Infosys topaz

WHITE PAPER

SMART ARTIFACTS: THE STRATEGIC SHIFT TOWARD AGENTIFIED INFORMATION

Infosys
Navigate your next

Abstract

In the evolving landscape of software engineering, the emergence of smart artifacts and smart repositories signals a transformative departure from static documentation toward dynamic, adaptive digital assets.

Traditionally, technical artifacts, code repositories, and artifact repositories have served as passive records, requiring manual updates and interpretation to remain relevant within fast-paced development environments. However, the integration of intelligent, context-aware technologies now enables these artifacts to function as active, self-evolving participants in the software lifecycle. Smart artifacts and repositories not only capture and reflect the current state of a project, but also autonomously adapt, interpret, and guide development processes in real time. This paradigm shift redefines information management in software engineering—empowering documents and repositories to become proactive collaborators that sustain relevance, foster efficiency, and drive innovation throughout the project's evolution.

We are entering an era where one no longer simply "author" an artifact; instead, one "seeds" it. The artifact evolves and adapts in tandem with the project it represents, actively guiding development through its growing intelligence and contextual awareness.

The Theoretical Foundation of Smartness in Artifacts

To understand the trajectory of smart artifacts, it is essential to define not only the dimensions of "smartness" but also to recognize "intelligence" as an indispensable facet even for non-computational entities. A smart artifact is a physical or digital object that has been augmented with technologies conferring autonomy, awareness, intelligence, and the ability to interact with its environment.

This augmentation preserves the object's original purpose while layering digital intelligence and cognitive capabilities, thereby enhancing decision-making, adaptability, and interaction.

Research demonstrates that the intelligence and adaptability of smart artifacts are anchored in three core pillars: heightened awareness, integrated intelligence, and contextual adaptability. Together, these foundational elements empower an artifact to function as an interactive, evolving agent within complex socio-technical systems, actively shaping and responding to its environment.

From Static Containers to Cognitive Agents

Traditionally, documents, code, process, script, bots, models, and agents have functioned merely as static containers, as repositories where information is stored and remains inert until manually updated. In contrast, smart artifacts represent a transformative leap, moving beyond passive storage to become dynamic, interactive entities characterized by four foundational pillars: Context Awareness, Self-Maintenance, Actionability, and now, Intelligence.

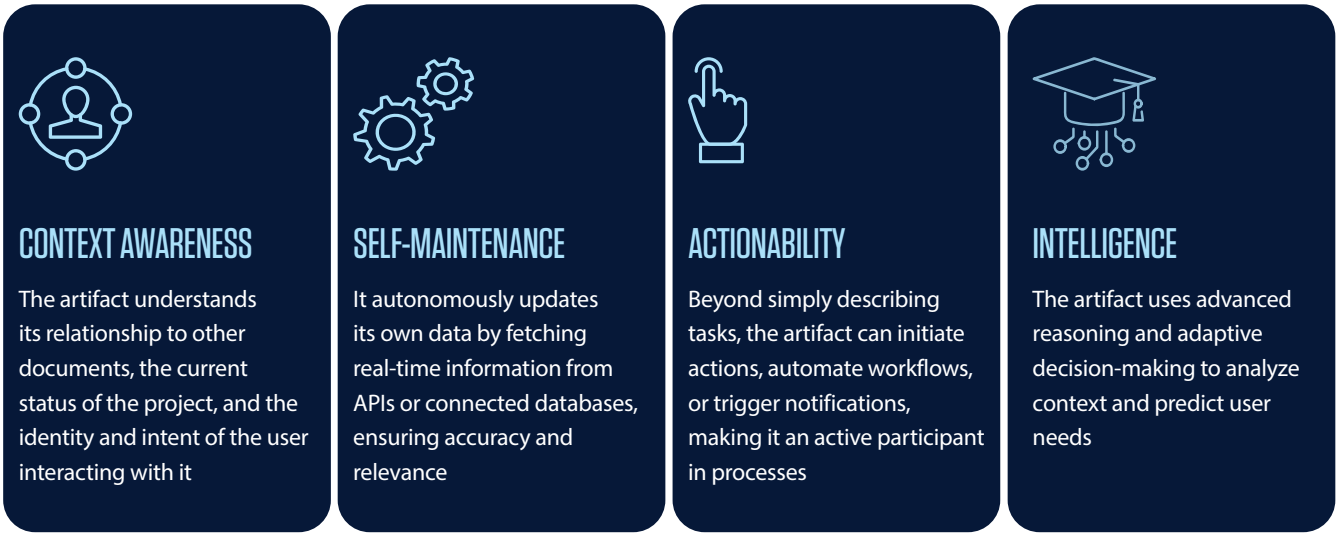


Fig. 1: The Four Pillars of Smart Artifacts

Example: Smart Documents: The "Living" Narrative

Imagine a Project Requirement Document (PRD) that is not just a static wall of text but a living, intelligent entity:

- **Personalized Content:** If a developer accesses the PRD, it highlights technical constraints and relevant API endpoints. If a stakeholder views it, the document emphasizes ROI and project timelines, adapting its presentation intelligently based on user roles and preferences.
- **Automated Consistency Checks:** When a date is changed in the project schedule, the Smart PRD intelligently flags conflicts in the dependencies section, using integrated logic and awareness of project dynamics.
- **Real-Time Synchronization:** Embedded live queries display actual production user counts and automatically update as underlying data changes.
- **Proactive Guidance:** By analyzing project workflows and user interactions, the document can suggest optimal paths, flag potential risks, and offer solutions before problems arise.

