On Task Automation Economics

An Insight into Task Automation Factories as the Production Mechanism

Mohamed A. Fouad
maf@axn.run

Abstract

Agentic AI can perform knowledge-based tasks, but data engineering teams need more than successful individual executions. This paper argues that the task automation factory is the production mechanism that turns recurring agent execution into governed software capacity. In practice, generative AI has made automation easy to attempt while leaving many prompts, scripts, workflows and agent runs as one-off artifacts. Agentic AI execution is therefore not automation. Automation begins when recurring work becomes verified software capacity. Task automation economics names the software engineering economics of this shift under generative-AI cost structures. It asks when recurring work should stop being purchased as isolated agent runs and should instead be released as governed automation assets. Data engineering grounds the insight because ingestion, cleaning, schema mapping, transformation, validation, orchestration, lineage and backfills recur across teams. The economic unit is not the agent run. It is the verified automation asset: a released object that can be specified, verified, audited, reused and replaced. The paper introduces task automation economics, defines the verified automation asset and explains how axnrun-backed task automation factories turn recurring data engineering demand into governed software capacity.

Keywords: task automation economics; task automation factories; verified automation assets; governed software capacity; agentic AI; software engineering economics; data engineering workflows; reusable automation assets; axnrun; software verification

1 The Task Automation Problem

Agentic AI execution is not automation. Automation begins when recurring work becomes verified software capacity. The task automation problem is not the absence of agent capability. It is the absence of a production mechanism that turns recurring agent execution into governed software capacity. A run proves feasibility. It does not create capacity.

The problem is a category error: it treats an event as an asset. An agent run is an event. It can answer a request, produce a document or compare records. An automation asset is different. It has a specification and evidence requirements. It also has review criteria, a release state and a replacement path. Task automation economics begins at this boundary.

Data engineering teams repeat knowledge-based tasks every day. They ingest files, map schemas, clean records, transform tables, validate data quality, orchestrate

jobs and reconcile failures. These tasks require interpretation, system context and operational judgment. They also require evidence preservation because sources, transformations, checks and lineage must remain inspectable. An agent can perform one instance of such work. That successful execution matters because it shows feasibility. It is still not yet automation.

This run-level capability is well represented by work on reasoning-and-acting agents [7]. Recent work on agentic software engineering extends this view toward workflows, harnesses and lifecycle control [15]. These developments make the task automation problem more urgent. The better agents become at execution, the easier it becomes to mistake performance for capacity.

The distinction matters because repeated execution has weak economics. Each run purchases output again. It does not by itself accumulate reusable capacity. A data engineering team therefore faces a narrow question: when should recurring workflow demand stop being bought as separate agent runs and become governed software capacity?

This paper advances one argument: agentic AI becomes economically useful for recurring data engineering work when task execution becomes task automation. The economic unit is not the agent, prompt or run. It is the verified automation asset. The asset can be specified and verified. It can be audited, reused and replaced. We use axnrun and data engineering workflows as grounding because they expose the missing runtime link between one-off execution and reusable automation.

The contribution is threefold. First, we introduce **task automation economics** as a software engineering economics problem under generative-AI cost structures. Second, we define the **verified automation asset** as the economic object that distinguishes task execution from task automation. Third, we propose the **task automation factory** as the production mechanism that turns recurring demand into governed software capacity.

The novelty lies in treating automation as capability formation rather than task performance. Capability formation means that repeated work leaves behind reusable organizational capacity. For data engineering teams, that capacity matters because automation must improve repeated work without hiding sources, schemas, transformations, checks or lineage. We focus on the economic and engineering boundary where repeated agent runs become verified automation assets.

2 Task Automation Economics

Definition 1. *Knowledge-based task.* A **knowledge-based task** is repeated work whose correctness depends on rules and records. It also depends on context, evidence and exceptions. It is automation-relevant because it recurs. It is governance-relevant because the team must justify how the result was produced.

