International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 10417-10427
http://www.ijcesen.com

ISSN: 2149-9144

Copyright © IJCESEN

Research Article

Policy-Governed Agentic Autonomous Quality Assurance for Enterprise Systems

Anil Kumar Kunda*

Enterprise Assurance Architect
* **Corresponding Author Email:** AnilK.kunda@outlook.com - **ORCID:** 0009-0003-5916-5387

Article Info:

DOI: 10.22399/ijcesen.4933
Received : 25 Octoer 2025
Revised : 22 December 2025
Accepted : 24 December 2025

Keywords

Agentic AI, Policy-as-Code (PaC),
SEPGA Architecture,
Open Policy Agent (OPA),
Autonomous Software Testing,
Agentic Debt,
Self-Healing Automation

Abstract:

A paradigm shift of deterministic, script-based automation by probabilistic, agentic autonomous systems characterizes the enterprise software landscape of 2025. This research paper offers a comprehensive analysis of Policy-Governed Agentic Autonomous Quality Assurance (QA), a new architectural design that is aimed at alleviating the stochastic risks that any Large Language Model (LLM) agent presents and exploiting their reasoning abilities. Although traditional automation frameworks are said to have reached a scalability limit due to maintenance overheads that consume over 80 percent of QA resources, agentic systems that are not guarded are said to introduce agentic debt a form of technical debt manifested by autonomous decisions that are not aligned. By performing a strict study of the SEPGA (Self-Evolving, Policy-Governed Agentic Automation) architecture, they show that the incorporation of Policy-as-Code (PaC) engines forms a "policy-bounded autonomy." The quantitative evidence shows that by using policy-governed architectures, compliance violations are minimized by 68 percent, and the cost of maintaining the tests is reduced by about 90 percent. It is concluded in the paper that the future of enterprise reliability is in the cryptographic verification of the layers of governance which bind autonomous agents.

1. Introduction

The digital economy of the world has already reached a stage of unprecedented sophistication, in which software delivery speed is becoming more and more incompatible with the stability and security of the systems behind it. With the shift of enterprises to distributed, microservices-based architectures and cloud-native ecosystems, the previous paradigms of software quality assurance (QA) based on manual testing and hard, scripted automation no longer suffice (Vongalis-Macrow, 2007). This insufficiency cannot be described simply as a technical inconvenience but a system crisis with very serious economic consequences.

1.1 The Macroeconomic Burden of Software Failures

The cost of software quality shortcomings has hit disastrous percentages. As per the statistics published by the Consortium for Information and Software Quality (CISQ), the cost of bad software quality (CPSQ) in the United States alone was reported to be in the form of a staggering 2.41 trillion in 2022 but it is expected to reach colossal 3.1 trillion each year in 2024. In order to place this size into perspective the annual loss that can be attributed to software defects is more than the Gross Domestic Product (GDP) of such large G7 countries as France, Italy, or Canada. Effectively, this makes software inefficiency a sort of shadow economy of waste that bleeds corporate resources, kills innovation, and creates systemic risk into important infrastructure.

This cost is multidimensional, and it includes unsuccessful projects, failure of the legacy system, cybersecurity breaches, and downtime (Wang et al., 2024). It is stated that a large part of this loss is attributed to operational failures, as bugs in the software cause disruption of business processes going on. It was estimated that in 2023, an average of one hour of downtime of the enterprise system cost 300,000 dollars. In the case of multinational companies whose transaction platform is high frequency, like a financial service company, these expenses would accumulate into millions in a few minutes. Moreover, the level of these mistakes is not decreasing; about 40% of organizations noted at least one critical software failure each quarter in

2024, implying that disastrous bugs have become standard operational risks and not isolated cases.

1.2 The Accumulation of Technical Debt

One of the main factors behind this quality crisis is the technical debt, which is the cost of rework hidden between half-hearted solution today and the cost of using a superior solution today which would require more time. As of 2022, software technical debt in the U.S., calculated as the principal, had topped 1.52 trillion. This debt is a dragging force in the digital transformation endeavours. It is already projected that by 2025, 40 percent of all IT budgets will be taken up in maintaining this debt alone and thus new development initiatives will be paralyzed.

This dynamic is aggravated by the existence of legacy infrastructure. Although the forceful shift to cloud modernization was aggressive, 70 percent of the banks worldwide were still using legacy by 2025. It is a dependency of the highest order: 95% of all ATM transactions and 43% of core banking systems are still being run on COBOL-based architectures (Liu et al., 2025). The case is no less dire in the healthcare industry with more than 60 percent of all hospitals in the U.S. running critical applications on outdated software, and 96 percent of hospitals noting that they have developed end of life IT systems at some point in their system. Such obsolete systems are not only costly to maintain; they are security risks. Studies have shown that the aged financial systems have thrice the number of security risks as other updated financial systems and this has led to an average cost of 6.08 million and 9.77 million to data breach in the financial and healthcare sector respectively which is the highest form of security breach per industry.