Awareness Dimension	Functional Capability	Technological Implementation
Activity Awareness	Sensing and logging real-time conditions and events related to use and handling.	SDLC entails defining, designing, developing, testing, and maintaining systems to monitor and respond to real-time user and system events. This approach ensures responsive, interactive apps with reliable performance and adaptability to evolving user behaviors.
Policy Awareness	Understanding and enforcing compliance with rules, regulations, and threshold-breaking activities.	Rule-based engines, local processing of regulatory logic.
Process Awareness	Recognizing workflows, transitions between activities, and offering guidance on optimal paths.	Agentic AI, state management, cognitive modeling.
Integrated Intelligence	Analyzing context, learning from data and user interactions, predicting future needs, and making adaptive decisions.	Machine learning algorithms, natural language processing, reasoning engines.

Fig. 2

This expanded hierarchy of awareness and intelligence positions smart artifacts as transformative elements throughout software engineering. Rather than being passively controlled by external systems, smart artifacts now actively contribute to the development process by supporting engineers and project teams. In software engineering, these intelligent components can facilitate requirements gathering, monitor compliance with coding standards, and adapt workflows based on real-time feedback. As smart artifacts are integrated—from code analysis tools to intelligent deployment agents, they not only influence how software is designed, tested, and maintained, but also reshape the mindsets and practices of development teams. This creates a seamless blend between human-driven processes and intelligent automation, tightly weaving together the digital and operational aspects of software engineering.

The Transition to Digital Intelligence

In the digital realm, the manifestation of smartness has evolved from simple metadata tagging to the creation of self-contained units of intelligence. The Self-Thinking Data Manifest (STDM) represents the pinnacle of this evolution in documentation. An STDM is a digital artifact that bundles primary data with explicit, structured instructions that direct how an external interpretation engine, such as an LLM, should process and present that data.

This creates a portable, self-directing experience that preserves author intent while unlocking new analytical capabilities. Unlike traditional files that are "read" by software, STDM "instructs" the software on how it wishes to be interacted with, effectively turning the document into a specialized librarian or assistant for its own content.

EMBEDDED INTELLIGENCE

Bundles data with directives (goals, constraints, workflows)

SELF-MAINTENANCE

Updates itself using APIs or connected sources

CONTEXT AWARENESS

Maintains relationships to other documents and user roles

VISUAL INDICATOR

STDM-enabled documents often display an "I am Thinking Data" ribbon for easy identification

ACTIONABILITY

Can trigger tasks or analyses, not just describe them

Fig.3: *STDM Core Characteristics*

Smart Repositories

The shift toward smartness is perhaps most visible in the evolution of software repositories. We have transitioned through several distinct epochs of software engineering, each defined by the level of autonomy granted to the repository and its associated tools.

The Hierarchy of Engineering Paradigms

The progression from SE 1.0 to SE 3.0 represents a broadening of context and a deepening of autonomous capabilities.

Dimension	SE 1.0 (Manual)	SE 1.5 (Predictive)	SE 2.0 (AI-Assisted)	SE 3.0 (Agentic SE)
Unit of Work	Manual code entry—files and lines managed by hand	Tokens and individual lines of code	Functions, classes, and discrete files	High-level features and business tickets
Autonomy	None; all actions performed manually by the developer	None; provides only inline hints and autocomplete	Low; relies on single-shot generation based on prompts	High: utilizes plan–execute–verify loops and autonomous agents
Context Scope	Current file or code section only; no automatic context awareness	Immediate cursor position and current file	Entire file or local repository context	Whole system architecture and external documentation tools
Team Structure	Individual developers or small ad hoc teams	Collaborative teams with defined roles, often working in silos	Cross-functional teams integrating AI specialists and engineers	Autonomous squads or agent teams orchestrating multiple roles and systems
Primary Benefit	Full control and understanding; no automation	Incremental typing speed and syntax accuracy	Rapid prototyping and simplified refactoring	Scalable automation of complex development tasks

Fig. 4

Reimagining Repos

In the SE 3.0 paradigm, repositories are transformed into "smart repositories." These are not passive storage for source code but active environments where autonomous coding agents, such as GitHub Copilot, Devin, Cursor AI, and Claude Code, that function as virtual teammates.

These agents can independently navigate a codebase, update multiple files, run tests, and manage tasks from start to finish. This marks the shift from code completion to goal-oriented task execution, where the developer defines the "what" in natural language, and the repository, through its agentic layers, determines the "how."

The evolution of repositories like GitHub and JFrog Artifactory is currently moving from "passive storage" to "agentic intelligence." In this reimagined landscape, a repository is no longer just a place to store code and binaries; it is an active participant in the software development lifecycle.

In the era of GenAI and Agentic AI, these platforms are becoming smart repositories—autonomous environments that reason, act, and self-heal.

The Core Paradigm Shift: From Reactive to Proactive

Traditional repositories wait for a developer to push code or a CI/CD pipeline to pull a package. A "smart repository" flips this logic. The repository monitors its own health and acts before a human even notices a problem.

