Named Ruleset: A Long-Lived Parallel Development Architecture for Enterprise-Scale Pega Implementations

Harikrishnan Muthukrishnan and Chandrashekar Konasirasagi

Abstract Pega Platform™ is a market leader in BPM/CRM, low-code, buy-and-build, and build-for-change solutions, enabling organizations to develop digital solutions more rapidly compared to traditional programming. Pega Platform™ provides a robust application development environment, with separate portals for citizen developers ("App Studio"), system architects ("Designer Studio"), and system administrators ("Admin Studio") to rapidly design, build, and manage applications. Pega Application code created using these studios is saved to rulesets, with versions, in Pega's own code repository. When development teams grow large and multiple functionalities must be developed and delivered in parallel, managing code within the same application ruleset version becomes challenging. The Pega Platform uses Pega branches to support parallel development. These Pega branches allow each team to create and update rules without impacting other teams.

Pega branching is designed for frequent code merges to trunk versions and continuous release to production. In many scenarios, frequent merging and releasing capabilities to production are not feasible for all parallel development teams; they may need to remain in development mode longer before moving to production. Providing a future version for release is a single solution that most customers adopt; it is efficient when customer priorities and release dates remain stable, but it becomes inefficient when those priorities and dates change. In this research paper, we propose an innovative solution to address this problem: a custom branch that maintains development tracks without merging into the trunk, along with its advantages and disadvantages. This proposed solution, called "Named Ruleset," enabled parallel development teams to remain in a custom branch for years while still incorporating trunk changes. We envision that this custom branching solution can address the limitations of Pega branching and advance the Pega DevOps practice for larger enterprise teams and implementations.

***Index Terms*—Continuous Delivery, Feature Toggles, Healthcare IT Systems, Low-Code Platforms, Long-Lived Development Tracks, Named Ruleset, Pega DevOps, Pega Branching and Merging, Pega Ruleset Versioning, Release Governance**

I. Introduction

OVER the last decade, all regulated industries have experienced a rapid demand for digital modernization, which is difficult to achieve through traditional programming alone due to complexity, skillset requirements, and resource constraints. Low-code application platforms emerged as a unique solution that enables organizations to build applications rapidly, using drag-and-drop UIs, reusable components, and easily configurable configurations, while also delivering secure, compliant, and integrated digital experiences [1]. Market predictions reinforce this shift: by the end of 2028, Agentic AI will be implemented via enterprise LCAPs in 4 of 5 businesses globally [2], and the global low-code development platform market is expected to reach $187.0 billion by 2030 [3], signaling a structural change in how regulated digital systems are built and maintained. One of the most significant advantages of mature low-code platforms is the ability to both buy and build. With decades of industry experience, these platforms already offer proven vertical solutions for healthcare [4], banking [5], insurance [6], and related sectors [7], built on Business Process Management (BPM), Customer Relationship Management (CRM) [8], and Sales Automation (SA) [9]. Horizontal capabilities such as drag-and-drop UI, process and logic automation, built-in integrations, DevSecOps tools, one-click deployment [10] with auto-scaling, and built-in rule-based access control (RBAC), etc., provide the platform's strength, while vertical solutions deliver industry-ready speed. Together, they let organizations modernize safely without losing control or reinventing the wheel.

Pega Platform™ has emerged as a market leader in the low-code application space for regulated enterprises, where reliability, auditability, and controlled change are essential.

Harikrishnan Muthukrishnan, Principal IT Developer, Independent Researcher, Jacksonville, Florida, USA, https://orcid.org/0009-0001-7938-9623, hari.linux@gmail.com.

Chandrashekar Konasirasagi, Principal IT Developer, Independent Researcher, Jacksonville, Florida, USA, https://orcid.org/0009-0009-4620-4424, sk.chandrashekar@gmail.com.

Pega platform provides a complete Integrated development environment through multiple development portals and a unique code repository available within the platform. In Pega, everything is a versioned rule stored in the database, retrieved, and executed in real-time by the PRPC engine. Applications, user IDs, processes, UI, APIs, and middleware settings are all managed as rules with built-in version control [11]. Pega uses multiple rulesets, each with associated versions, to store Pega rules [12]. These rule sets are part of an application rule, which includes application versions and is associated with an access group [13]. The top-level component, access group, is assigned to a user (Operator) to provide the necessary application functionality and versioning within the Pega platform. Together, these rulesets, application rules, and access groups define the structure of the code repository and user access provisioning within the Pega Platform.

II. Background and Problem Statement.

When development teams grow large and multiple functionalities must be developed and delivered in parallel, managing code within the same application ruleset becomes challenging. If two teams are working on the same application ruleset and the same rule, the rules' "check-in" back to the trunk results in a conflict. Resolving conflicts during active development/sprint work requires understanding other functionalities developed by parallel teams, which can delay overall project delivery of individual teams.