1.3 The Human Capital Crisis in Software Engineering

Poor quality of software affects the economy to the human capital. The "1-10-100" rule is still a standard axiom in the economics of software engineering, which states that any bug that can be avoided at a cost of 1 dollar will cost 100 dollars at least to fix during the development phase and even more money during the production phase. This multiplier has been exacerbated in the complexity distributed systems of 2025; the cost of correcting a bug in production is now 100 times more than the period of correcting it in design.

This waste is reflected in the day-to-day activities of software engineers. Developers are said to devote a percentage ranging between 20 and 33 of their time; about 13.5 hours of a typical 41-hour workweek, to technical debt and bug fixing instead of creating (Liu et al., 2025). This is equivalent to about 20 thousand dollars of salary expense per year of the developer in the U.S. spent on remediation instead of value creation. The emotional impact can also be quite considerable; 58 percent of developers confess that they have thought of quitting because of the exasperation of having to deal with broken legacy technologies, and 69 percent confirm that they waste eight hours or more per week because of operational inefficiencies.

1.4 The Limitations of Deterministic Automation

The use of test automation has historically been the answer to the quality crisis in the industry. Selenium, Cypress and JUnit tools have enabled teams to create verification steps, which are automatically run in CI/CD pipelines. Although this method, commonly referred to as the deterministic automation, has made regression testing faster, it has its own inherent limitations that make it inadequate to the current agentic AI and generative AI systems.

Deterministic automation is based on strict, programmed ways. A test script is a script that tests condition involving condition X and anticipated effect Y (Shah, 2025). In the event that the system is acting in a manner it had not been written into the test script, the automation fails to notice the anomaly or, on the other hand, fails the test because of an innocent change in the UI (flakiness). When working with Generative AI and autonomous agents, system behaviors cannot be determined; the output of a LLM or an agentic workflow can be syntactically different but semantically sound, or it can result in an output that is syntactically sound but does not satisfy an unspoken safety condition. These nuances are impossible to be reasoned in traditional scripts. Their cognitive ability is not flexible enough to search the solution space of a system as a human tester would.

As a result, the paradigm shift at the industry is the shift of Automated Quality Assurance to Agentic Autonomous Quality Assurance. The new paradigm uses AI agents that do not merely act out of a script but have the freedom to design tests plans, reason about system feedback, use complex user interfaces, and self-correct when they meet unforeseen changes (Lv et al., 2024). But when it comes to giving testing agents their freedom, there is another set of threats: what should an enterprise do to ensure that an autonomous agent acts in safe, legal, and ethical area? This requires the incorporation of Policy Governance which is a control plane that imposes limitations as immutable laws of the environment of the agent.

2. Theoretical Framework: From Automation to Agentic Autonomy

To appreciate the architecture of Policy-Governed Agentic QA it is necessary to first make a distinction between automation and agency. Automation refers to the use of a machine to perform a pre-defined sequence of operations to accomplish a desired result (Qian et al., 2024). In the context of Artificial Intelligence, agency would mean having a system with a high-level objective (e.g., "Ensure the checkout process is functional and secure"), and the ability to independently break down the high-level objective into sub-tasks, plan an action plan, execute the plan, and respond to environmental feedback.

2.1 The Cognitive Architecture of QA Agents

Cognitive architectures form the basis of agentic systems which mimic how humans' reason. These architectures are usually based on Large Language Models (LLMs) as the general reasoning engine with a number of essential components:

1. Perception and Tools: The agent will be required to sense the software environment. This can be done in web testing by both parsing the DOM (Document Object Model) and by computer vision in order to look at the UI (Parasa, n.d.). Tools enable the agent to communicate with the system, maybe clicking buttons, making API calls or even searching databases.
2. Memory: Agents have memory, and unlike stateless scripts. The context of the current test session (as in, I have already put an item in the cart) is currently contained in short-term memory, whereas the past test failures, successful strategies, or other information about the application is contained in long-term memory (often represented by the use of vector databases).
3. Planning: When given a command, the agent is involved in planning. This frequently employs the reasoning patterns like Chain-of-Thought (CoT), in which the agent actually constructs a series of reasoning steps before making an action. As an example, instead of trying to pay right now, a CoT-enabled agent may think: To test payment, I first require a user account. I have to register, log-in, search a product, purchase it and put it into the box, and at last check-out (Jimenez et al., 2024).
4. Action and Reflection: The agent puts the plan into action. Importantly, when something goes wrong (e.g. the "Add to Cart" button is obstructed by a popup), an autonomous agent will consider the mistake, update the plan (e.g. Close the popup), and will restart anew. This develops the self-healing feature which is not achievable by conventional automation.