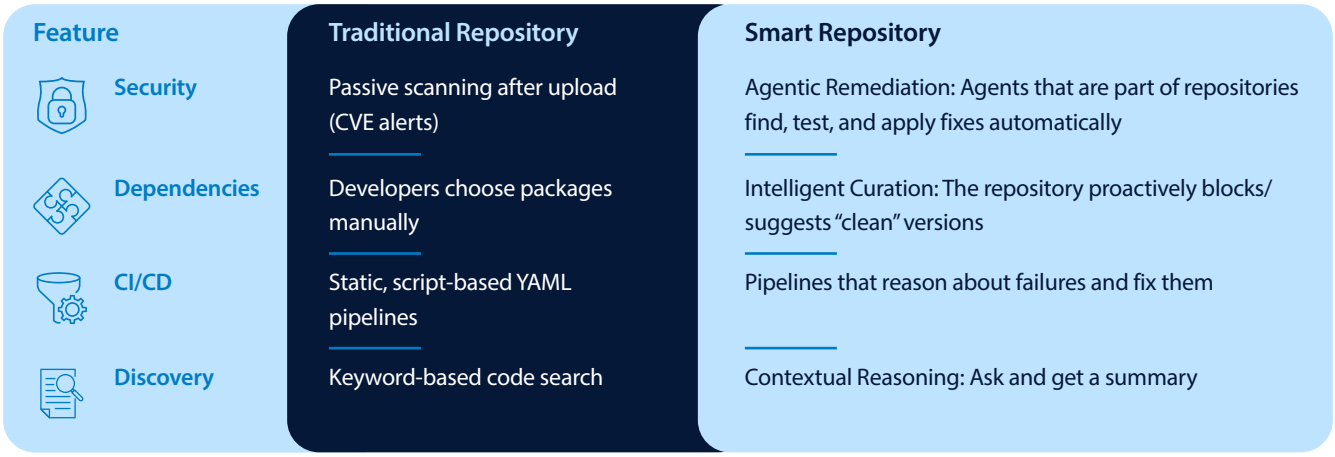


Fig. 5

Key Pillars of the Smart Repository

1. Agentic Remediation & Self-Healing

GitHub and JFrog have begun integrating agentic workflows where AI agents don't just point out a security vulnerability; they clone a temporary branch, write the fix, run the tests to ensure no regression, and present a completed pull request for approval.

Example: If a zero-day exploit is found in a Java library, the artifactory agent identifies every project using that binary and triggers an autonomous upgrade flow across the entire organization.

2. Intelligent Artifact Curation (The "Clean Room" Approach)

Instead of just storing binaries, smart repositories act as a firewall of intent. Platforms like JFrog Artifactory and GitHub serve as intentional security layers for software artifacts.

Blast Radius Analysis: JFrog Artifactory can assess the potential impact of a package update, helping teams understand the risks before deployment.

Automatic Quarantine: If a new artifact in JFrog Artifactory or a repository in GitHub exhibits anomalous or unpredictable behaviors, the system can proactively isolate the package before it is accessible to developers.

3. The Repository Knowledge Graph

Smart Repositories are moving away from flat file structures toward knowledge graphs. This "brain" maps the relationships between Jira issues, Microsoft Teams/Slack discussions, code commits, and binary versions.

Impact: An AI agent can use this graph to answer complex questions: "Why was this specific encryption method chosen three years ago?" It would cross-reference the commit with a historical Teams/Slack thread and a 2025 security policy stored in the repository.

4. The Role of MCP (Model Context Protocol)

A critical "glue" in this reimagining is the Model Context Protocol. MCP allows AI agents (like GitHub Copilot) to securely "talk" to the repository's data.

GitHub + JFrog Integration: Developers can now use a coding agent that understands not just the code in the IDE, but also the specific org-approved binaries available in Artifactory. The agent won't suggest a library that is restricted by company policy because it has "live" context from the repository.

5. Current Real-World Examples

JFrog Fly: A pioneering agentic repository where AI agents handle and automate software delivery tasks like storing, sharing, correlating, managing versions and deploying artifacts.

GitHub Agent HQ: A centralized platform for managing and coordinating multiple agents (such as coding, security agent) directly within the repository.

Harness AI DevOps Agent: Uses a "Software Delivery Knowledge Graph" to troubleshoot failed builds and propose YAML fixes in natural language.

Bitbucket Rovo: Atlassian's agentic layer that can autonomously resolve merge conflicts and generate internal documentation based on recent code changes.

6. The Future: CI/CD

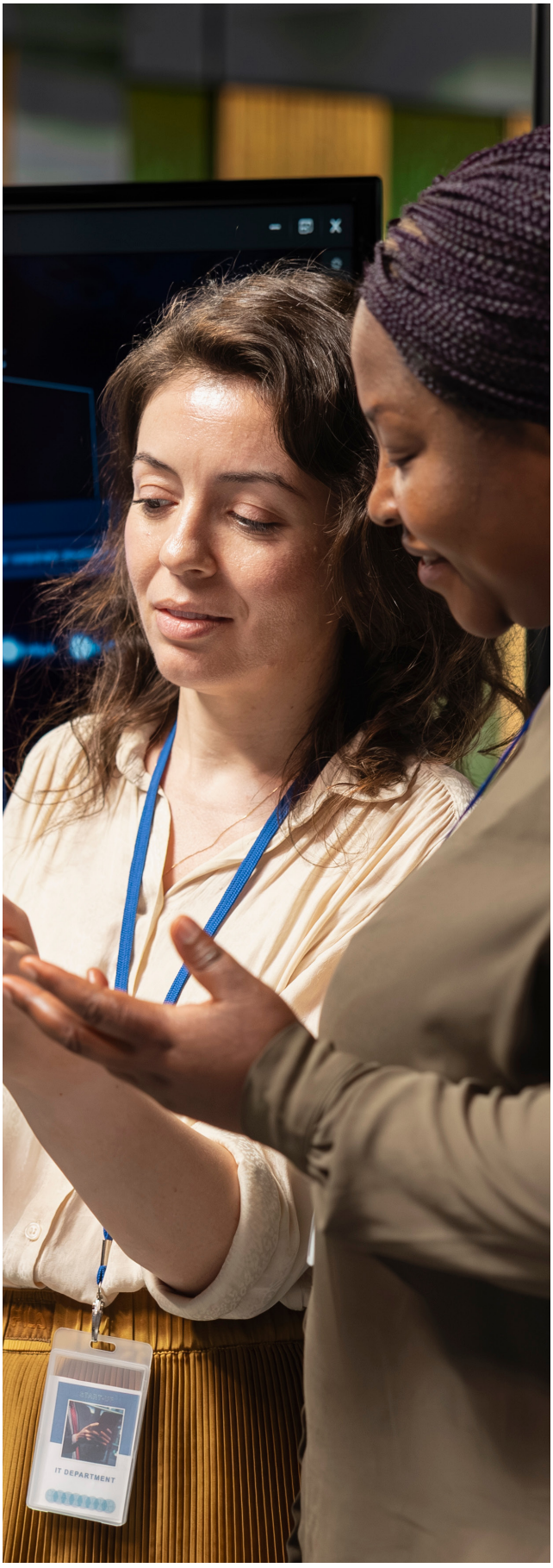
We are moving toward a world where "repository" and "pipeline" are the same thing. In a CI/CD model:

Intent: A developer describes a feature in natural language.