Definition 2. *Task automation economics.* **Task automation economics** is the software engineering economics of turning recurring knowledge work into governed automation assets under generative-AI cost structures. It asks when recurring work should stop being bought as separate agent runs and become governed software capacity. Its unit of value is not an agent run. It is a verified asset that can be released and reused.

We extend software engineering economics from software products to recurring agent-mediated task assets [1]. Prior work framed software engineering agent economics around markets and economic network simulations [14]. This paper moves the economic unit from the software engineering agent to the verified automation asset.

Definition 3. *Verified automation asset.* A **verified automation asset** is a released automation object that converts recurring task execution into reusable software capacity. Its task specification and evidence requirements are explicit. Its review criteria and release state are captured. Its replacement path is defined.

Governed software capacity means a released automation asset whose specification, evidence, release state and replacement path can be reviewed. For example, an agent may clean one source table once. That is execution. The task becomes automation only when the transformation logic, evidence requirements, review criteria and release state are captured as a reusable asset.

The economic question is whether engineering effort should be invested so repeated execution becomes reusable organizational capacity. Repeated execution spends value. Verified assets accumulate value. The economics become favorable when reuse value exceeds lifecycle cost. Value comes from accumulated capability; cost comes from specification, engineering, verification, audit records and replacement. This lifecycle burden is familiar from technical debt in machine learning systems [2]. A one-off agent execution can still be useful. It is only the first signal that a task may be worth turning into software capacity.

The decision is a software engineering decision. The team asks whether the task is stable enough to specify. It asks whether rules and evidence can be represented. It asks whether a pipeline can be replayed and inspected. It also asks whether the resulting asset can survive changes in sources, schemas, dependencies and downstream contracts. If the answer is no, the task may remain assisted execution. If the answer is yes, it becomes a candidate for task automation.

The distinction follows a software engineering logic: requirements are made explicit, verification criteria are applied, and provenance records are preserved [1, 3, 4, 5].

We use a compact conceptual model, but the argument does not depend on formalization.

Conceptual model. A candidate automation asset has seven reviewable parts: a specification, governance template, pipeline, criteria, evidence, release snapshot and replacement rule. The asset is accepted only when conformance, evidence sufficiency, auditability and release integrity all pass. The model is shorthand for reviewability rather than a formal claim.

The release snapshot and integrity requirement connect the asset to software supply-chain concerns about artifact trust [6]. An automation asset is not trusted because one run succeeded. It is trusted because its specification and governance can be reviewed. Its pipeline, evidence and release state must also be reviewable. Its replacement path must be clear. A pipeline output that fails these checks remains execution. It is not automation.

3 Task Automation Factory Mechanism

A task automation factory is the mechanism that makes task automation economics operational.

Definition 4. *Task automation factory.* A **task automation factory** is a software engineering production mechanism for recurring task automation. It turns repeated demand into governed automation assets. It connects specification and governance; engineering and verification; audit records and release. It also supports reuse and replacement over time.

The factory selects repeated demand and specifies the work. It then engineers tools and pipelines. The accepted asset is verified, released and maintained. In this paper, axnrun instantiates the runtime part of this mechanism: users submit tasks, axnrun breaks each task into actions, runs existing tools and records what is missing. When the same missing tool appears again, the factory can promote that gap into a verified reusable asset.

We place this mechanism at the center of our argument. Without specification, automation guesses. Without engineering, it remains repeated execution. Without verification, it cannot be trusted. Without release, it cannot be reused. Without replacement, it decays.

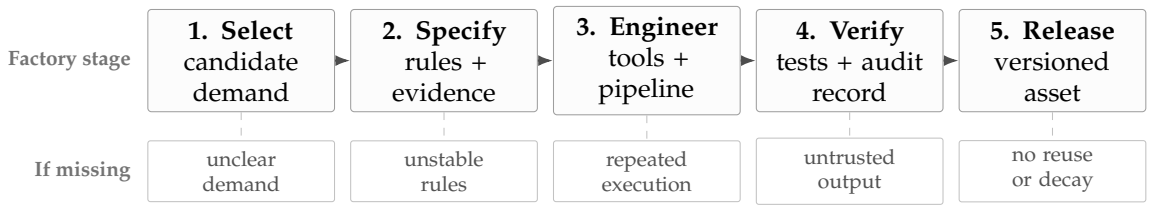

Figure 1. Task automation factory mechanism. Repeated execution becomes automation only when it is specified and engineered. It must also be verified, released and kept current.