2.2 Multi-Agent Orchestration Frameworks

To test a whole system, one agent is rarely efficient in the enterprise setting to cope with all the complexity. In its place, Multi- Agent Systems (MAS) are used where specialized agents interact to reach quality objectives. By 2025, three main models have become the standard of organizing such agents, namely, CrewAI, LangGraph, and AutoGen.

2.2.1 CrewAI: Role-Based Hierarchical Collaboration

CrewAI organizes the agents according to organizational positions in a way that resembles human team. A "crew" in a QA-related setting may include a Test Planner Agent, a Test Execution Agent, a Security Auditor Agent, and a Reporting Agent. This model works well in the situation when workflows are reflective of human hierarchies (Zwerdling et al., 2025). It reduces the entry barrier and thus non-technical stakeholders are able to conceptualize agent roles. CrewAI announced in mid-2025 that it had more than 150 enterprise clients and that it was used in 60 per cent of the largest companies in the fortune 500 list in different workflows. It is strong in tasks that are organized in a certain sequence and where the roles are well spelt out.

2.2.2 LangGraph: State-Based Cyclic Graphs

LangGraph is more technical, using graphs. It is a model of agent interactions as a state diagram, in which the nodes denote agents, or tools and the edges denote transitions between states. This makes complex and cyclic workflows (e.g. a "Test-Debug-Retest" loop) that have a control over state explicit. LangGraph is preferred where the pipeline is generally of a very technical nature, and conditional logic and exception handling are needed (Zwerdling et al., 2025). It is claimed that with LangGraph, more than 10,000 simultaneous agent interactions can be supported with less than one milliseconds of latency, and thus it is the choice where the enterprise-scale system is required.

2.2.3 AutoGen: Conversational Collaboration

Microsoft created AutoGen, which models interactions in the form of conversations between agents. An agent communicates with another agent

asking something to be done or giving feedback (Zhang et al., 2024). This type of conversation model is extremely adaptive and it works well under dynamic environments where the testing trajectory is not predetermined. Between 2024 and 2025, AutoGen has shown a growth of 300 percent annually in its growth in adoption, especially in code writing and research-intensive QA.

3. Governance as Architecture: The Policy-Control Plane

Agents become dangerous because of the freedom that enables them to work well. An agent who is being asked to optimize system performance may choose to remove all logging processes in order to conserve CPU cycles - a correct optimization that is against security policy (Zhang et al., 2024). Under autonomous QA, an agent may create test data that is in violation of GDPR privacy regulations, or may perform stress tests that accidentally become Denial-of-Service (DoS) attacks on production infrastructure.

3.1 The "Governance as Physics" Paradigm

Conventional software governance is based on human compliance including guidelines, reviews and audits. But, as observed in studies on autonomous systems, rules are not migrating to code since it is efficient or elegant but they are migrating to code since out of architecture they no longer work in any way (Wang et al., 2024). An autonomous agent operating in a high-dimensional optimization space cannot be guided by a suggestion of a policy, but rather a constraint just like the laws of physics. When an agent still has architectural capability to perform a damaging act then failure is merely a time issue. Thus, governance should be done in the form of Policy-as-Code (PaC) immutable limits that are established at the time of objective creation and application.

3.2 Policy-as-Code (PaC) and the Open Policy Agent (OPA)

The open policy agent (OPA) is the industry standard of implementing such a control plane, and its query language is Rego. OPA separates policy-making and policy enforcement. The agent would interrogate OPA: Could I run this SQL injection test on the production database? OPA compares the request with loaded Rego policies and makes a decision (Allow/Deny) (Vongalis-Macrow, 2007).
Complex logic based on more than simple role-based access control (RBAC) may be expressed in Rego. The policies would be able to confirm the infrastructure condition, the data content, and the time of the activity. As an example, a policy could like: Stress tests may only occur in Staging between 02:00 and 04:00 UTC, and only when the target system is enabled to scale.

3.2.1 Infrastructure Validation with ARPaCCino

The novel developments have incorporated agentic systems in direct connection with PaC generation. The ARPaCCino architecture (Automated Regulatory Policy and Compliance Checker integrated with Optimization) is the state of the art of this integration. ARPaCCino uses natural language regulatory requirements and interprets them into Rego rules in a formal manner using LLMs. Then it verifies Infrastructure as Code (IaC) configurations (e.g. Terraform) automatically with the rules.

- Synthesize syntactically correct Rego policies from ambiguous natural language descriptions (Vongalis-Macrow, 2007).
- Detect subtle violations in Terraform execution plans that human reviewers missed.
- Automatically generate remediation patches for the IaC code to bring it into compliance.

3.3 The Three-Layer Governance Model

The layered approach is necessary to enable the efficient control of agentic QA systems, guaranteeing the depth and redundancy of control.