Agentic Execution: The repository's agents generate the code, select the safest dependencies from the artifact store, and build the environment.

Autonomous Testing: If a test fails, the agent reasons about the error, adjusts the code, and re-runs the cycle until it passes.

Human-in-the-Loop: The developer acts as a "Reviewer-in-Chief" rather than a manual laborer.



Architectural Implementation: RAG and the Model Context Protocol

The technical "smartness" of an artifact is enabled by sophisticated architectural patterns that allow it to ground its intelligence in reality. The two most prominent patterns in this domain are Retrieval-Augmented Generation (RAG) and the Model Context Protocol (MCP).

Retrieval-Augmented Generation (RAG) within Documents

RAG is an AI framework that merges traditional information retrieval with LLMs' generative abilities. When applied to a smart document, RAG ensures that any generated insight is grounded in the specific, verifiable content of that document, thereby mitigating the risk of "hallucination."

The RAG pipeline for an intelligent artifact involves several distinct steps:

- **Ingestion and Chunking:** Documents are segmented into semantically coherent units (e.g., 1000 characters with a 100-character overlap) to maintain context while respecting the model's context window.
- **Vectorization:** Each chunk is converted into a high-dimensional vector representation using an embedding model. These vectors numerically capture the semantic meaning of the text.
- **Storage and Retrieval:** Vectors are stored in a vector database (e.g., ChromaDB). When a user queries the document, the query is vectorized, and a cosine similarity search is performed to retrieve the most relevant chunks.

The Model Context Protocol (MCP) as a Universal Interface

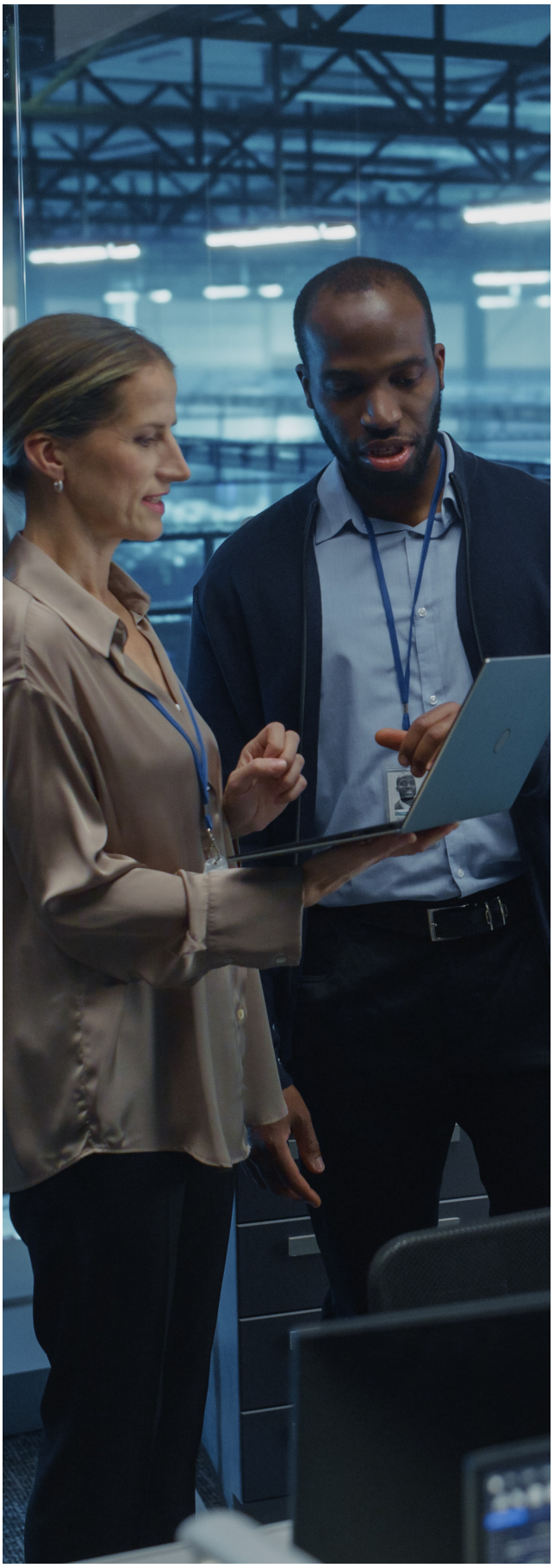
As the number of specialized AI agents and smart tools grows, the need for a standardized communication protocol has become paramount. The Model Context Protocol (MCP), introduced in late 2024, provides an open standard that allows any AI agent to talk to any data source or tool.

Component	Role in the Ecosystem
Host	The AI application (e.g., GitHub copilot) that requires external capabilities
Client	A component within the Host that manages the connection to specific servers
Server	A specialized service that exposes "Resources" (data), "Tools" (functions), and "Prompts" (templates)

Fig. 6

GitHub MCP Server — repository truth, not web search; serves as a prime example of this pattern. Reference MCP servers that provide Resources (repository files, issues), Tools (search/modify repositories), and sometimes Prompts for code workflows. Agents read/act on actual repository state via GitHub APIs. It formalizes a direct, verifiable interface to repositories, ensuring agents consult the real repository data (branches, files, issues), not stale web mirrors.

MCP servers act as trusted interpretation endpoints within the STDM framework, with documents specifying the servers to use. This ensures queries and actions are routed to authoritative back-end systems, preserving author intent and grounding interactions in trusted source systems rather than generic web search.