A task automation factory is dark only as a software engineering production metaphor. It can automate engineering work, but it must not make data, pipeline or

workflow accountability invisible. Rules, evidence, audit records and replacement paths must remain visible.

Recent industry discourse uses *dark factory* to describe software workflows where agents operate inside explicit specifications, tests and release constraints [21, 22, 23, 24]. Broader work on tool use, autonomous agents, agentic software engineering and software-agent evaluation provides background for the run-level capabilities that factories must govern [8, 9, 16, 17, 18, 19, 20].

We use a short bottleneck lens to clarify the factory mechanism. Factory throughput is not the number of agent executions. It is the rate at which repeated demand becomes verified automation assets. The bottleneck may be specification, evidence or review. It may also be release discipline. We use this lens to support the argument, but not as a second theory [12, 13].

4 Data Engineering as Starting Domain

We start with data engineering workflows because repeated work is everywhere. Ingestion scripts, schema mappings, cleaning rules, transformation jobs, validation checks, orchestration DAGs and backfill procedures often begin as one-off scripts or agent-assisted runs. They are useful, but they decay quickly when sources change, schemas drift or the next pipeline needs the same capability in a slightly different form.

axnrun addresses this gap as an open-source runtime for reusable automation. Users submit tasks. axnrun breaks each task into actions, runs existing tools and records what is missing. The important economic signal is not only that a task was completed. It is that the same missing tool appears again. Repetition identifies candidate demand for a reusable automation asset.

The hosted axn.run layer and shared toolbox provide implementation context for axnrun, but the commercial layer is secondary to the paper's mechanism. The central claim is that recurring data engineering demand becomes valuable when it can be converted into verified reusable assets rather than repeated one-off executions.

Data engineering artifact	Factory reading	Why it matters
Ingestion script	Evidence of repeated source demand	Shows what users already try to automate
Schema mapping	Candidate reusable contract	Stabilizes changing source structure
Cleaning rule	Candidate reusable transformation	Turns repeated preparation into governed capacity
Transformation job	Executable automation asset	Connects data logic, checks and release state
Validation check	Review and evidence layer	Supports auditability and data quality
Missing-tool record	Demand signal for asset creation	Shows what the factory should build next

Table 1. How data engineering workflow artifacts map to task automation factory functions.

5 Evaluation Criteria and Limits

A task automation asset qualifies as durable capacity only when reviewers can answer six questions. We use this checklist as the practical test for whether the unit of analysis has shifted from run to asset.

Criterion	Reviewer question
Explicit task	Is the recurring work described clearly enough to engineer?
Defined evidence	Is acceptance tied to rules and records?
Replayable pipeline	Can the mechanism be rerun and inspected?
Reconstructible acceptance	Can a reviewer explain why the asset passed?
Known release	Is reuse tied to a reviewed release state?
Replacement path	Is there a trigger for repair or retirement?

Table 2. Evaluation checklist for distinguishing task execution from task automation.

The reviewed-release criterion is also an artifact-trust requirement, not only a deployment label [6]. The checklist avoids a common error: measuring only pipeline or agent performance. A high-scoring output is insufficient if the team cannot link it to a rule and evidence record. It must also link to a verification criterion, release snapshot and replacement path. Conversely a modest deterministic tool may be valuable if it replaces repeated execution with governed software capacity.