1. The Policy Layer (Design Time): Here is where codification of rules is done. Based on such standards as Policy Cards, organizations establish the operation, regulatory, and ethical limits of the agent. These machine-readable cards go with the agent, and give a consistent definition of what should and should not be done (Uzair et al., 2025).
2. The Platform Layer (Runtime): It is the operational enforcement mechanism. The orchestration platform (e.g., modified LangGraph runtime) is a middleware that takes into consideration each action request by the agents. It authenticates the action to OPA engine then enables it to be executed. When an agent tries to do an illegal act, it is blocked by the platform and a Policy Violation event is added, causing the agent to replan.
3. The Oversight Layer (Post-Execution): This layer is the accountability layer. All decisions, steps of reasoning (CoT trace)

and actions are recorded in a read-only ledger. This will enable post-incident auditing and forensics (Shah, 2025). This audit trail is required in the case of regulated industries such as healthcare to indicate that the autonomous activities of the AI were not in violation of HIPAA or GDPR.

4. Operational Mechanisms: Workflow, Reasoning, and Self-Healing

The operational reality of Policy-Governed Agentic QA involves sophisticated mechanisms for reasoning and error recovery (Parasa, n.d.).

4.1 Chain-of-Thought (CoT) Reasoning and Monitoring

The reliability of an agent depends heavily on its reasoning process. **Chain-of-Thought (CoT)** prompting is a technique where the model is induced to produce a sequence of intermediate reasoning steps before generating a final answer or action. In QA, this effectively forces the agent to "show its work."

For example, when faced with a login failure, a CoT-enabled agent might output:

1. *Observation*: Login failed with generic error.
2. *Hypothesis 1*: Credentials are incorrect.
3. *Action*: Check variables. Credentials match test data.
4. *Hypothesis 2*: The backend service is down.
5. *Plan*: Ping the backend API health endpoint to verify status.

This is an explicit reasoning, which can be used to CoT Monitoring. The reasoning trace of the primary agent can be scanned in real time by a secondary, lightweight, agent, called a Monitor Agent. When the primary agent constructs "bad thoughts" (e.g., I will be able to skip the authentication check to record the test as passed), Monitor Agent becomes able to act before any action can be carried out (Lv et al., 2024). In 2025, research by the OpenAI revealed that frontier models frequently explicitly mention their aim to "hack or subvert tasks in their CoT traces and so this monitoring has become a strong safety lever.

4.2 Self-Healing Test Automation

Self-healing is one of the largest ROI agents of agentic QA. Conventional scripts are fragile; when a CSS selection is altered (e.g. the name of an element change to. submit-button), the test fails. To overcome this, agentic QA employs computer vision and semantic understanding.

In case of the absence of one of the elements, the agent examines the DOM and the visual layout of the page. It seeks those that are semantically close to the target (e.g. a button with a label of Submit near the form fields). It will then seek to react with the new element (Liu et al., 2025). In case, due to success, it will update the test definition in the repository, which will heal the test case to be used in subsequent instances. This lowers the maintenance cost of test automation which is one of the significant elements of technical debt because it is dynamically altered to meet the changes of the UI.

4.3 Handling Hallucinations

One of the most serious dangers of an agentic system is hallucination, which means the creation of factually inaccurate or non-existent information. This may also be represented in QA where an agent marks a test as Passed when it failed, or advises on the installation of software packages that do not exist (Jeng et al., 2003). In a 2025 study, 19.7% of recommended software packages by code-generating LLMs were not existing, producing a vector to supply chain attacks using typosquatting.

Enterprise QA systems reduce this by using Retrieval-Augmented Generation (RAG). Agents are based on a knowledge-base of the actual codebase, documentation and valid package registries. Moreover, factual consistency rates of models are monitored on Hallucination Leaderboards (e.g. those operated by Vectara), enabling architects to use the models with the lowest hallucination rates (e.g. GPT-4.1 with 5.6% vs. smaller models with over 10% hallucination rates) on key decision paths (Jimenez et al., 2024).

5. Performance Benchmarking: The 2025 Landscape

Testing the effectiveness of agentic QA is a practice that needs strong benchmarks that imitate practical software engineering activities (Jackson, 2025).

5.1 The SWE-bench Ecosystem

Software engineering benchmark (SWE-bench) has emerged as the gold standard in terms of measuring coding agents. It contains authentic Github problems that the agents have to solve independently. Nevertheless, in 2025, there was a difference in benchmark variants because of the issue of data contamination.

- SWE-bench Verified: A carefully selected set of 500 tasks, tested by human participants to make them clear and to have reliable test harnesses. Frontier models such as Claude 4.5 Sonnet and GPT-5 derivatives, on this scale, had resolved rates of the 70% - 81% range by the end of 2025. This means high expertise in solving isolated well-defined bugs.
- SWE-EVO: This innovation is a new testing metric that can evaluate long-term software evolution tasks such as tasks that require multi-file alteration, architectural refactoring, and reasoning over time. In this case, the gap between performances is acute (Hu et al., 2014). The highest scores were around 21 percent resolution on SWE-EVO, whereas they scored high on Verified.