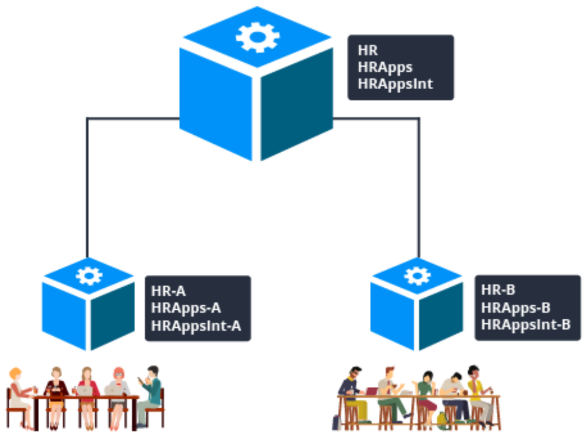


Fig. 1. Parallel Development in Pega [16]

An example of this scenario is shown in Figure 1. Team Alpha and Team Beta are developing an HR Onboarding application. Team Alpha is assigned to create a new UI feature. Team Beta is assigned to work on the candidate profile information. Although each team works independently, both features modify the same rule sets: HR, HRApps, and HRAppsInt. Developers from both teams must collaborate to resolve conflicts before they can check in code.

III. Pega Solution for Parallel Development

Pega Platform™ uses branches to support parallel development. Using Pega branches, each team creates its own development branch, which contains a branch application tag and branch rulesets. During development, each team manages changes to its own development branch rulesets, which are isolated from those of others. This enables the team to complete development without rule conflicts. When development is complete, each team merges its branch into the main ruleset, called the trunk. During this merger, Pega will identify any rule conflicts; developers must resolve them, and reviewers must approve them before merging the code.

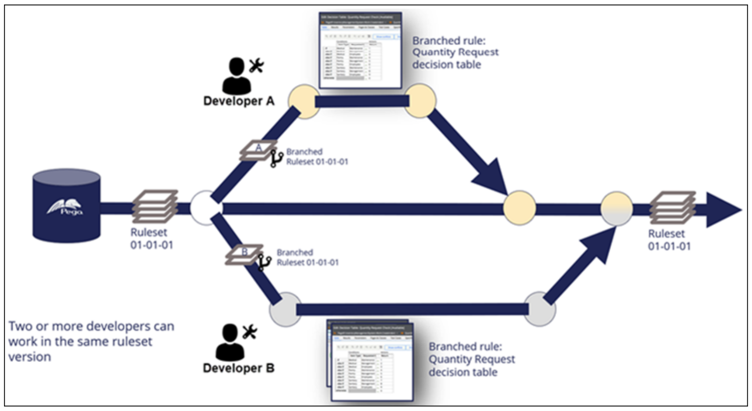


Fig. 2. Branching and Merging in Pega [15]

For example, Fig. 2, above, team Alpha and team Beta complete development and unit testing in their respective branches. Merge their branch changes into trunk, thereby making both changes available for regression testing on the trunk version.

IV. Pega Branch: Best Practices and Limitations

When working with a Pega branching solution, it is important to consider best practices, limitations, and challenges. In this section, we will cover the following aspects of working with a Rule branch: 1) best practices for using a branch, 2) limitations of using branches, and mitigations.

A. Pega best practice in branch development [15].

- Maintain short-lived branches:
  - Leverage release toggles for features in flight.
  - Leverage feature toggles for feature work that spans across multiple Development sprints.
  - Merge a branch only once.
  - Modify the leveraged features outside of a branch in coordination with the team.
- Do not add branches to the Production application; create a separate application rule for working with branches.
- Use the Sandbox environment, which allows the

association of Dynamic System Settings (DSS) and classes to branch when planning a major refactor of classes and DSS.

- Do not migrate branches from Dev to Test; merge the code to the trunk before migrating.

Work with short-lived branches: - Pega recommends merging your branch frequently with the trunk. As per Pega best practice, delaying the merger can lead to the following issues:

- Conflict resolution: - Resolving conflicts becomes increasingly complex as changes increase.
- Integration issues: - Early testing becomes difficult, especially when a production-like configuration of the application is unavailable.
- Tracking dependencies: - Managing dependencies between features is complex, as changes are not ready for early integration.

B. Limitations of Pega branches and mitigations [15].

When you work with a branch, managing certain aspects of Pega rules development will cause you to encounter the following limitations:

- Modify non-rule-resolved rules: - These types of regulations will affect everyone outside of the branch, coordinate with other developers for resolution, for example, an application record.
- Modify data instances: - You cannot modify data instances in a branch; you must associate the data instance with the application ruleset to modify.
- Modify classes in rules: - you cannot modify classes in the branch; coordinate with other developers for resolution in the development environment.
- Merge withdrawn rules: - You cannot test rules that are withdrawn in branches until after you complete a code merger. You could use a dedicated branch to withdraw rules and merge and test them independently.

