous pipeline for execution. Instead of making risk management a sporadic task undertaken by humans, the new approach makes it a continuous process that is deterministic and operates behind the scenes without requiring any human intervention. To achieve this, we leverage the Orchestrator Agent to execute a distributed verification process that works on the basis of multiple agents.

3.1 Multi-Agent Telemetry Acquisition

Upon commit submission, the Orchestrator Agent distributes the repository data payload to specialized verification nodes. These nodes generate the critical telemetry required to solve the decision efficiency equation:

- The Requirements Agent calculates the Opportunity Benefit Amount (OBA_k) by determining how well the payload meets the client’s requirements. This compatibility does not depend on subjective cues but is instead measured through the cosine similarity between high-dimensional vectors generated from the textual representation of the code diff ($\mathbf{c}$) and the project requirements specification ($\mathbf{s}$):

$$\text{SemanticSim}(\mathbf{c}, \mathbf{s}) = \frac{\mathbf{c} \cdot \mathbf{s}}{\|\mathbf{c}\| \|\mathbf{s}\|}$$

- The Security and Architecture Agent detects architectural vulnerabilities to compute the Risk Reduction Amount ($(RRA_Net)_k$) while simultaneously estimating the Other Inconvenient Consequences Amount ($(OICA_Net)_k$) introduced by the payload. Furthermore, the total cost of operation (C_k) is determined by summing up the consumed LLM API tokens and the number of hours spent on refactoring.

3.2 Anomaly Synthesis, Conflict Resolution, and Global Fail-Safe

The Anomaly Synthesis Node aggregates the disparate telemetry streams and executes two critical pre-routing evaluations.

First, it performs analysis at the repository level through measurement of the bug density and code churn of the incoming payload, using weighted hyperparameters ($\alpha, \beta > 0$) to calculate the General Consequence Amount ($GICA$). The weights here are used to normalize historical defect density and commit churn in relation to the 90-day rolling average of the repository.

Second, it combines the confidence levels of individual scanning agents compared to an explicitly determined empirical safety margin ($\tau_{\text{safety}} \in [0, 1]$). In order to address inconsistencies between the diagnoses of different agents (for instance, strong functional alignment but poor architectural integrity), the approach follows a hard safety override principle: if the Security Agent identifies an essential flaw in the structure ($\text{ScanConfidence} < \tau_{\text{safety}}$), the payload is automatically excluded from autonomous integration, regardless of other agents’ consensus. If the collective confidence level of multiple agents is below that threshold because of structural uncertainty or unconfirmed inconsistencies, the system automatically interrupts autonomous operation and elevates the payload to the Human Escalation Node [21].

3.3 Automated Decision Efficiency Routing

If the telemetry validity exceeds the safety threshold, the synthesis node calculates the final decision efficiency (ED_i). To resolve the dimensional mismatch between theoretical risk measures and concrete financial costs, the formula adapts the baseline methodology [25] by introducing a normalization coefficient ($\omega \in (0, 1.0]$):

$$ED_i = \sum_{k=1}^{\text{all actions}} \left[(RRA_Net)_k + OBA_k - (OICA_Net)_k - (\omega \cdot \hat{C}_k) \right] - GICA$$

The parameter ω functions as a risk-aversion coefficient for organizations, which helps environments constrained by budget to severely penalize heavy usage of computation tokens and overhead cost associated with refactoring code ($\hat{C}_k$) heavily against potential feature gains. To achieve dimensional alignment, the operational cost C_k , which includes costs of API token consumption as well as automated developer refactoring, undergoes Min-Max normalization across task budget boundaries $[C_{\min}, C_{\max}]$.

For example, if within a theoretical payload scenario, verifying the result by the LLM will take 2,000 API tokens (for \$0.02), requiring additionally 0.5 hours of work on automated developer refactoring, which will cost \$30.00. Consequently, the cost C_k is \$15.02. If we assume that in the organization's practice, $C_{\min} = \$0$ and $C_{\max} = \$100$, then the resulting $\hat{C}_k$ cost will amount to 0.1502 after Min-Max scaling.