The Agentic Value Stream: Orchestrating Smart Artifact Promotion from Intent to Production

In an autonomous SDLC, the value stream transforms from a series of manual hand-offs into a continuous intelligence loop. Artifact promotion is no longer a "check-the-box" activity; it becomes a reasoning-based progression where AI agents validate the "integrity, intent, and impact," of code and binaries at every stage.

Here is how autonomous repositories and artifact promotion function across the value stream.

The Autonomous Value Stream Stages

Stage 1: Intent to Code (The "Authoring" Repo)

- **Repository Role:** The repository acts as a context engine.
- **Autonomous Promotion:** As the developer (or AI Agent) writes code, a pre-flight agent scans the diff against the "Enterprise Knowledge Graph". This graph serves as organization's "technical DNA", mapping internal architecture standards, approved security patterns, and historical project decisions.
- **The Gate:** If the code uses a deprecated library or violates a security pattern, the agent blocks the commit and rewrites the code snippet autonomously before it even reaches the remote branch.

Stage 2: Commit to Build (The "Validation" Repo)

- **Repository Role:** Ephemeral environment orchestration.
- **Autonomous Promotion:** Upon a PR (Pull Request), a Validation Agent doesn't just run tests; it reasons about which tests to run based on the changes.
- **The Gate:** If a build fails, a self-healing agent analyzes the logs, fixes the configuration (e.g., a missing environment variable or a broken dependency version in the POM/Package file), and restarts the build.

Stage 3: Build to Artifact (The "Curation" Repository - e.g., JFrog)

- **Repository Role:** Intelligent Quarantine.
- **Autonomous Promotion:** Once a binary is created, it is pushed to a "Sandbox" registry. A security agent performs "deep reachability analysis,"checking if a vulnerable function in a library is actually being called by the code.
- **The Gate:** If "reachable," the agent triggers a remediation loop, creating a new version of the artifact with the patched library. If "clean," the artifact is promoted to the "verified" registry.

Stage 4: Verified to Release-Ready (The "Staging" Repo)

- **Repository Role:** Predictive Performance Store.
- **Autonomous Promotion:** AI agents deploy the artifact to a staging environment. They compare the artifact's telemetry (CPU, memory, latency) against the current production baseline.
- **The Gate:** The promotion agent evaluates "release readiness" by calculating a risk score. If the score is within the threshold, it signs the artifact with a digital signature, moving it to the "golden" production registry.

Stage 5: Production & Feedback (The "Live" Repo)

- **Repository Role:** Observability-linked Registry.
- **Autonomous Promotion:** The repository triggers Canary or Blue/Green deployment.
- **The Gate:** A guardrail agent monitors live metrics. If an anomaly is detected, it autonomously triggers a "rollback" by pulling the previous "golden" version from the artifact repository and updating the cluster.

The "Value Stream" Impact

In this reimagined SDLC, the Value Stream Map (VSM) changes fundamentally:

- **Wait Time (Waste):** Usually caused by human approvals or waiting for security scans. In an autonomous repo, wait time drops to near zero because agents operate in real-time.
- **Processing Time:** Remains consistent, but the "Quality of Result" is higher because AI agents perform exhaustive checks that humans often skip.
- **Lead Time:** The time from "Idea" to "Production" shrinks from weeks to minutes, as the Artifact Promotion becomes a fluid, automated movement through increasingly high-trust zones.

Feature	Traditional Promotion	Autonomous Promotion
Trigger	Manual Approval / Timer	Event-driven Reasoning (Metrics + Policy)
Security	Point-in-time scanning	Continuous Security Scanning & Auto-Patching
Evidence	Log files (JSON/YAML)	Knowledge Graph (Provenance + Impact Analysis)
Rollback	Human intervention	Agentic Circuit Breaker (Auto-revert)
Trust Model	RBAC (Role-Based)	ABAC (Agent-Based) + Digital Signatures

Fig. 7

Self-Documenting Systems and the End of Knowledge Vaporization

One of the most persistent challenges in the continuous Software Development Lifecycle (SDLC) is knowledge vaporization: the gradual loss of essential design decisions and operational insights as teams race to deliver features quickly. Traditional documentation is often perceived as a tedious chore that lags behind development, making it prone to being outdated and incomplete.

From Static Manuals to Executable, Self-Updating Knowledge

Smart artifacts address this issue by transforming documentation into an executable and self-updating resource. Unlike static manuals or PDFs, modern documentation platforms—such as Mintlify—are AI-native, integrating intelligent agents across the entire documentation lifecycle. These context-aware agents automatically draft, edit, and maintain documentation, ensuring it always reflects the latest state of the project. The result is a living, interactive dashboard that guides users through project information via dynamic, conversational interfaces.

Strategies to Prevent Knowledge Vaporization

Researchers and practitioners have identified three key approaches to leveraging smart artifacts for preserving organizational knowledge:

- **Just Enough Upfront Documentation:** Capture only the essential shaping ideas and interface definitions required to initiate a project, avoiding unnecessary detail that quickly becomes obsolete.
- **Executable Knowledge:** Use artifacts such as infrastructure-as-code and test-driven development (TDD) scripts as authoritative, living sources of truth that drive both development and documentation.
- **Automated Text Analytics:** Apply deep learning and text mining to automatically extract and document design decisions from sources like Git commit messages and team chat logs.

This paradigm shift ensures that documentation is no longer a disconnected, manual activity, but rather an organic byproduct of the intelligent development process. By making specifications executable and continuously updated, organizations permanently close the gap between "what we intended" and "what we built." This not only preserves critical knowledge but also enhances trust, auditability, and agility in autonomous and agent-driven software environments.

Governance, Security, and the Human-in-the-Loop Mandate

As artifacts and repositories become increasingly autonomous, the need for robust governance and validation frameworks becomes critical. The "hallucination" risk of LLMs means that blindly trusting AI-generated code or documentation can lead to broken builds, security vulnerabilities, and "data privacy risks.