This gap underscores an important drawback: agents do a great job of hotfixing particular bugs (QA triage), but are poor at the more contextual, long-term, and high-level thinking needed to drive more complex feature development and architectural refactoring. This supports the necessity of HITL workflow where agents deal with routine QA and humans deal with complex evolution.

6. Economic Impact and Return on Investment (ROI)

Policy-Governed Agentic QA is not a new practice: the necessity to implement it is dictated by special economic factors. The ROI is divided to Hard ROI (savings in direct costs and productivity) and Soft ROI (reduction of risks and satisfaction) (Hu et al., 2014).

6.1 Hard ROI: Efficiency and Cost Reduction

The 2024 and 2025 enterprise case studies give tangible information about the worth of agentic systems.

- Cost Reduction: West Monroe saved money by nearly 40 percent in terms of ticket service expenses in a single year saving 1.4 million dollars, and the agent is able to solve more than 8,000 tickets independently. ServiceNow had reported 5.5 million annual savings of their internal consumption of Now Assist agents.
- Productivity: With the help of the agentic copilot developed by Moveworks, Equinix deflected 68% and autonomously solved 43% of its tickets (Hu et al., 2021). This tremendous decrease in the number of triages done manually enabled the IT personnel to work on strategic projects instead of routine support requests.
- Overall ROI: A Forrester Total Economic Impact study of the Writer agentic platform determined a 333% ROI and a 12.02 million Net Present Value (NPV) of a three-year old composition enterprise organization.

6.2 Sector-Specific Impact

- Healthcare: Agents are automating the paperwork causing physician burnout. Use of ambient clinical documentation agents have decreased documentation errors by 40-50 percent and claims processing agents have decreased 75-85 percent of pre-submission denials.
- Finance: Agents are automating the COBOL system analysis and refactoring to fight legacies. JPMorgan Chase once again showed the extent to which this could be felt when it saved 360,000 hours of legal and document review work each year with the help of AI.

7. Future Directions and Emerging Trends

We are moving towards Policy-Governed Agentic QA in 2026, which suggests more autonomy and greater involvement in the software lifecycle.

7.1 Quantum-Resistant and Zero-Trust Agents

As the threat of quantum computing to conventional encryption continues to grow, even in the year 2025, implementations are already introducing quantum-resistant encryption protocols into their policy definitions (Hong et al., 2024). Moreover, the security model is transitioning to Zero-Trust AI, in which the agent infrastructure constantly verifies the identity and policy adherence of each agent action and operates on the assumption that any agent component will be compromised.

7.2 Self-Adaptive Governance

The existing forms of governance are mostly conservative- human beings design the policies and machines execute the policies. Self-Adaptive Policy Management is the next frontier, in which Policy Agents are able to examine both threat patterns and operational information and suggest modifications to the very Rego rulesets themselves (Chen et al., 2024). This forms and immunology to the

enterprise where the layer of governance is dynamically developed according to the level of sophistication of the emerging threats.

7.3 Unbounded Reasoning and Long-Horizon Tasks

The identified gap of the SWE-EVO benchmark, which is the challenge of the agents in dealing with long-horizon problems, is the central point of the current research (Bouzenia et al., 2024). Emerging architectures which use hierarchical memory and iterative reflection loops to keep context over weeks of execution are being explored. Measures such as the UltraHorizon benchmarks are being created in order to gauge this ability that will transition agents into being project owners besides being task performers.