The checklist is narrower than general responsible-AI governance. It tests whether a recurring task has become a reviewable asset, not whether every AI risk has been resolved [10, 11]. The claim is deliberately limited. Not every data engineering task should be automated. The argument applies to recurring workflow demand where evidence and review matter. The axnrun and data engineering grounding is also limited: it illustrates a plausible path from repeated workflow demand to task automation factories, but it is not a complete longitudinal proof. Future work should

measure conversion rate from repeated runs to released assets, time-to-asset, reuse count, audit reconstruction success and replacement events.

6 Conclusion

This paper identifies the task automation problem as a category error in agentic AI adoption: treating a successful execution event as if it were an automation asset. The issue is not that agents cannot perform useful work. The issue is that successful runs do not, by themselves, leave behind reviewable, reusable and replaceable organizational capacity. A run proves feasibility. It does not create capacity.

The central insight is that automation begins only when recurring work is converted into governed software capacity. Task automation economics names the decision problem: when should repeated agent execution stop being purchased run by run and instead be engineered as a verified automation asset? The verified automation asset is the proper unit of analysis because it carries the properties data engineering teams need: explicit specification, evidence requirements, review criteria, release state, auditability, reuse and replacement.

Task automation factories provide the production mechanism that makes this conversion practical. The justification is economic and operational. Economically, repeated execution repurchases output, while released assets accumulate capability across reuse. Operationally, data engineering automation must remain reproducible, inspectable, governable and controlled over time. Factories make these requirements concrete by connecting demand selection, specification, engineering, verification, release and replacement.

Data engineering grounds the insight without overclaiming. It shows that scripts, DAGs and agent runs can make recurring work visible, but visibility is only the starting condition. axnrun adds the runtime signal by recording actions, existing tools and missing tools; the factory adds the production discipline by promoting repeated gaps into verified reusable assets. The conclusion is therefore simple: agentic execution demonstrates what can be done; task automation factories determine what can become durable governed capacity.

References

[1] Boehm, Barry W. *Software Engineering Economics*. Prentice Hall, 1981.

[2] Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M.; Crespo, J.-F.; and Dennison, D. "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems 28*; 2015. Available at: https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html.

[3] International Organization for Standardization with International Electrotechnical Commission and IEEE. *ISO/IEC/IEEE 29148:2018: Systems and software engineering – Life cycle processes – Requirements engineering*. Geneva; 2018. Available at: https://www.iso.org/standard/72089.html.

[4] IEEE Standards Association. *IEEE 1012-2024: IEEE Standard for System, Software, and Hardware Verification and Validation*. New York; 2024. Available at: https://standards.ieee.org/ieee/1012/7324/.

[5] World Wide Web Consortium. *PROV-DM: The PROV Data Model*. W3C Recommendation, 30 April 2013. Available at: https://www.w3.org/TR/prov-dm/.

[6] Open Source Security Foundation. *Supply-chain Levels for Software Artifacts (SLSA)*. Available at: https://slsa.dev.

[7] Yao, Shunyu; Zhao, Jeffrey; Yu, Dian; Du, Nan; Shafran, Izhak; Narasimhan, Karthik; and Cao, Yuan. "ReAct: Synergizing Reasoning and Acting in Language Models." *International Conference on Learning Representations*; 2023. Available at: https://arxiv.org/abs/2210.03629.

[8] Schick, Timo; Dwivedi-Yu, Jane; Dessì, Roberto; Raileanu, Roberta; Lomeli, Maria; Zettlemoyer, Luke; Cancedda, Nicola; and Scialom, Thomas. "Toolformer: Language Models Can Teach Themselves to Use Tools." *Advances in Neural Information Processing Systems 36*; 2023. Available at: https://proceedings.neurips.cc/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

[9] Wang, Lei; Ma, Chen; Feng, Xueyang; Zhang, Zeyu; Yang, Hao; Zhang, Jingsen; Chen, Zhiyuan; Tang, Jiakai; Chen, Xu; Lin, Yankai; Zhao, Wayne Xin; Wei, Zhewei; and Wen, Ji-Rong. "A Survey on Large Language Model Based Autonomous Agents." *Frontiers of Computer Science*; 2024. Available at: https://doi.org/10.1007/s11704-024-40231-1.