V. Challenges of Pega Branching in Enterprise Implementations

Pega branches are ideal for short development without toggles, such as a minor feature or user story, that merge into the trunk within a sprint or two. Production support releases, typically scheduled every two weeks, rely extensively on Pega branches and benefit significantly from their flexibility.

For long-running parallel development teams, the best practice for Pega branching is to implement a release- or feature-level toggle in the code and merge frequently into the trunk. We identified three scenarios, listed below, in which this approach is not feasible for large organizations with long-running parallel development tracks.

A. Toggles make the code complex than the actual logic.

In real-world scenarios, we observed that Pega branching can complicate development and testing, with multiple toggles at both the release and feature levels. This also makes the code difficult to understand due to numerous toggles and scenarios, with more toggle logic than application logic in a standard implementation. Pega's solution to this toggle problem is to issue another release to remove toggles, which is impractical for project-based organizations.

B. Parallel development tracks with conflicting release dates.

In many scenarios, parallel development tracks will be underway, with the actual release date unknown. There are multiple possibilities, such as releasing all tracks simultaneously or in a different sequence, which were not finalized during the project's initiation and development phases. Developing toggle logic for all these scenarios became challenging.

C. Long-lived Development tracks.

A parallel development team can run for months to sometimes years, based on the scope of the work. Some scenarios could cause the work to be put on the shelf for years or altogether scrapped. An example is the retirement of a legacy system, repeatedly delayed due to organizational policy and budget changes. This may result in redundant code in the trunk or unused code in production for years, necessitating additional development and testing to clean up the code.

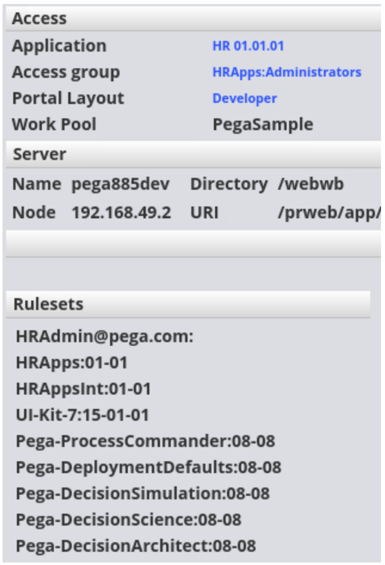
VI. Methods and Procedures: custom branching

The initial solution we implemented for parallel development teams was to provide a higher minor version in the trunk. This allocation was based on the known release-date sequence: for example, Track 1 received minor version 01.05.01, and Track 2 received minor version 01.06.01 because Track 2's release date was later than Track 1's. This would work well if there were no changes in release dates and all development tracks were made to production in sequence. However, the release dates changed as business priorities shifted, and some development was abandoned, resulting in extensive cleanup and rework of the trunk rulesets.

To address these issues with long-running development tracks, we developed a solution called “Named Ruleset”. Our design goal was to enable visibility for long-running tracks to trunk updates, avoid adding toggle logic solely for a future release, keep them in the branch until the release date is confirmed, and merge to trunk only once during the development cycle.

A. How Custom branching is designed

Our solution was to use the layer architecture of the Pega application and ruleset, and to build a custom ruleset and application on top of the trunk version. We appended a four-character track-specific name to the ruleset and application; we identified this custom branching solution as “Named Ruleset”. We also created “Named Ruleset” specific Pega access groups, appended the same four characters to these group names to support the newly created application [18], and maintained a consistent naming convention across the application. This resonated with Pega branching, a short name for the named ruleset, which I appended to application rules, rulesets, and access groups as a standard. Together, these named access groups, named applications, and named rule sets built on top of the trunk version provided branching capabilities for a long-running development team. This enabled the development team to work on features without affecting the trunk, while still monitoring trunk changes and merging them only after release dates are confirmed.



Access

Application	HR 01.01.01
Access group	HRApps:Administrators
Portal Layout	Developer
Work Pool	PegaSample

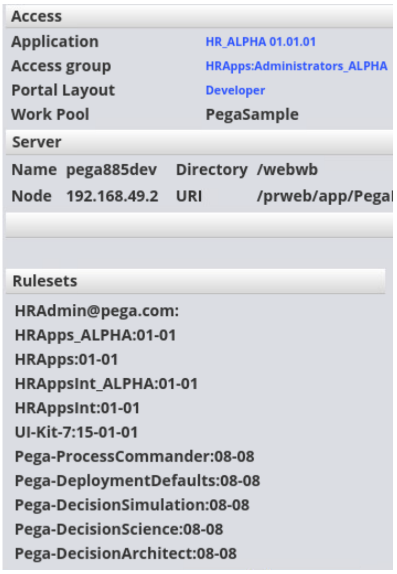
Server

Name	pega885dev	Directory	/webwb
Node	192.168.49.2	URI	/prweb/app/

Rulesets

HRAdmin@pega.com:
HRApps:01-01
HRAppsInt:01-01
UI-Kit-7:15-01-01
Pega-ProcessCommander:08-08
Pega-DeploymentDefaults:08-08
Pega-DecisionSimulation:08-08
Pega-DecisionScience:08-08
Pega-DecisionArchitect:08-08

