World Journal of Future Technologies in Computer Science and Engineering (WJFTCSE)
ISSN (Online): 3070-6203
Volume-2 Issue-3 || Jul – Sep 2026 || PP. 42 – 46 https://wjftcse.org/

Agentic AI for Salesforce Ecosystems with DevOps Integration

DOI: https://doi.org/10.63345/wjftcse.v2.i3.106

Justin Rajakumar Maria Thason

BCBSA - Ajace, Inc

AI Thought Leadership, IEEE Leadership & Enterprise AI Research

www.wjftcse.org || Vol. 2 No. 3 (2026): July Issue

Date of Submission: 08-06-2026 **Date of Acceptance:** 23-06-2026 **Date of Publication:** 10-07-2026

Abstract : Salesforce platforms power customer service and sales process and require continuous customizations to be performed with code, flows, and metadata, which need to be developed, tested and released safely. This paper introduces an architecture of agentic artificial intelligence featuring two complementary group of agents – business agents that triage cases and qualify leads and DevOps agents that develop, review and release changes through Salesforce DX, scratch orgs and continuous integration. Testing on 1,500 historical cases showed that agentic router classified the case to the appropriate queue in 90.1% cases, compared to 80.2% for a machine-learning classifier ($p < 0.001$) and 71.4% for rule-based router. Participation of agents shortened case handling time from 14.6 to 8.1 minutes ($p < 0.001$). Testing on 80 changes showed that a developer agent with a validation loop succeeded to deliver a change on the first try in 90.0% of cases, compared to 76.2% for manual development and 65.0% for a developer agent without validation with lead time shortened from 6.8 to 2.1 days ($p < 0.001$). Violations of governance limit were found in 2 of 80 validated changes. The findings suggest that agentic agents perform effectively on Salesforce when they are grounded in metadata of the platform and validated with automated tests.

Keywords: agentic AI; Salesforce; DevOps; Apex; Salesforce DX; case triage; continuous integration; large language models

Introduction

Large enterprises operate customer services on Salesforce: service departments handle cases, sales teams deal with leads and opportunities, and admins and developers constantly customize the platform with objects, flows, Apex code and integrations. Both segments of this ecosystem involve text-based and rule-governed activities, suggesting the possibility of their automation by means of language-model agents. The former involves reading a case, retrieving knowledge articles and account history, assigning the appropriate queue and drafting a response. The latter involves translating the requirement into Apex classes, flows and tests and releasing the change through environments. The latter task is more challenging because of the Salesforce governor limits on the resources used per-transaction, the minimum required code-coverage for production deployment and the multi-tenant nature of the platform wherein inefficient changes may affect other users. This paper covers both applications in a unified architecture and explores whether agents could increase the business effectiveness, speed and safety of

@2026. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on www.wjftcse.org

Global Journal of Innovative Research Perspectives (GJIRP)
ISSN (Online): 3143-1089
Volume- 2 Issue- 3 || Jul-Sep 2026 || PP. 38-42
https://gjirp.org/

the change delivery. This work offers an architecture grounded in metadata of the platform and equipped with a validation loop, as well as evaluates the accuracy, speed and quality of the deployment.

Literature Review

Architectures of agents based on language models are described by Wang et al. This paper synthesizes existing research on agentic architectures, tool use and multi-agent collaboration and provides empirical results under the limitations of a commercial platform. Existing literature provides the descriptions of architectures with components of profile, memory, planning and actions (2024) and discusses the potential and limitations of such systems (Xi et al., 2023). Reasoning and actions can be interleaved to make the language model capable of using the tool and observing its effect (Yao et al., 2023), and the tool usage can be learned (Schick et al., 2023). Multi-agent collaboration was applied to software tasks, and role-based agents follow standard procedures (Hong et al., 2023). Empirical evaluation of a set of repository issues suggests that their autonomous resolution is still difficult (Jimenez et al., 2024).