[10] International Organization for Standardization with International Electrotechnical Commission. *ISO/IEC 42001:2023: Information technology – Artificial intelligence – Management system*. Geneva; 2023. Available at: https://www.iso.org/standard/81230.html.

[11] Raji, Inioluwa Deborah; Smart, Andrew; White, Rebecca N.; Mitchell, Margaret; Gebru, Timnit; Hutchinson, Ben; Smith-Loud, Jamila; Theron, Daniel; and Barnes, Parker. "Closing the AI Accountability Gap: Defining an End-to-End Framework for Internal Algorithmic Auditing." *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*; 2020. Available at: https://doi.org/10.1145/3351095.3372873.

[12] Goldratt Eliyahu M. and Cox Jeff. *The Goal: A Process of Ongoing Improvement*. North River Press; 1984.

[13] Forsgren Nicole; Humble Jez; and Kim Gene. *Accelerate: The Science of Lean Software and DevOps*. IT Revolution Press; 2018.

[14] Ibrahim, Mohamed Ahmed Fouad Aly Mohamed. *Software Engineering Agent Economics: A Blockchain Software Development Kit for Economic Network Simulations*. Master's dissertation, 2025. DOI: https://doi.org/10.14393/ufu.di.2025.712.

[15] Ning, Xuying et al. "Code as Agent Harness: Toward Executable, Verifiable and Stateful Agent Systems." arXiv preprint arXiv:2605.18747, 2026. Available at: https://arxiv.org/abs/2605.18747.

[16] Hassan, Ahmed E.; Li, Hao; Lin, Dayi; Adams, Bram; Chen, Tse-Hsun; Kashiwa, Yutaro; and Qiu, Dong. "Agentic Software Engineering: Foundational Pillars and a Research Roadmap." arXiv preprint arXiv:2509.06216, 2025. Available at: https://arxiv.org/abs/2509.06216.

[17] Roychoudhury, Abhik. "Agentic AI for Software: Thoughts from Software Engineering Community." arXiv preprint arXiv:2508.17343, 2025. Available at: https://arxiv.org/abs/2508.17343.

[18] Bhati, Happy. "Agentic AI in the Software Development Lifecycle: Architecture, Empirical Evidence, and the Reshaping of Software Engineering." arXiv preprint arXiv:2604.26275, 2026. Available at: https://arxiv.org/abs/2604.26275.

[19] Xu, Bin. "AI Agent Systems: Architectures, Applications, and Evaluation." arXiv preprint arXiv:2601.01743, 2026. Available at: https://arxiv.org/abs/2601.01743.

[20] Badertdinov, Ibragim et al. "SWE-rebench: An Automated Pipeline for Task Collection and Decontaminated Evaluation of Software Engineering Agents." arXiv preprint arXiv:2505.20411, 2025. Available at: https://arxiv.org/abs/2505.20411.

[21] Voroshkov, Andrey. "From Figma to Production Code: Building a Dark Factory with AI Agents." EPAM AI/Run Blog, Apr. 22, 2026. Available at: https://www.epam.com/insights/ai/blogs/building-a-dark-factory-with-ai-agents.

[22] Dark Factory. *Dark Factory: Lights-Out Software Development*. Project website, n.d. Available at: https://godarkfactory.com/.

[23] Shapiro, Dan. "The Five Levels: from Spicy Autocomplete to the Dark Factory." Dan Shapiro's Blog, Jan. 23, 2026. Available at: https://www.danshapiro.com/blog/2026/01/the-five-levels-from-spicy-autocomplete-to-the-software-factory/.

[24] Tyagi, Saumya. "The Dark Factory Pattern: Moving From AI-Assisted to Fully Autonomous Coding." HackerNoon, Feb. 19, 2026. Available at: https://hackernoon.com/the-dark-factory-pattern-moving-from-ai-assisted-to-fully-autonomous-coding.