Fig. 3. User Profile in Pega, HRApps-Trunk Version



Access

Application	HR_ALPHA 01.01.01
Access group	HRApps:Administrators_ALPHA
Portal Layout	Developer
Work Pool	PegaSample

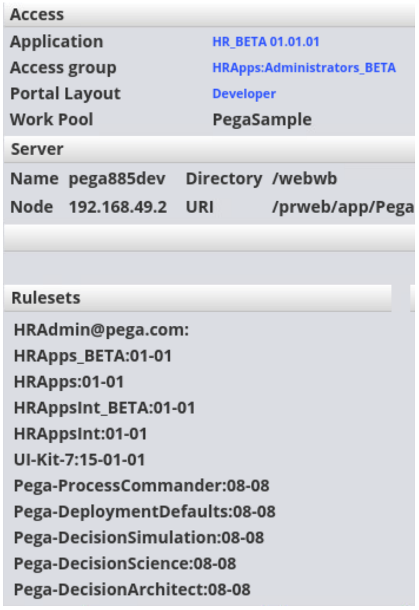
Server

Name	pega885dev	Directory	/webwb
Node	192.168.49.2	URI	/prweb/app/Pega

Rulesets

HRAdmin@pega.com:
HRApps_ALPHA:01-01
HRApps:01-01
HRAppsInt_ALPHA:01-01
HRAppsInt:01-01
UI-Kit-7:15-01-01
Pega-ProcessCommander:08-08
Pega-DeploymentDefaults:08-08
Pega-DecisionSimulation:08-08
Pega-DecisionScience:08-08
Pega-DecisionArchitect:08-08

Fig. 4. User Profile in Pega, HRApps-Alpha Version



Access

Application	HR_BETA 01.01.01
Access group	HRApps:Administrators_BETA
Portal Layout	Developer
Work Pool	PegaSample

Server

Name	pega885dev	Directory	/webwb
Node	192.168.49.2	URI	/prweb/app/Pega

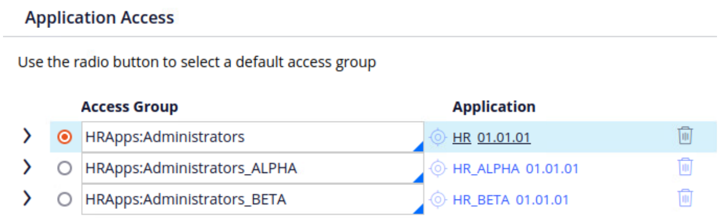
Rulesets

HRAdmin@pega.com:
HRApps_BETA:01-01
HRApps:01-01
HRAppsInt_BETA:01-01
HRAppsInt:01-01
UI-Kit-7:15-01-01
Pega-ProcessCommander:08-08
Pega-DeploymentDefaults:08-08
Pega-DecisionSimulation:08-08
Pega-DecisionScience:08-08
Pega-DecisionArchitect:08-08

Fig. 5. User Profile in Pega, HRApps-Beta Version

Named ruleset structure with Team Alpha and Beta as examples provided below in Fig. 7, please note that each team can still create branches in their own named rulesets for small development or user stories, which will help them align with Pega best practice for DevOps.

Consecutive diagrams, Fig. 3, Fig. 4, and Fig. 5, explain the order in which the "Named Rulesets" Alpha and Beta are prioritized in the developer and user profile [19], compared to trunk versions. "Named Rulesets" are always prioritized over the trunk by design, thereby providing branch capabilities.



Application Access

Use the radio button to select a default access group

	Access Group	Application
●	HRApps:Administrators	HR 01.01.01
○	HRApps:Administrators_ALPHA	HR_ALPHA 01.01.01
○	HRApps:Administrators_BETA	HR_BETA 01.01.01

Fig. 6. User Profile in Pega, HRApps-Beta Version
Access groups and Applications created as part of "Named Rulesets" Alpha and Beta are shown in Fig. 6 with four characters of identifier appended.