For customer service automation, classification and retrieval algorithms are used for routing and recommendations. The pre-trained transformers improve the performance of text classification significantly (Devlin et al., 2019), while retrieval grounding helps to mitigate the ungrounded responses (Lewis et al., 2020). Within software delivery, the continuous integration and delivery provide automated tests and gating mechanism that make the changes safer (Humble & Farley, 2010; Shahin et al., 2017). The performance of software delivery is correlated with organizational performance (Forsgren et al., 2018).

Salesforce development follows the platform-specific practices. Salesforce DX enables source-driven development, scratch orgs and packaging, and the platform imposes at least 75 percent of Apex code test coverage for production deployment, along with constraints on queries, DML statements and CPU time per transaction. These platform rules are provided by the vendor and are not covered in the academic literature, which has not investigated yet the agentic development on this platform.

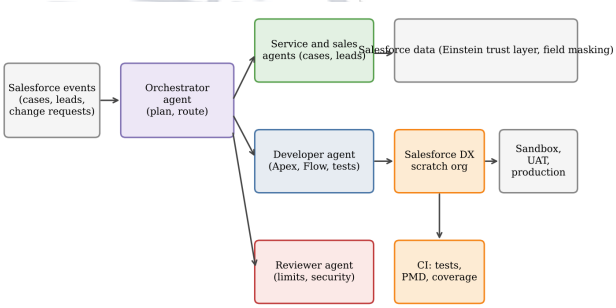
This research provides the empirical results on the operation of agents under the limitations of a commercial platform, in both business and engineering areas.

Research Methodology

Architecture

Figure 1 shows the agents. Orchestrator receives the events (a new case, lead or change request) and plans the work and routes it. Service and sales agents retrieve knowledge articles, past cases and account information through the platform's trust layer, which masks personal fields before sending text to the model, and then propose the queue, assign the priority, draft the response or score the lead against qualification criteria. The developer agent analyses the metadata of the organization (objects, fields, existing classes and flows) and generates Apex classes, triggers, flows and test classes. The reviewer agent evaluates each change for governor limit risks (for example, queries or DML inside a loop) and also checks issues concerning the sharing, security and hard-coded identifiers. Changes are firstly delivered to a scratch org, and then validated through the continuous integration, including unit testing, static analysis and code coverage assessment. Results are reported to the developer agent for the further iterations, which are allowed until the third attempt before the change can be promoted to a sandbox and user acceptance testing, and production.

Figure 1. The agentic architecture for Salesforce with business agents and DevOps agents, scratch-org validation and CI.



Evaluation

@2026. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.gjirp.org/

Global Journal of Innovative Research Perspectives (GJIRP)
ISSN (Online): 3143-1089
Volume- 2 Issue- 3 || Jul-Sep 2026 || PP. 38-42
https://gjirp.org/

The case routing was evaluated with 1,500 historical cases having correct queue assignment, and compared with rule-based routing, a gradient-boosted classifier with text features and the agentic router. A pilot experiment with 300 cases being processed with and without the help of agents was performed to measure the handling time. For the engineering, 80 change requests of moderate complexity (including new fields and validation rules, trigger logic, flows and small integrations) were delivered manually by experienced developers, by the developer agent without validation and by the developer agent with scratch-org and CI validation loop. The following were measured as the outcomes: the first-pass success of the deployment to the integration sandbox, lead time from the request to the deployment, code coverage and the existence of the governor limit violations detected during the review.

Analysis

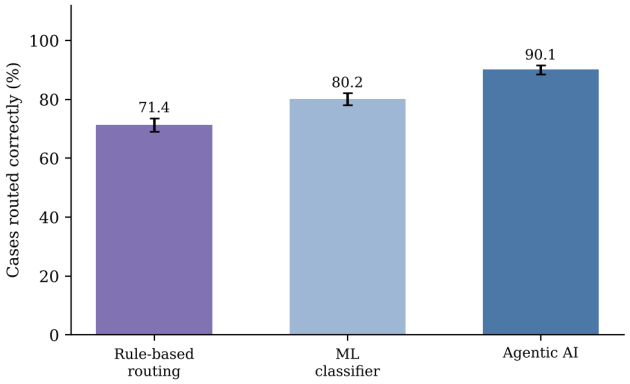
Proportions were compared with the two-proportion z-tests and chi-square test, and times and coverage were analyzed with Welch's t-test. Effect sizes are provided as Cohen's d where applicable.