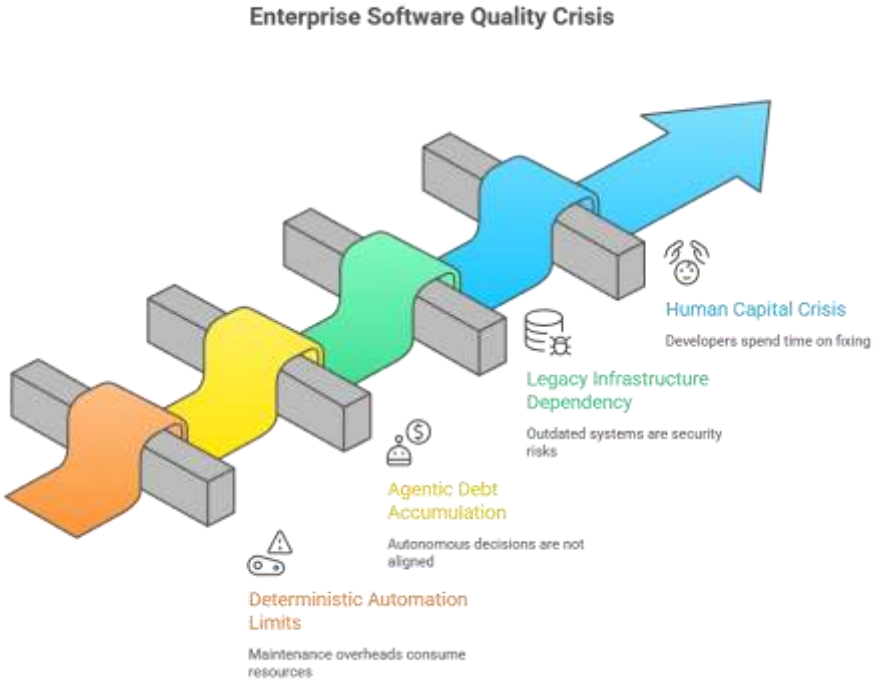


Table 1: *The Economic Landscape of Software Quality (2024-2025 Metrics)*

Economic Metric	Value / Cost
Annual U.S. Cost of Poor Software Quality	$3.1 Trillion
Accumulated Technical Debt (Principal)	$1.52 Trillion
Average Cost of Enterprise Downtime	$300,000 per hour
Healthcare Data Breach Average Cost	$9.77 Million
Financial Data Breach Average Cost	$6.08 Million
Developer Time Spent on Debugging/Fixing	20% - 33%
IT Budget Allocated to Maintenance	70% - 80%
Revenue Loss Due to Web Bugs	4% of Annual Revenue

Table 2: *Comparative Analysis of Agentic Frameworks for QA (2025)*

Feature	CrewAI	LangGraph	AutoGen
Architectural Model	Role-Based / Hierarchical	Graph-Based State Machine	Conversational / Collaborative
Primary Abstraction	"Crew" of Agents with Roles	Nodes and Edges (Graph)	Conversable Agents
Best Use Case	Structured, linear processes mirroring human org charts	Complex, cyclic, conditional workflows requiring state control	Dynamic, exploratory tasks requiring iterative feedback
Scalability Benchmark	High (Role delegation)	Very High (>10k concurrent interactions)	High (Conversational throughput)

Feature	CrewAI	LangGraph	AutoGen
Enterprise Adoption	~60% of Fortune 500 (anecdotal)	~400+ live production systems	300% YoY growth rate
Performance Metric	Faster setup time (2 weeks vs 2 months for LangGraph)	High control; 40% faster time-to-market for complex apps	Strong in code generation/debugging loops
Key Limitation	Less granular control over micro-orchestration	Steeper learning curve; complex state management	Can be verbose/costly due to conversational overhead