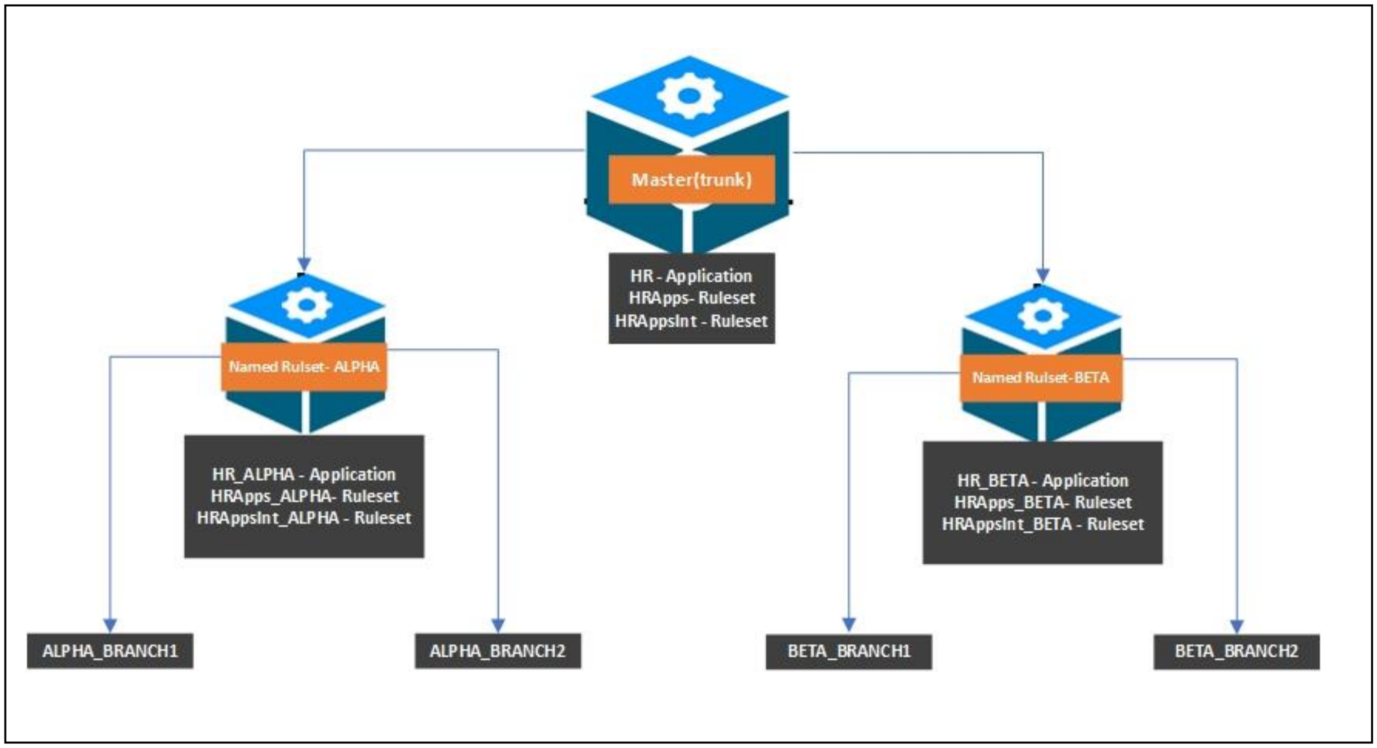


Fig. 7. Custom "Named Ruleset" Structure.

VII. Discussion and Results of Custom Branching

Our custom "Named Ruleset" enabled parallel development teams to remain in the custom branch for years while still incorporating changes from the trunk. These long-running tracks only merged into the trunk when the release date is confirmed. Thus, implementing a toggle was unnecessary to support the Pega branch and integration; it reduced complexity and saved time during development and testing.

Merge conflict resolution was straightforward because each named ruleset's check-in process was always notified of the trunk version changes.

We designed and implemented custom branching and applied our case study methodology for more than seven years across multiple projects involving Pega parallel development teams. We continue to monitor the case study results with new parallel tracks we are still onboarding. This seven-year duration is a perfect lead time to identify the pros and cons of this custom feature. Pega custom

Branching, which we named "Named Ruleset", solved the issues we faced with parallel development. The advantages and disadvantages of this solution are presented below.

A. *Pros*

- Each development team has its own custom application rule, rulesets, and access groups to independently develop/validate their functionality without impacting others.
- Named ruleset can be migrated to QA for regression and User acceptance testing.
- No need for toggle logic to support parallel development; however, it can be added at the release level if needed.
- "Named Ruleset" can see trunk version changes in real-time and retrofit trunk version changes on a planned date, not mandatory to align with production release dates.

B. *Cons*

- Initial setup of named ruleset, applications, rulesets, access groups, and associated QA/UAT login setup requires additional time.
- Due to the overhead required for the first setup, we only enforced a named ruleset for a track that has more than 3 to 4 sprints of work.

C. *Best practices*

- We only allowed named rulesets in Dev and Test Regions; merging to trunk was mandatory for migration to stage and production. This enabled testing in an actual "production-like" stage environment in terms of ruleset structure.