Validation Models: HITL vs. HOTL

To mitigate these risks, organizations implement Human-in-the-Loop (HITL) and Human-on-the-Loop (HOTL) validation frameworks.

- **Human-in-the-Loop (HITL):** In this model, the AI serves as a partner that proposes solutions, but results are "blocked" until a human expert verifies them. This is essential for high-stakes tasks like security reviews or architectural decisions where error tolerance is low.
- **Human-on-the-Loop (HOTL):** Here, the human acts as a supervisor, monitoring the AI's autonomous actions and intervening only when necessary. This is suitable for high-volume, lower-risk tasks like style consistency checks or initial drafting.

HackerOne's methodology for AI code security exemplifies the power of HITL. Their system distills the AI agent's "thought process" into Explainable AI (XAI), which catches the human expert up on the reasoning behind a specific detection.

This collaboration allows the human to validate assumptions and handle "unintuitive edge cases" that the AI might miss, while the AI "shadows" the human to learn architecture-specific nuances and institutional knowledge. This feedback loop turns the agent into a true teammate that gets smarter with every interaction.

Evaluating Success in the Era of SE 3.0

The transition to smart repositories and artifacts necessitates a new set of metrics to measure effectiveness. Traditional metrics like "lines of code," or "number of documents," or DORA metrics (like Deployment Frequency or MTTR) are necessary but insufficient.

In an autonomous environment, you aren't just measuring how fast human + AI Agents move code; you are measuring the "reasoning efficiency" and "trust reliability" of your AI agents.

Here are the new metrics and KPIs categorized by their impact on the value stream.

Autonomy & Agent Performance Metrics
These measure how "smart" and independent your repository agents actually are.

Quality & "Smart" Security Metrics

a. **In a smart repository,** security is about reachability,

Metric	Definition	Why it matters
Autonomous Remediation Rate (ARR)	% of security vulnerabilities or build failures fixed by agents without human intervention	Measures the reduction in "Toil" for developers
Decision Turn Count	The number of autonomous actions an agent takes before needing a human "reviewer"	Indicates the level of trust and maturity of the agent's reasoning
Mean Time to Autonomy (MTTA)	The average time an artifact stays in the pipeline before reaching "golden" status without any human touch	Replaces "lead time" as the primary speed metric for autonomous flows
Tool Utilization Efficacy (TUE)	How accurately the agent selects and uses repo tools (e.g., calling the correct JFrog Xray API or GitHub Action)	Tracks the technical precision of the agentic layer

Fig. 8

not just a list of CVEs.

b. **Reachability Fix Ratio:** The percentage of vulnerabilities fixed that were actually "reachable" (exploitable) in the code. This filters out the "noise" of traditional scanners.

c. **Knowledge Graph Precision:** How often does the agent's reasoning align with the Enterprise Knowledge Graph (e.g., "The agent suggested Library X because the EKG marked it as the corporate standard").

d. **Policy Violation Pre-emption:** The number of policy violations caught and fixed by the pre-flight agent before the code was even committed to the main branch.

e. **Hallucination Rate in PRs**: The frequency with which an AI agent suggests code or configurations that do not exist or are functionally incorrect.

Value Stream Efficiency

a. **As AI generates** more code, pipelines can actually slow down due to "reviewer fatigue." These metrics track that friction.

b. **Reviewer Cognitive Load:** The ratio of "AI-generated lines" to "human-reviewed lines." (If this is too high, quality may drop).

c. **Autonomous Recovery Rate:** The % of production anomalies that were automatically rolled back or self-healed by the guardrail agent using the golden registry.

d. **Artifact Promotion Velocity:** The average time it takes for an artifact to move between registries (Sandbox Verified Golden).

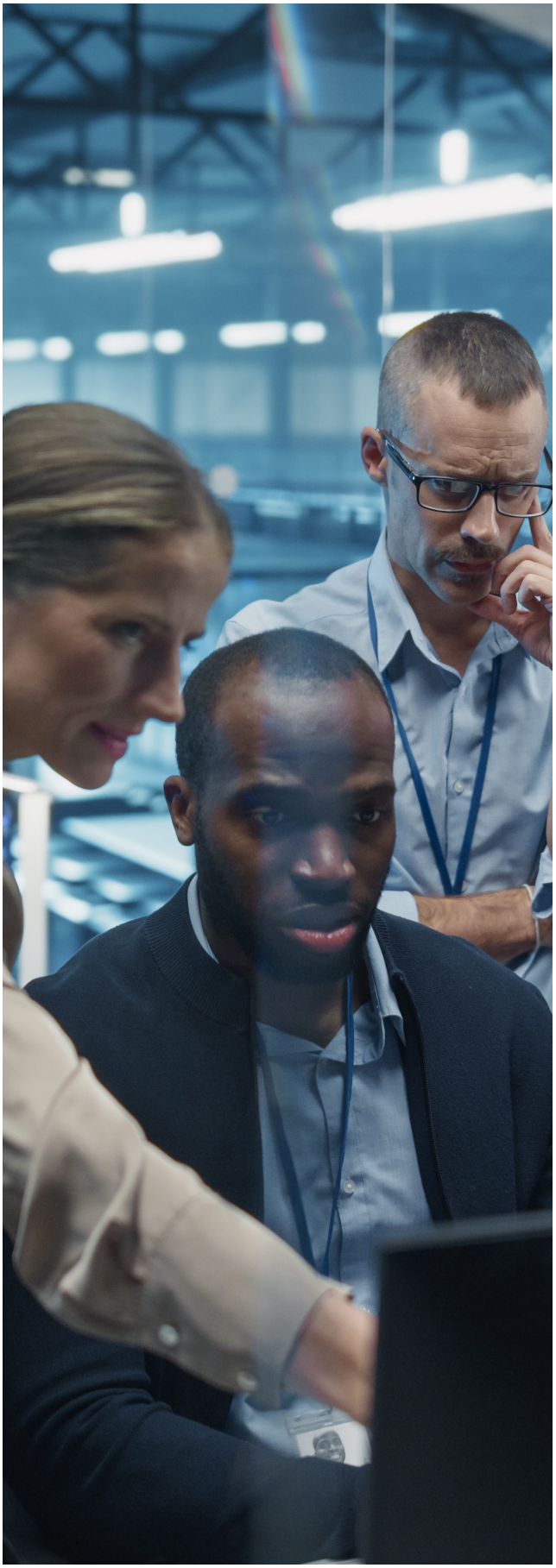
e. **The "Shadow" Build Cost:** The compute/token cost of running autonomous agents versus the developer hours saved.