The calculated value for ED_i is then passed on to the Deviation Gate, where one of three predetermined routing processes occurs:

1. In situations where unproven technology stacks are recognized, the autonomous agent generates targeted documentation through automation.
2. When standard structural faults or dependency issues are detected, the system immediately issues automated refactoring directives.
3. If no deviations are detected while retaining a positive value for ED_i , the software payload is automatically approved for entry into the main code.

3.4 Algorithmic Realization of the Framework

The core operational logic of the Agentic-DVC framework is formalized through interconnected, deterministic algorithms executing sequentially within the continuous integration pipeline. The tunable consensus weights $(\gamma, \delta, \epsilon)$ must satisfy the constraints of a unit simplex ($\gamma + \delta + \epsilon = 1.0$, where $\gamma, \delta, \epsilon \geq 0$) and are calibrated to reflect organizational domain priorities (e.g., security-critical systems configure higher δ , whereas rapid prototyping environments prioritize requirement velocity via γ).

4 Theoretical Walkthrough and Proof of Concept

In order to investigate the effectiveness and robustness of the routing algorithm presented by the Agentic-DVC model, fault-injection routing matrix is proposed for this study. Contrary to the process of choosing random historical commits, the matrix is able to create edge-case scenarios and thresholds according to the continuous integration process. Fault injection has been used as the most effective approach to estimate the dependability and reliability of the software systems in adverse conditions [9]. The idea behind fault injection method consists in understanding how Deviation Gate and utility function react in different structural scenarios. Thus, the routing simulation matrix has been developed to understand the reaction of the deterministic routing algorithms to the failure modes and technology vectors which are relevant to the current state of software engineering [25].

Algorithm 1 Agentic-DVC Main Process

Require: *CommitPayload*, *DevHourlyRate*, *TokenPricingTier*, $C_{\min}$, $C_{\max}$
Ensure: Decision (Approve, Escalate, Refactor, NUT)

```

// Step 1: Calculate Cost and Normalize
1:  $C_k \leftarrow (CommitPayload.TotalTokens \times TokenPricingTier) + (RefactoringTime \times DevHourlyRate)$ 
2:  $\hat{C}_k \leftarrow MINMAXSCALE(C_k, C_{\min}, C_{\max})$ 
3:  $\omega \leftarrow OrgRiskToleranceWeight$ 
// Step 2: Compute  $ED_i$  (Synthesized Formula)
4:  $RRA\_Net \leftarrow MitigatedFlaws \times ScanConfidence$ 
5:  $OBA \leftarrow SemanticSim(CodeDiff, ProjectSpec) \times FeatureComp$ 
6:  $OICA\_Net \leftarrow IntroducedConflicts \times ConflictProb$ 
7:  $EDA_i \leftarrow (RRA\_Net + OBA - OICA\_Net) - (\omega \times \hat{C}_k)$ 
8:  $GICA \leftarrow (BugDensity \times \alpha) + (CodeChurn \times \beta)$ 
9:  $ED_i \leftarrow EDA_i - GICA$ 
// Step 3: Agent Consensus via Weighted Hyperparameters
10:  $AgentScore \leftarrow (ReqAgent \times \gamma) + (SecAgent \times \delta) + (RiskAgent \times \epsilon)$ 
// Step 4: Safety Gate
11: if  $AgentScore < EmpiricalSafetyThreshold$  then
12:   return HUMANESCALATE(CommitPayload,  $ED_i$ ,  $AgentScore$ )
13: end if
// Step 5: Deviation Gate
14:  $Threshold \leftarrow OrgRiskTolerance \times HistoricalMedianED$ 
15: if  $ED_i \geq Threshold$  and NoDeviations then
16:   return APPROVEMERGE(CommitPayload)
17: else if UnprovenTechDetected then
18:   return TRIGGERNUTAGENT(CommitPayload)
19: else
20:   return AUTOMATEDREFACTOR(CommitPayload)
21: end if
    
```

4.1 Theoretical Routing Matrix

The localized mathematical model tested the design through four different payload conditions. The primary intention of this table is to prove the mathematical integrity of the ED_i equation, as well as the corresponding routing algorithm used. The outputs of the routing algorithm are shown in Table 1 below.

The findings represent a theoretical analysis of the stability of the proposed model. Here, the “Confidence” column represents the overall confidence level among all nodes that validate, whereas the ED_i index evaluates socio-technical efficiency.