TABLE I
COMPARATIVE ANALYSIS TABLE

Dimension	Git/SVN	Pega Native Branching	Named Ruleset
Branch Lifetime	Short to medium	Short-lived	Long-lived (months–years)
Merge Frequency	Frequent	Frequent	Single planned merge
Toggle Dependency	High for long tracks	High	None required
Conflict Resolution	File-level, manual	Rule-level	Isolated by design
Auditability	Tool-dependent	Platform-native	Platform-native
Suitability for Regulated Environments	Moderate	Moderate	High
Runtime Awareness	Compile-time	Runtime	Runtime
Support for Uncertain Release Dates	Limited	Limited	Strong
Operational Overhead	Medium	Medium	Low after initial setup

Comparison of Git vs Pega native and Custom Named Ruleset.

D. *Comparative Analysis with Git/SVN*

- A comparative analysis of Pega Named Ruleset custom branching with traditional Git/SVN and Pega Native branching is provided in Table 1. This comparative analysis clearly demonstrates the advantages of "Named Ruleset" in enterprise-level development and release cycles spanning multiple years.

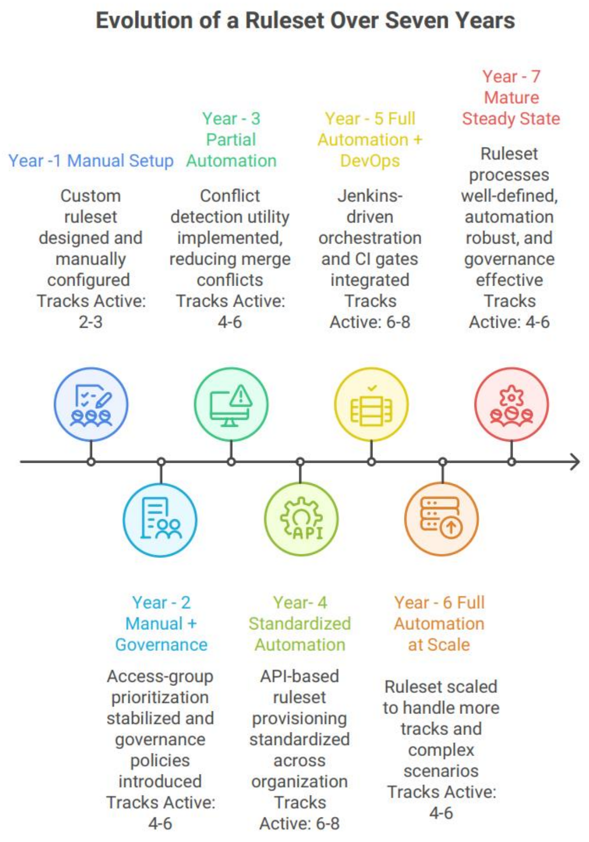

Evolution of a Ruleset Over Seven Years
Year -1 Manual Setup
Custom ruleset designed and manually configured Tracks Active: 2-3
Year - 2 Manual + Governance
Access-group prioritization stabilized and governance policies introduced Tracks Active: 4-6
Year - 3 Partial Automation
Conflict detection utility implemented, reducing merge conflicts Tracks Active: 4-6
Year- 4 Standardized Automation
API-based ruleset provisioning standardized across organization Tracks Active: 6-8
Year - 5 Full Automation + DevOps
Jenkins-driven orchestration and CI gates integrated Tracks Active: 6-8
Year - 6 Full Automation at Scale
Ruleset scaled to handle more tracks and complex scenarios Tracks Active: 4-6
Year - 7 Mature Steady State
Ruleset processes well-defined, automation robust, and governance effective Tracks Active: 4-6
API

Fig. 8. Longitudinal case study spanning 7 years.

E. *Low-Code Cross-Platform Comparison*

- Cross-platform perspective: Most enterprise low-code platforms, including market leaders such as Microsoft Power Platform, optimize parallel development through trunk-based or short-lived branching models aligned with DevOps best practices that favor frequent integration to limit merge risk [22] [23]. Platforms such as Mendix (Git-backed repositories) [24] and OutSystems (lifecycle-driven governance) [25] support collaboration at scale; however, long-running development tracks with uncertain release timelines are typically treated as exceptions due to escalating dependencies and stabilization complexity. The Named Ruleset architecture addresses this gap by enabling governance-preserving, long-lived parallel development, maintaining durable isolation while deferring trunk integration until release readiness, thereby avoiding premature mergers and excessive toggle-driven complexity in regulated enterprise environments.

VIII. AUTOMATION AND INTEGRATION FOR CUSTOM BRANCH.

As part of this architecture, we implemented full automation to create and maintain the "Named Ruleset" in multiple phases. Automation enabled us to identify and adapt to development track requirements carefully and to scale the solution for the case study. During the initial stages, we started with implementing a manual "named ruleset" creation, where the Pega administrator for the team manually created custom "Named rulesets", associated applications, and custom access groups. We also implemented automated detection of rule conflicts between "Named Ruleset" and trunk versions following rule merging into the trunk. This utility generates reports on conflicting rules that must be resolved after merging.