The "CLASSic" Framework for Agentic Repositories

Organizations should adopt and implement CLASSic framework to evaluate their Smart repositories:

a. **Cost:** Token and compute spend per autonomous promotion.
Smart Repo Metric: What is the "Compute-to-Commit" ratio? If an agent fixes a bug but requires $50 worth of GPU tokens to reason through knowledge graph, is it worth the autonomous promotion?

b. **Latency:** Time taken for an agent to "reason" and execute a fix.
Smart Repo Metric: How long does the Pre-flight agent take to approve a diff? If the "wait time" for an agent's reasoning exceeds the "wait time" of a human reviewer, the value stream efficiency drops.



c. **Accuracy:** How often the agent's autonomous fix passes the final human audit.
Smart Repo Metric: "Pass#1" vs "Pass#8" rates. Does the agent's proposed fix to a maven pom.xml actually compile and pass the security scans on the first try?

d. **Security:** The integrity of the agent's access, security analysis and fix
Smart Repo Metric: Does the agent correctly identify if a vulnerability is exploitable? It also measures agent's own safety.

e. **Stability:** The failure rate of agent-promoted artifacts in production.
Smart Repo Metric: Does the agent provide the same high-quality fix for the same issues across different repos, or does its reasoning "drift" as code base grows?

Traditional Metric	Smart Artifact & Repo Evolution
Lead Time for Change	Mean Time to Autonomous Promotion
Change Failure Rate	Autonomous Recovery/Self-Healing Rate
Vulnerability Count	Reachability Fix Ratio
Build Success Rate	Pre-flight Correction Accuracy
MTTR (Recovery)	MTTD (Detection to Autonomous Rollback)

Fig. 9

The AI-Native Software Factory

The ultimate goal of the "smartness" shift is the realization of the AI-native software factory. This is a strategic approach to modernizing the entire software development lifecycle (SDLC) by integrating AI across all workflows.

- **Phase 1:** Assistants - Individual productivity is boosted through AI copilots for requirement generation, coding, testing, and deployment (1.2x efficiency).
- **Phase 2:** Agents: - The focus shifts to entire workflows, with AI agents performing automated checks and documentation updates (2x efficiency).
- **Phase 3:** Multi-agent - Multiple AI agents coordinate across the lifecycle, with actions triggered by ticket or pipeline states (3x efficiency).
- **Phase 4:** Autonomous Agents: - AI begins to orchestrate the entire delivery pipeline, managing deployments, compliance, and self-healing actions within governance guardrails (5x efficiency).

In this advanced stage, every artifact, whether documentation, code, or repository, functions as part of the factory itself. These smart, interconnected elements comprehend their roles, actively sustain their integrity, and autonomously collaborate to deliver ongoing value.

Central to this transformation is the emergence of the smart artifactory. Unlike traditional artifact repositories, a smart artifactory is equipped with AI-driven capabilities that enable real-time monitoring, validation, and optimization of software artifacts. It not only stores code, binaries, and documentation, but also intelligently manages their lifecycle—ensuring compliance, triggering automated corrections, and facilitating seamless collaboration between AI agents and human developers.

By connecting the AI-native software factory to a smart artifactory, organizations unlock continuous improvement, adaptive governance, and self-healing mechanisms. This integration allows every artifact to become an active participant in the delivery process, supporting autonomous workflows and driving higher efficiency and reliability throughout the SDLC.

Conclusion

In summary, the evolution toward AI-native software factories, smart artifacts, and autonomous repositories is revolutionizing software engineering. By seamlessly integrating intelligent agents, intent-driven workflows, and self-healing ecosystems, organizations are poised to achieve unprecedented speed, quality, and strategic alignment in software delivery.