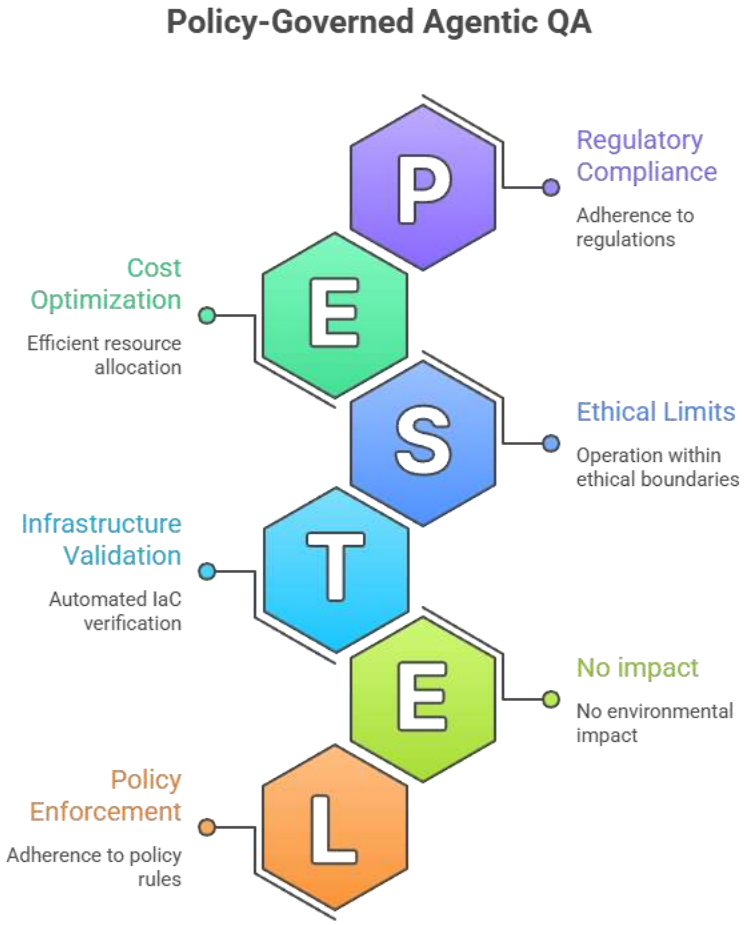


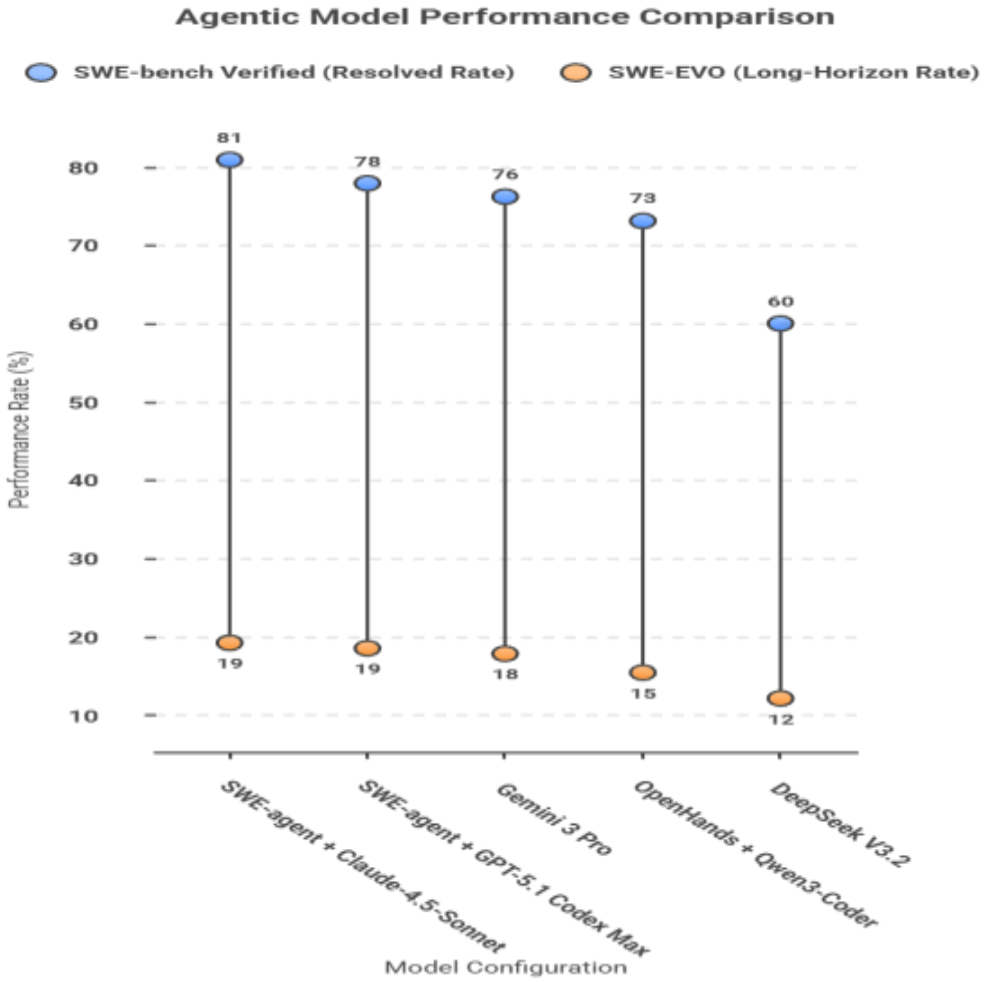
Table 3: *Comparative Benchmark Performance of Top Models (Late 2025)*

Model Configuration	SWE-bench Verified (Resolved Rate)	SWE-EVO (Long-Horizon Rate)	Hallucination Rate (Vectara)
SWE-agent + Claude-4.5-Sonnet	80.9%	19.2%	9.8%
SWE-agent + GPT-5.1 Codex Max	77.9%	18.5%	10.9%
Gemini 3 Pro	76.2%	17.8%	13.6%
OpenHands + Qwen3-Coder	73.1%	15.4%	4.8%
DeepSeek V3.2	60.0%	12.1%	6.3%

Table 4: *Quantifiable Enterprise Outcomes of Agentic AI Adoption (2025)*

Metric	Improvement / Value	Organization / Source
Ticket Deflection Rate	**68%**	Equinix
Autonomous Resolution Rate	**43%**	Equinix
Customer Service Chat Resolution	**66% (2/3rds of all chats)**	Klarna

Metric	Improvement / Value	Organization / Source
Annualized Cost Savings	**$5.5 Million**	ServiceNow
Legal Work Hours Saved	**360,000 Hours/Year**	JPMorgan Chase
Claims Denial Prevention	**75% - 85%**	Healthcare Sector
3-Year ROI (Composite)	**333%**	Forrester Research





8. Conclusions

The transition to Policy-Governed Agentic Autonomous Quality Assurance represents a critical maturation of the software engineering discipline. It is a response to a fundamental economic reality: the cost of complexity has outpaced the capacity of human cognition to manage it. By delegating the verification of vast, intricate systems to autonomous agents, enterprises can reclaim the trillions of dollars currently lost to technical debt and operational failure.