During the second stage, following the successful implementation of the named ruleset, as the number of named ruleset creations for the case study increased, we developed automation using Pega rules to create the "Names Ruleset" and exposed it as APIs. These APIs are then triggered via the Jenkins Orchestration platform, which provides comprehensive automation and additional DevOps capabilities.

In the third stage, we extended automation to standard Pega DevOps processes for "Names Rulesets", such as the "Lock and Roll", migrating code versions to higher environments, and validating guardrails as a stage gate, among others. The "Lock and Roll" [17] process locks one version of the ruleset and creates a new one to support continuous development, sprint cycles, and quarterly and yearly release cycles. Migrating the "named ruleset" code to higher environments and validating guardrails before migration are standard DevOps practices for the Pega platform.

Figure 8 illustrates the evolution of the Named Ruleset approach across seven years, progressing from manual setup to fully automated DevOps-driven provisioning. The longitudinal observation spans multiple concurrent enterprise development tracks, demonstrating architectural stability, scalability, and sustained operational benefits over time. In the first year, we began with a limited number of development tracks, onboarding a POC to assess the effectiveness of the new methodology, and gradually expanded the number of tracks as we implemented additional automation and governance. In a few years, the development reached a peak, with six to eight named rulesets used in parallel, and then entered a steady state, with development velocity stabilizing over time.

IX. Case study: Enterprise Healthcare Organization

A large U.S. health insurance organization operating in adherence to HIPAA and SOC2-regulated environment onboarded a mission-critical enterprise platform in legacy systems to the Pega platform with the intention of consolidating legacy applications, reducing onboarding lead times, and bringing innovations in healthcare insurance solutions.

Over time, the platform has accumulated multiple concurrent long-running business initiatives, Parallel regulatory development, enhancements, frequent production hotfixes, strict uptime (99.99%) requirements, and external auditability requirements, all of which have become standard operating procedures.

Traditional Pega versioning models, based on sequenced rule-set versions and application-level branching, began to exhibit operational strain, particularly in coordinating parallel development, as business priorities changed and release dates, testing, and stabilization plans were disrupted.

Named Ruleset Strategy: To address these issues, the platform architects introduced the Named Ruleset pattern as a long-lived, parallel-development construct rather than a short-term branching mechanism.

Measurable Outcomes: The introduction of the Named Ruleset approach delivered quantifiable, enterprise-scale impact.

- ~ 20% gain in development and testing effort for long-running development tracks due to avoidance of frequent mergers, adding toggle-level code, and testing toggle logic.
- ~50% reduction in deployment-related rework and conflict resolution compared to branch-based development.
- Zero critical production incidents attributed to parallel development collisions
- Faster regulatory response, enabling isolated compliance updates without halting other initiatives
- Improved release confidence, supporting continuous delivery with 99.99% uptime
- Operational cost savings exceeding $1M annually through reduced rework, faster remediation, and improved platform stability.

X. Conclusion.

This paper introduces the Named Ruleset, a custom parallel development architecture created to overcome the practical limitations of native Pega branching in large, long-running enterprise programs. The model allows teams to operate in purpose-aligned, long-lived branches for years while safely consuming trunk-level changes. By reducing reliance on feature toggles, avoiding premature trunk integration, and simplifying conflict resolution, the approach is well-suited to environments with extended or uncertain release timelines. Validated through multi-year healthcare and regulated-industry implementations, the Named Ruleset improved developer autonomy, strengthened release governance, reduced merge risk, and delivered an estimated 20% annual savings in development and testing effort. While the initial setup overhead is most suitable for development tracks exceeding several sprints, the long-term benefits outweigh this cost in large programs, where stability, predictability, and governance are non-negotiable.

Although this paper focused on Pega, this innovation highlights the need to move beyond short-lived branching assumptions that apply to other low-code platforms operating in regulated DevSecOps environments across multiple verticals [22]. As enterprise systems grow in scope, functionality, and longevity, parallel development models must accommodate extended timelines, shifting priorities, and regulated release controls [20]. The "Named Ruleset" approach represents a practical step in this direction, enabling a foundation for future research and tooling to support scalable, governed parallel development on enterprise low-code platforms. We believe that this custom branching solution can address the limitations of Pega branching and advance the Pega DevOps practice to the next level for AI-ready innovations [21].

References

[1] P. Chen, X. Cui, E. Xu, and H. Zhang, "Research on the Teaching Experiment Mode of Low-code Platform Integrated with Intelligent Agent," *IECT '25*, pp. 103–110, Jun. 2025, doi: 10.1145/3764206.3764222

[2] "Magic Quadrant for Enterprise Low-Code application platforms," www.gartner.com, Jul. 28, 2025. https://www.gartner.com/doc/reprints?id=1-2LJ2SZQ2&ct=250725&st=sb (accessed Dec. 23, 2025).