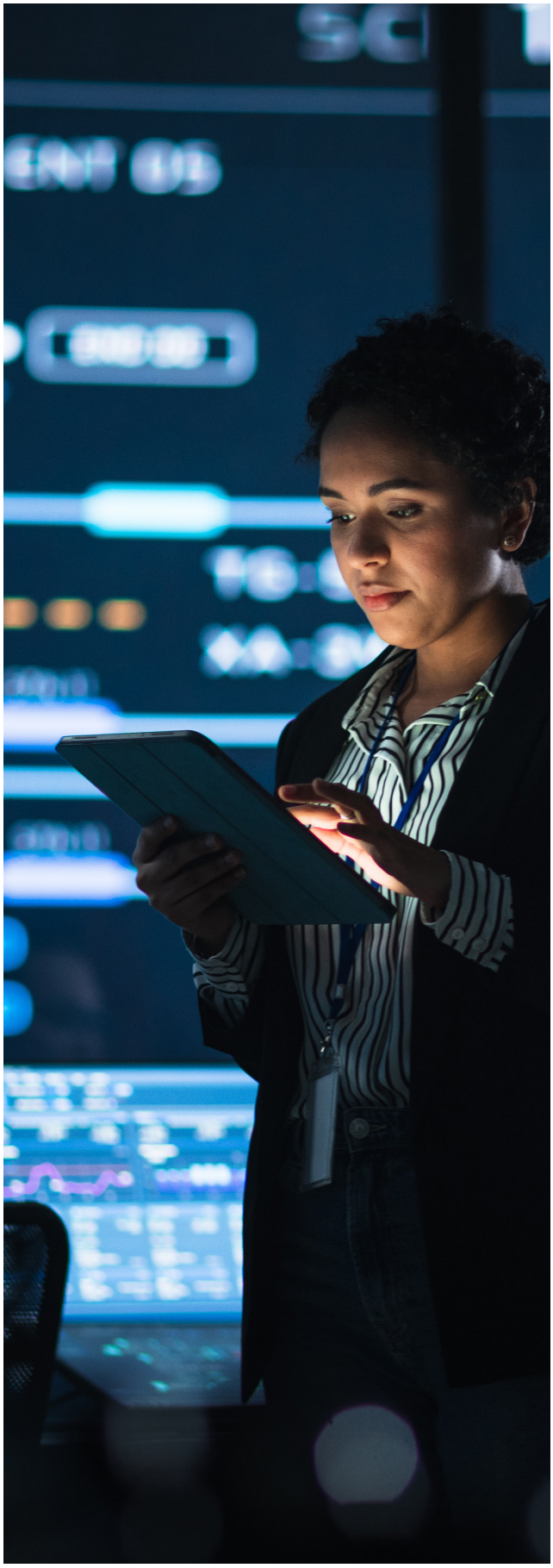
This shift redefines the role of human experts, empowering them to focus on high-level strategy and innovation, while AI agents manage execution, optimization, and routine tasks. As digital artifacts and repositories become intelligent collaborators, software development transforms into a dynamic, adaptive process that continuously evolves to meet business goals. Ultimately, this paradigm marks the dawn of a new era—where automated processes and AI-driven collaboration accelerate innovation, uphold governance, and ensure that human creativity remains at the heart of technological advancement.

About the Author

Naresh Choudhary
SVP – Head – Enterprise Productivity and Engineering Excellence

Co- Author

Srinivasa Sujit Rao
Industry Principal



References

1. Smart Construction Objects | Journal of Computing in Civil Engineering - ASCE Library
2. (PDF) Agent-oriented smart objects development - ResearchGate
3. csiro/stdm: Self Thinking Data Manifest - GitHub
4. Sensor Networks or Smart Artifacts? An Exploration of Organizational Issues of an Industrial Health and Safety Monitoring System – ResearchGate
5. (PDF) Designing smart artifacts for smart environments – ResearchGate
6. (PDF) Conception of Smartness: A Design Research on User Experience of Smart Artifacts
7. Development capabilities for smart products | Request PDF – ResearchGate
8. Jotai MCP Server: A Deep Dive for AI Engineers - Skywork.ai
9. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering - arXiv
10. GitHub Copilot vs Cursor vs Windsurf AI Comparison - Digital Marketing Agency
11. GitHub Copilot vs Copilot Agent: AI Coding Tools Compared - DZone
12. How GitHub Next took Copilot Workspace from concept to code · community · Discussion #142971
13. GitHub Copilot Workspace: Welcome to the Copilot-native developer environment
14. Managing Technical Debt with AI Solutions in 2025 - Zencoder
15. How to Use AI to Reduce Technical Debt - Semaphore CI
16. The Role of AI in Reducing Tech Debt | IT IDOL Technologies
17. What is Retrieval-Augmented Generation (RAG)? - Google Cloud
18. Retrieval Augmented Generation (RAG) and Large Language Models (LLMs) for Enterprise Knowledge Management and Document Automation: A Systematic Literature Review - Preprints.org
19. (PDF) Implementation of Retrieval-Augmented Generation (RAG) and Large Language Models (LLM) for a Document and Tabular-Based Chatbot System - ResearchGate
20. Implementation of RAG Based Question-Answering Application - IEEE Xplore
21. Documentation in Continuous Software ... - Theo Theunissen
22. Theunissen Et Al. - 2022 - Approaches For Documentation in Continuous Softwar - Scribd
23. Mintlify - The Intelligent Documentation Platform
24. What's new in Eficode ROOT: October 2025 - Eficode.com
25. Approaches for Documentation in Continuous Software Development
26. Handling Complexity in Modern Software Engineering: Editorial Introduction to Issue 32 of CSIMQ
27. Hexaview at Dreamforce 2025: Driving AI-First Transformation for
28. The Ultimate Guide to AI-Powered Product Documentation | Narratize
29. Process Documentation Specialist AI Agents | Relevance AI
30. Validate AI Code: Human-in-Loop PR Testing with TestQuality
31. Human-In-The-Loop: What, How and Why | Devoteam
32. AI code review implementation and best practices - Graphite
33. Human-in-the-Loop Validation in AI Code Security - HackerOne
34. AI Agent Evaluation: Metrics, Strategies, and Best Practices
35. AI Evaluation Metrics 2026: Tested by Conversation Experts - Master of Code
36. Evaluating Agentic AI Systems: 10 Metrics for Smarter Autonomous AI Agents | by Dhiraj K
37. AI Agent Metrics- A Deep Dive - Galileo AI
38. AI and the Rise of Self-Healing IT Systems - IT Resources - Tampa, Florida
39. A Beginner's Guide to Implementing Self-Healing AI Systems - SuperAGI
40. Self-Thinking Data Manifest
41. JFrog Fly
42. Beyond Accuracy: A Multi-Dimensional Framework for Evaluating Enterprise Agentic AI Systems
43. AI Agents Benchmarking: A Framework for Real-World Enterprise Success

Infosys Topaz is an AI-first set of services, solutions and platforms using generative AI technologies. It amplifies the potential of humans, enterprises, and communities to create value. With 12,000+ AI assets, 150+ pre-trained AI models, 10+ AI platforms steered by AI-first specialists and data strategists, and a 'responsible by design' approach, Infosys Topaz helps enterprises accelerate growth, unlock efficiencies at scale and build connected ecosystems. Connect with us at infosystopaz@infosys.com.

For more information, contact askus@infosys.com

Infosys
Navigate your next