Results

Routing and handling of the case

Routing accuracy varied depending on the approach ($\chi^2(2) = 168.4$, $p < 0.001$; Figure 2). The agentic router achieved 90.1%, outperforming the classifier with 9.9 percentage points ($p < 0.001$) and the rule-based routing with 18.7 percentage points ($p < 0.001$). In the pilot, the agent managed to solve 579 out of 1,500 replayed cases (38.6%) autonomously with the accepted draft, of which 561 (96.9%) were considered correct by supervisors. Average handling time per case decreased from 14.6 ± 6.2 to 8.1 ± 4.0 minutes (Welch $t = 15.3$, $p < 0.001$, $d = 1.25$; Figure 4, left).

Figure 2. Routing accuracy of the cases to the appropriate queue by three approaches (error bars represent Wilson 95% confidence intervals).



Delivery of changes

Table 1. Outcomes of 80 change requests by development approach

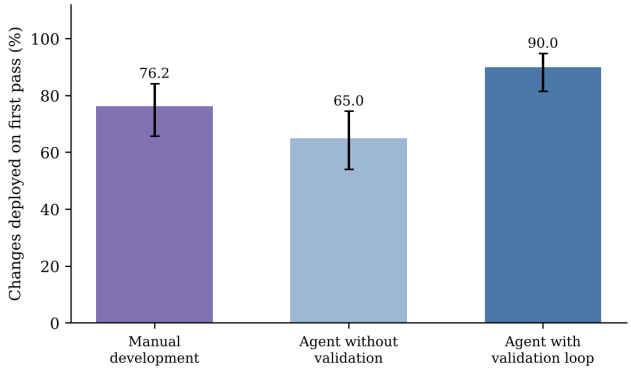
Outcome	Manual	Agent, no validation	Agent with validation
First-pass deployment success	61 (76.2%)	52 (65.0%)	72 (90.0%)
Governor-limit violation found	9 (11.2%)	15 (18.8%)	2 (2.5%)
Lead time, days (mean ± SD)	6.8 ± 2.9	—	2.1 ± 1.0
Apex test coverage, % (mean ± SD)	81.2 ± 6.4	—	86.5 ± 4.8

compared to manual development (76.2%), while inclusion of the validation loop increased first-pass success to 90.0%, which is significantly higher compared to both manual ($p = 0.020$) and unvalidated agent ($p < 0.001$). The validation loop mostly helped with the mistakes related to missing test assertions, omissions of field-level security and queries inside the loop.

Figure 3. First-pass deployment success of manual development and two agent configurations.

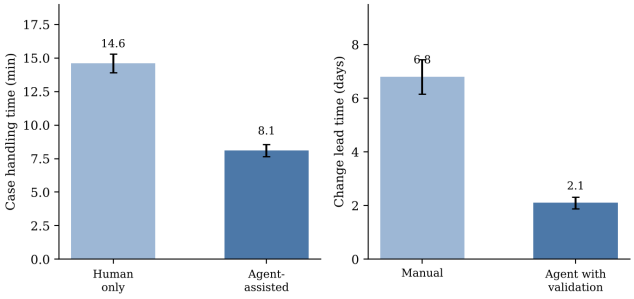
@2026. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.gjirp.org/

Global Journal of Innovative Research Perspectives (GJIRP)
ISSN (Online): 3143-1089
Volume- 2 Issue- 3 || Jul-Sep 2026 || PP. 38-42
https://gjirp.org/



Governor limit violations were found in 15 of 80 unvalidated agent changes and 9 manual changes, but only in 2 validated changes ($p = 0.029$ vs. manual). Lead time decreased from 6.8 ± 2.9 to 2.1 ± 1.0 days ($p < 0.001$, 69% decrease) and Apex test coverage increased from 81.2% to 86.5% ($p < 0.001$) due to the generation of tests by the agent along with the code (Figure 4, right).