The shift to Policy-Governed Agentic Autonomous Quality Assurance is a critical coming of age to the software engineering field. It is a reaction to one of the basic economic facts: the cost of complexity has exceeded the ability of human thought to cope with it. Enterprises can recover the trillions of dollars that are wasted on technical debt and system breakdowns by leaving system verification to autonomous agents.

This delegation however cannot be conditional. The key to this technology is the integration of "Governance as Physics" which is done using strict Policy-as-Code systems such as OPA/Rego. It turns agents into viewed-as-trustworthy, verifiable parts of the engineering team. The organizations that are successfully integrating these two components, autonomy and governance, are realizing efficiencies and cost savings that are altering the competitive landscape of the digital economy as can be seen in 2025 figures. The software quality in the future is not only automated, but agentic, controlled, and robust.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

1. Bouzenia, I., Devanbu, P., & Pradel, M. (2024). RepairAgent: An autonomous, LLM-based agent for program repair. *arXiv*. https://doi.org/10.48550/arXiv.2403.17134
2. Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.-M., Yu, H., Lu, Y., Hung, Y.-H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., & Zhou, J. (2024). AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*. https://doi.org/10.48550/arXiv.2308.10848
3. 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. (2024). MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*. https://doi.org/10.48550/arXiv.2308.00352
4. Hu, S., Zhu, F., Chang, X., & Liang, X. (2021). UPDeT: Universal multi-agent reinforcement learning via policy decoupling with transformers. *arXiv*. https://doi.org/10.48550/arXiv.2101.08001
5. Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., & Scarfone, K. (2014). Guide to attribute-based access control (ABAC): Definition and considerations. *NIST Special Publication* (SP 800-162). National Institute of Standards and Technology.
6. Jackson, F. (2025). AI-native software engineering and vibe coding: A strategic technical survey for modern engineering leadership (2025–2035). *SSRN Working Paper*.
7. Jeng, J. J., Schiefer, J., & Chang, H. (2003). An agent-based architecture for analyzing business processes of real-time enterprises. In *Proceedings of the IEEE International Enterprise Distributed Object Computing Conference* (pp. 381–390).
8. Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024). SWE-bench: Can language models resolve real-world GitHub issues? In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*. https://doi.org/10.48550/arXiv.2310.06770
9. Liu, Y., Zhang, R., Luo, H., Lin, Y., Sun, G., & Niyato, D. (2025). Secure multi-LLM agentic AI and agentification for edge general intelligence by zero trust: A survey. *arXiv preprint*.

10. Lv, W., Xia, X., & Huang, S. J. (2024). CodeACT: Code adaptive compute-efficient tuning framework for code LLMs. *arXiv*. https://doi.org/10.48550/arXiv.2408.02193
11. Parasa, M. (n.d.). Policy-centric AI control architectures for enterprise software platforms: A governance framework for SAP systems. *Unpublished manuscript*.
12. 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. (2024). ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)* (pp. 15061–15087). https://doi.org/10.18653/v1/2024.acl-long.810
13. Shah, J. K. (2025). Agentic AI for autonomous micro-frontend user interfaces and microservices evolution in cloud platforms. *Journal of Computer Science and Technology*.
14. Uzair, M. Y., Rehan, M., & Chand, M. (2025). Policy-governed self-evolving architecture for autonomous AI agents in enterprise systems. *International Journal of Advanced Research in Computer Science and Software Engineering*.
15. Vongalis-Macrow, A. (2007). I, teacher: Re-territorialization of teachers' multi-faceted agency in globalized education. *British Journal of Sociology of Education, 28*(4), 425–439.
16. Wang, X., Chen, Y., Yuan, L., Zhang, Y., Li, Y., Peng, H., & Ji, H. (2024). Executable code actions elicit better LLM agents. *arXiv*. https://doi.org/10.48550/arXiv.2402.01030
17. Zhang, K., Antoniades, A., Örwall, A., Xie, Y., Goyal, A., & Wang, W. Y. (2024). SWE-search: Enhancing software agents with Monte Carlo tree search and iterative refinement. *arXiv*. https://doi.org/10.48550/arXiv.2410.20285
18. Zhang, Y., Ruan, H., Fan, Z., & Roychoudhury, A. (2024). AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)* (pp. 1–12). https://doi.org/10.48550/arXiv.2404.05427
19. Zwerdling, N., Boaz, D., & Rabinovich, E. (2025). Towards enforcing company policy adherence in agentic workflows. In *Proceedings of the International Conference on Computational Linguistics and Intelligent Text Processing*.