4.2 Scenario Analysis and Execution Traces

Scenario A (Zero Deviations): In Scenario A, the payload created a significant amount of Opportunity Benefit Amount (*OBA*) while consuming fewer amounts of computing power. Deviation Gate gave its permission to execute the

Algorithm 2 Autonomous NUT Agent (Targeted Documentation)

Require: *CommitPayload*, *DevProfile*, $\lambda_{\max_words}$, $\eta_{\max_examples}$
Ensure: Targeted Documentation Package

```

1: UnprovenStack  $\leftarrow$  EXTRACTNEWTECH(CommitPayload)
   // Phase 1: Context Aggregation
2: Docs  $\leftarrow$  CRAWLOFFICIALDOCS(UnprovenStack)  $\cup$ 
   QUERYDEVELOPERFORUMS(UnprovenStack)  $\cup$  QUERYCODEREPOSITORIES(UnprovenStack)
   // Phase 2: Competency-Based Filtering
3: DevLevel  $\leftarrow$  DevProfile.SeniorityLevel
4: FilteredDocs  $\leftarrow$  FILTERBYCOMPETENCY(Docs, DevLevel)
   // Phase 3: Targeted Documentation Synthesis
5: TargetedDocs.Summary  $\leftarrow$  SYNTHESIZE(FilteredDocs,  $\lambda_{\max\_words}$ )
6: TargetedDocs.Examples  $\leftarrow$  EXTRACTVERIFIEDEXAMPLES(FilteredDocs,  $\eta_{\max\_examples}$ )
   // Phase 4: Sandbox Validation (Hallucination Prevention)
7: ValidationStatus  $\leftarrow$  EXECUTESANDBOXVALIDATION(TargetedDocs.Examples)
8: if ValidationStatus = "FAILED" then
9:   return HUMANESCALATE(CommitPayload, "Hallucination Risk")
10: end if
   // Phase 5: Pipeline Injection
11: INJECTDOCUMENTATION(TargetedDocs, CommitPayload.Author)
12: return ORCHESTRATORAGENT(CommitPayload)

```

Table 1. Theoretical Routing Matrix

Profile	Characteristics	Conf.	ED_i	Routing
Commit A	High alignment, zero execution risk.	94%	+4.25	Approved Merge
Commit B	Standard errors, moderate tech debt.	88%	-1.50	Auto Refactor
Commit C	Unfamiliar API, unproven tech risk.	72%	-3.10	Auto NUT Agent
Commit D	Chaotic code, ambiguous anomalies.	28%	Halted	Human Escalate

operation by having a detection confidence value of 94% with an efficient rating value of +4.25, resulting in lower integration times [29, 30].

Scenario B (Standard Error and Automated Patch Trace): Even with a high detection confidence value of 88%, a threat was detected within the execution environment which created an *OICA* penalty, thereby lowering the value of ED_i to -1.50 [20, 32]. The end-to-end execution trace for this profile proceeds through four sequential phases:

1. *Anomaly Interception:* The Security and Architecture Agent detects an unhandled resource leak in the AST diff.
2. *Refactoring Directive Synthesis:* Algorithm 4 constructs a deterministic AST transformation plan ($\Psi_{\text{standards}}$).
3. *Isolated Sandbox Verification:* The synthesized patch is executed in an isolated containerized environment; if compilation or regression tests fail due to hallucinated dependencies, the payload is immediately aborted and escalated.