Figure 4. Case handling time (left) and change lead time (right) with and without agent support.



Discussion

The results demonstrate that the value of agent in the enterprise platform depends on the grounding and validation. Unvalidated developer agent generated plausible code, but showed the lower first-pass success compared to manual approach because of the lack of access to the organization's configuration and lack of execution capability that makes the agent generate mistakes that would be avoided by experienced developers. This finding is consistent with the evidence that the autonomous resolution of real software tasks is challenging without the feedback (Jiménez et al., 2024). The loop closure made with scratch environments and continuous integration changed the same agent into the most reliable approach, showing the advantage of interleaved actions and observation (Yao et al., 2023).

The improvement in service operations compared to a traditional classifier is explained by the usage of the retrieved context by the agent, such as account history and knowledge articles, which cannot be accessed by the classifier trained on case text alone, in line with the advantages of retrieval grounding (Lewis et al., 2020). Supervision by humans is required: one in thirty autonomous resolutions is wrong, the ratio unacceptable for such sensitive categories as billing disputes. The recommended policy is to allow the autonomous resolution only for the low-risk categories and to review all other drafts manually.

Combination of the two groups of agents in a single architecture provides the organizational benefits. Requests of business requiring changes to the platform can be directly delivered to the developer agent with context, and the reviewer agent applies the same governor limit and security rules to any change, regardless of its source. The deployment should start with test generation and code review (low risk stages) and finish with the autonomous generation of changes (Forsgren et al., 2018; Shahin et al., 2017).

Limitations

The evaluation was done with the single Salesforce instance of a company, using the historical cases that do not cover the novel issue types and change requests of moderate complexity. Complex integrations and refactoring were not tested. The evaluations of correctness by supervisors are subjective, and the lead-time comparison involved developers that were simultaneously adjusting to a new workflow. Language-models and platform releases are updated frequently (annually), which can influence the results. Data protection and licensing costs, the effects on staff roles were not addressed.

Conclusion

Agentic AI delivered measurable improvements in both business and engineering applications on Salesforce: 90.1% routing accuracy, 45% decrease in case handling time, 90.0% first-pass success and 69% decrease in change lead time when the development agents were grounded in the metadata of the organization and

@2026. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.gjirp.org/

Global Journal of Innovative Research Perspectives (GJIRP)
ISSN (Online): 3143-1089
Volume- 2 Issue- 3 || Jul-Sep 2026 || PP. 38-42
https://gjirp.org/

validated through scratch environments and CI pipelines. The validation, but not generation alone, resulted in the improvements. Organizations should adopt the agents incrementally, restrict the autonomy to low-risk tasks and keep human approval for production changes. The future research should examine complex integrations and maintainability of the agent-authored code.

References

1. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of NAACL-HLT, 4171–4186.
2. Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The science of lean software and DevOps. IT Revolution Press.
3. Hong, S., Zhuge, M., Chen, J., et al. (2023). MetaGPT: Meta programming for a multi-agent collaborative framework. arXiv:2308.00352.
4. Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley.
5. Jimenez, C. E., Yang, J., Wettig, A., et al. (2024). SWE-bench: Can language models resolve real-world GitHub issues? International Conference on Learning Representations.
6. Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459–9474.
7. Schick, T., Dwivedi-Yu, J., Dessì, R., et al. (2023). Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36.
8. Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. IEEE Access, 5, 3909–3943.
9. Wang, L., Ma, C., Feng, X., et al. (2024). A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6), 186345.
10. Xi, Z.; Chen, W.; Guo, X.; et al. (2023). The Rise and Potential of Large Language Model-Based Agents: A Survey. arXiv:2309.07864.
11. Yao, S.; Zhao, J.; Yu, D.; et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. Proceedings of the International Conference on Learning Representations.

@2026. This is an open access article distributed under the terms of the Creative Commons License [CC BY NC 4.0] and is available on http://www.gjirp.org/