[3] R. A. Markets, "Global $187 billion Low-Code Development Platform market to 2030," GlobeNewswire News Room, Nov. 10, 2020. [Online]. Available: https://www.globenewswire.com/news-release/2020/11/10/2123468/0/en/Global-187-Billion-Low-Code-Development-Platform-Market-to-2030.html

[4] "Pega for Healthcare & Life Sciences | Pega," Dec. 28, 2021. https://www.pega.com/industries/healthcare

[5] "Pega Financial Services: Find solutions built for you | Pega," Oct. 08, 2021. https://www.pega.com/industries/financial-services
[6] "Insurance software | Pega," Apr. 12, 2022. https://www.pega.com/industries/insurance
[7] "Industry software solutions for workflows and decisioning | Pega," Sep. 16, 2022. https://www.pega.com/industries
[8] "Pega Customer Service built for personalized experiences | Pega," May 05, 2025. https://www.pega.com/products/customer-service
[9] "AI-powered Sales Automation Platform | Pega Sales Automation," Sep. 22, 2025. https://www.pega.com/products/sales-automation
[10] "DevOps that work with your organization | Pega," Feb. 08, 2023. https://www.pega.com/products/platform/devops
[11] "PegaSystems Documentation." https://docs.pega.com/bundle/platform/page/platform/hub/pega-platform-overview.html
[12] "Rulesets | Pega Academy," Jan. 13, 2020. https://academy.pega.com/topic/rulesets/v1
[13] "PegaSystems Documentation." https://docs.pega.com/bundle/platform/page/platform/app-dev/access-groups.html
[14] "PegaSystems Documentation." https://docs.pega.com/bundle/platform/page/platform/app-dev/branches-branch-rulesets.html
[15] "Best practices for branch-based development | Pega Academy," Sep. 08, 2021. https://academy.pega.com/topic/best-practices-branch-based-development/v1
[16] "Parallel development | Pega Academy," Mar. 04, 2020. https://academy.pega.com/topic/parallel-development/v1/in/2866
[17] "Locking and rolling ruleset versions | Pega Academy," Dec. 11, 2020. https://academy.pega.com/topic/locking-and-rolling-ruleset-versions/v2
[18] "Application versioning | Pega Academy," Mar. 04, 2020. https://academy.pega.com/topic/application-versioning/v2
[19] "Personas, operators, and work access | Pega Academy," Mar. 20, 2023. https://academy.pega.com/topic/personas-operators-and-work-access/v1
[20] B. Binzer, D. Fürstenau, and T. J. Winkler, "Bridging Business and IT Through Low-Code/No-Code: Insights into Business-IT Collaboration in Enterprise Citizen Developer Programs," Proceedings of the ... Annual Hawaii International Conference on System Sciences/Proceedings of the Annual Hawaii International Conference on System Sciences, Jan. 2025, doi: 10.24251/hicss.2025.054.
[21] "Pega GenAI: the first generative AI for the enterprise | Pega," Apr. 10, 2024. https://www.pega.com/technology/generative-ai
[22] "DORA | Capabilities: Trunk-based development." https://dora.dev/capabilities/trunk-based-development
[23] M. Heuer, C. Kurtz, and T. Böhmann, "Towards a governance of Low-Code development platforms using the example of Microsoft Power Platform in a multinational company," Proceedings of the ... Annual Hawaii International Conference on System Sciences/Proceedings of the Annual Hawaii International Conference on System Sciences, Jan. 2022, doi: 10.24251/hicss.2022.831.
[24] "Version Control Management Tools | Mendix Evaluation Guide," Mendix. https://www.mendix.com/evaluation-guide/app-lifecycle/develop/version-control/
[25] OutSystems, "Development collaboration and version control," OutSystems. https://www.outsystems.com/evaluation-guide/lifecycle-management/version-control

Harikrishnan Muthukrishnan, Principal IT Developer, Independent Researcher, Jacksonville, Florida, USA, https://orcid.org/0009-0001-7938-9623, hari.linux@gmail.com.

Harikrishnan Muthukrishnan is a Senior Member of IEEE, a Fellow of BCS, and a member of the Forbes Technology Council. He is also a Distinguished Member of the Soft Computing Research Society and serves on the Computer Advisory Board at the University of Florida.

He has authored and co-authored more than 10 peer-reviewed, high-impact publications in journals, conference proceedings, and practitioner platforms. He regularly serves as a reviewer and a member of the technical program committee for IEEE and Springer-affiliated conferences and journals. His work focuses on enterprise-scale low-code architecture, DevSecOps governance, and the modernization of regulated systems.

Chandrasekhar Konasirasagi, Principal IT Developer, Independent Researcher, Jacksonville, Florida, USA, https://orcid.org/0009-0009-4620-4424, sk.chandrashekar@gmail.com.