Algorithm 3 Human Expert Escalation

Require: *CommitPayload*, ED_i , *AgentConfidence*, *AgentLogs*, τ_{timeout}
Ensure: Human Decision (Approved, Rejected, Halted)

```

// Phase 1: Prepare Escalation Package
1: Package  $\leftarrow$  {ID : CommitPayload.ID, Timestamp :
   GETCURRENTTIME(), Score :  $ED_i$ , Confidence : AgentConfidence, Reasoning :
   AgentLogs, Diff : CommitPayload.Diff}
// Phase 2: Notify Expert
2: NOTIFYESCALATIONNODE(Package)
// Phase 3: Await Decision
3: Decision  $\leftarrow$  AWAITREVIEW(CommitPayload.ID,  $\tau_{\text{timeout}}$ )
// Phase 4: Resolution Routing
4: if Decision.Status = "APPROVED" then
5:   LOGDECISION(CommitPayload.ID, "APPROVED",  $ED_i$ )
6:   return APPROVEMERGE(CommitPayload)
7: else if Decision.Status = "REJECTED" then
8:   LOGDECISION(CommitPayload.ID, "REJECTED",  $ED_i$ )
9:   return REJECTANDROUTETODEVELOPER(CommitPayload, Decision.Feedback)
10: else if Decision.Status = "TIMEOUT" then
11:   LOGDECISION(CommitPayload.ID, "SYSTEM_HALTED",  $ED_i$ )
12:   return EMERGENCYHALT(CommitPayload)
13: end if
    
```

4. *Pipeline Re-entry*: Once tests in the sandbox pass successfully, the commit is automatically resubmitted to the Orchestrator with an updated positive ED_i trajectory.

Scenario C (Unknown Technology): The payload utilized an API technique that was quite unknown to the team members. Although the confidence level surpassed the safety threshold of 72%, the dangers associated with the performance aspect were greatly lowered by 3.10. Thus, the system decided to move the payload towards Autonomous NUT Agent processing, implementing knowledge package on an isolated sandbox and thus lowering the danger associated with technology that is unknown [3, 15, 25]. **Scenario D (Fail-Safe Triggering)**: Due to significant structural inconsistencies, the confidence levels dropped down to 28%. Since the figure is lower than the empirical safety threshold, autonomous processing of ED_i was stopped and the payload was passed over to a human review board for structure assessment purposes [21, 22].

4.3 Threats to Validity and Limitations

Although the preliminary walkthrough provides mathematical validation regarding the structural routing process with respect to the Agentic-DVC framework, there are certain limitations associated with this approach in terms of external validity because it depends on simulated code profiles. Theoretical case studies demonstrate the determinism involved in the application of the ED_i criteria.

Algorithm 4 Automated Refactoring Directives**Require:** *CommitPayload*, ED_i , $\Psi_{\text{standards}}$ **Ensure:** Refactored Code Commit

```

// Phase 1: Synthesize Refactoring Plan
1:  $Plan \leftarrow \text{GENERATEPLAN}(CommitPayload.Diff, CommitPayload.Issues, \Psi_{\text{standards}})$ 
// Phase 2: Execute Code Transformation
2:  $RefactoredCode \leftarrow \text{EXECUTEREFACTORING}(CommitPayload.Code, Plan)$ 
// Phase 3: Construct Modified Payload
3:  $NewCommit \leftarrow \text{CONSTRUCTCOMMIT}(\text{Code} : RefactoredCode, \text{Message} : \text{"[Auto-Refactor]"} \oplus CommitPayload.Msg, \text{Parent} : CommitPayload.ID, \text{Author} : \text{"Agentic-DVC System"})$ 
// Phase 4: Pipeline Re-entry
4:  $\text{LOGREFACTOREVENT}(CommitPayload.ID, NewCommit.ID, ED_i)$ 
5: return  $\text{ORCHESTRATORAGENT}(NewCommit)$ 

```

Algorithm 5 Automated Codebase Integration**Require:** *CommitPayload*, $\mathcal{B}_{\text{main}}$ **Ensure:** Integration State

```

// Phase 1: Execute Version Control Merge
1:  $MergeResult \leftarrow \text{EXECUTEMERGE}(\text{Source} : CommitPayload.Branch, \text{Target} : \mathcal{B}_{\text{main}}, \text{CommitID} : CommitPayload.ID)$ 
// Phase 2: Autonomous Event Logging
2:  $\text{LOGAPPROVALEVENT}(CommitPayload.ID, \text{"AUTO\_APPROVED"}, \text{GETCURRENTTIME}())$ 
// Phase 3: Deployment Pipeline Initialization
3:  $\text{INITIALIZECICD}(\text{Branch} : \mathcal{B}_{\text{main}}, \text{Commit} : MergeResult.ID)$ 
4: return  $\text{SYSTEMSTATE}(\text{"INTEGRATION\_COMPLETE"})$ 

```

Yet, they are unable to take into account the unpredictability and chaotic nature of legacy codebases.

In order for the theoretical simulation to move to production deployment, future research requires benchmark testing on multiple real-world repositories across five formal dimensions: (1) Task-Level Success Rate (*TSR*) measuring un-reverted autonomous merges, (2) False Intervention Rate (*FIR*) tracking unnecessary human escalations, (3) Pipeline Verification Latency (*PVL*) across concurrent CI runners, (4) Total Operational Cost (*TOC*) per pull request evaluation, and (5) Decision Variance (σ^2) across repeated stochastic LLM inferences.

5 Conclusion

The agentic-DVC framework provides a formal architecture design that supports risk mitigation for software in situations where resources are limited. By designing a multi-agent automated pre-triage pipeline in addition to review committees, the agentic-DVC framework showcases how critical verification challenges can be solved in contemporary CI/CD pipelines [10]. As evident from the simu-

lated matrix, the above architecture design formulation is successful in modeling the various tradeoffs between cost, risk, and compatibility (RQ1)."

Given the preliminary theoretical calculations, the pipeline triaging process may be safely outsourced to Agent-Computer Interface mechanisms without damaging total software quality [13]. Along with the verification of high-confidence portions of code through automated means, the dynamic documentation provided by the Autonomous NUT Agent offers a specific way to mitigate the structural risks associated with using unproven technology stacks (RQ2).

In order for the model to remain safely functional, there must be strict adherence to operational constraints. The establishment of an empirically grounded confidence boundary allows the system to reconcile pipeline throughput and codebase stability by accounting for calibrated risks (RQ3). Consequently, the proposed system ensures that high-uncertainty payloads are routed directly to manual intervention [27].

Compared to conventional software development methods, whether plan-driven or Agile-based [5, 7, 11, 19], the proposed design framework represents a significant architectural progression for resource-constrained software engineering environments. Although this paper has demonstrated mathematical determinism using fault-injection techniques in a localized setting, immediate future research focuses on implementing and evaluating the pipeline across diverse real-world repository datasets. This empirical roadmap includes component-wise ablation studies for ED_i parameters and Deviation Gate thresholds, direct benchmarking against single-agent baselines, and systematic reporting of task success rates, false intervention rates, latency overhead, token expenditures, and decision variance.

References

1. Ameen, M.R., Alam, M.T.U., Islam, A.: Qasecclaw: A multi-agent llm approach for false positive reduction in static application security testing. arXiv preprint arXiv:2605.01885 (2026)
2. Applis, L., Zhang, Y., Liang, S., Jiang, N., Tan, L., Roychoudhury, A.: Unified software engineering agent as ai software engineer. In: Proceedings of the 48th International Conference on Software Engineering (ICSE) (2026)
3. Barke, S., James, M.B., Polikarpova, N.: Grounded copilot: How programmers interact with code-generating models. Proceedings of the ACM on Programming Languages **7**(OOPSLA1), 85–111 (2023)
4. Boehm, B.W.: Software risk management: Principles and practices. IEEE Software **8**(1), 32–41 (1991)
5. Bose, B., Khan, N.T., Ashreen, S., Ahmed, F., Haque, M.M.U., Bhowmik, A.: Hybrid scrum-xp: A proposed model based on effectiveness of agile model on varieties of software companies in bangladesh. AIUB Journal of Science and Engineering (AJSE) **22**(1), 35–44 (2023)
6. Cho, J., Kang, D., Kim, H., Lee, G.: Self-correcting code generation using small language models. arXiv preprint arXiv:2505.23060 (2025)
7. Chowdhury, A.E., Bhowmik, A., Hasan, H., Rahim, M.S.: Analysis of the veracities of industry used software development life cycle methodologies. Journal of

- Telecommunication, Electronic and Computer Engineering (JTEC) **9**(3-3), 185–189 (2017)
8. Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S.K.S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., Schmidhuber, J.: Metagpt: Meta programming for a multi-agent collaborative framework. In: The Twelfth International Conference on Learning Representations (ICLR) (2024)
 9. Hsueh, M.C., Tsai, T.K., Iyer, R.K.: Fault injection techniques and tools. *Computer* **30**(4), 75–82 (1997)
 10. İçöz, B., Biricik, G.: Context-aware code review automation: A retrieval-augmented approach. *Applied Sciences* **16**(4), 1875 (2026)
 11. Islam, M., Karmaker, D., Imran, M.A., Miah, M.S.U., Bhowmik, A.: Determining the best agile sdlc for bangladesh's software industry. *Asian Transactions on Computers* **5**(2) (2015)
 12. Jennings, N.R.: On agent-based software engineering. *Artificial Intelligence* **117**(2), 277–296 (2000)
 13. Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.: Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793* (2024)
 14. Khataavakhotan, A.S., Ow, S.H.: Development of a software risk management model using unique features of a proposed audit component. *Malaysian Journal of Computer Science* **28**(2), 110–131 (2015)
 15. Lethbridge, T.C., Singer, J., Forward, A.: How software engineers use documentation: The state of the practice. *IEEE Software* **20**(6), 35–39 (2003)
 16. Liu, F., Lin, S., Jian, Y., Wang, L.: Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971* (2024)
 17. Liu, J., Wang, G., Yang, R., Zeng, J., Zhao, M., Cai, Y.: Rtadev: Intention-aligned multi-agent framework for software development. In: Findings of the Association for Computational Linguistics (ACL) (2025)
 18. Lu, J., Yu, L., Li, X., Yang, L., Zuo, C.: Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning. In: IEEE International Symposium on Software Reliability Engineering (ISSRE) (2023)
 19. Mazumder, O., Mim, M.M., Labony, L.N., Bhowmik, A.: Finding efficiency in approach selection in software development and project management in bangladesh it industry. In: International Joint Conference on Computational Intelligence (IJCCI) (2022)
 20. Monperrus, M.: Automatic software repair: A bibliography. *ACM Computing Surveys (CSUR)* **51**(1), 1–24 (2018)
 21. Mosqueira-Rey, E., Hernández-Pereira, E., Alonso-Ríos, D., Bobes-Bascarán, J., Fernández-Leal, Á.: Human-in-the-loop machine learning: A state of the art. *Artificial Intelligence Review* **56**(4), 3005–3054 (2023)
 22. Pearce, H., Ahmad, B., Benjamin, B., Vasilakis, B., Edwards, A., Karri, R., Dolan-Gavitt, B.: Asleep at the keyboard? assessing the security of github copilot's code contributions. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 754–768 (2022)
 23. Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., Sun, M.: Chatdev: Communicative agents for software development. In: Proceedings of the Association for Computational Linguistics (ACL) (2024)
 24. Qodo Technologies: Qodo: Automated pull request routing and governance. <https://www.qodo.ai/> (2026)

25. Rahman, T., Saha, S.K., Sohel, M.S.R., Maula, M.T., Bhowmik, A., Nabil, R.H.: Risk identification and analysis in software development in bangladesh it industry: A hybrid model. *AIUB Journal of Science and Engineering (AJSE)* **21**(1) (2022)
26. Shahin, M., Ali Babar, M., Zhu, L.: Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access* **5**, 3909–3943 (2017)
27. Takerngsaksiri, W., Pasuksmit, J., Thongtanunam, P., Tantithamthavorn, C.: Human-in-the-loop software development agents. *arXiv preprint arXiv:2404.14811* (2024)
28. Tao, Y., Qin, Y., Liu, Y.: Retrieval-augmented code generation: A survey with focus on repository-level approaches. *arXiv preprint arXiv:2510.04905* (2025)
29. Tufano, R., Pascarella, L., Tufano, M., Poshypanyk, D., Bavota, G.: Using large pre-trained models towards automating code review activities. In: *Proceedings of the 44th International Conference on Software Engineering (ICSE)*. pp. 101–112 (2022)
30. Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., Filkov, V.: Quality and productivity outcomes relating to continuous integration in github. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (FSE)*. pp. 805–816 (2015)
31. Wang, X., Zhang, M., Li, J.: Mars: Toward more efficient multi-agent collaboration for llm reasoning. *arXiv preprint arXiv:2509.20502* (2025)
32. Xia, C.S., Zhang, L.: Automated program repair in the era of large pre-trained language models. In: *Proceedings of the 45th International Conference on Software Engineering (ICSE)*. pp. 1482–1494 (2023)