**GATR (Gating and Automation for Technology Releases) Documentation**

**Table of Contents**

1. **Introduction & Overview**
   * 1.1. Purpose and Vision
   * 1.2. Key Terminology (Gatr, CaaS, CI/CD, Gates)
   * 1.3. Target Audience
2. **System Architecture**
   * 2.1. High-Level CI/CD Pipeline Integration
   * 2.2. Core Components
     + 2.2.1. Detect Capabilities
     + 2.2.2. Rules Capabilities
     + 2.2.3. Configurable Rules Engine (Drools/OPA)
     + 2.2.4. API & Service Layer
     + 2.2.5. Data Sources
     + 2.2.6. Action Capabilities
     + 2.2.7. Govern / Report
3. **Roadmap**
   * 3.1. Overarching Goals
   * 3.2. Q1 2024 Highlights
   * 3.3. Q2 2024 Highlights
   * 3.4. Q3 2024 Highlights
   * 3.5. Q4 2024 & Later Highlights
   * 3.6. Recently Completed Milestones
4. **Product Backlog**
   * (Status Note)
5. **RACI (Responsible, Accountable, Consulted, Informed) & Governance**
   * 5.1. Governance Model Overview
   * 5.2. Key Roles and Responsibilities
     + 5.2.1. Policy Standard Owner
     + 5.2.2. Controls Product Manager (CPM)
     + 5.2.3. Gater Product Manager
     + 5.2.4. Gater Technical Product Owner
     + 5.2.5. Gater Engineer
     + 5.2.6. Pipeline Technology Manager
   * 5.3. Rule Authoring & Approval Process (Maker-Checker)
   * 5.4. Rule Makers and Checkers (Examples)
6. **Customer Process**
   * 6.1. Engaging with Gatr
   * 6.2. Requesting Exceptions
   * 6.3. Requesting New Controls/Gates
7. **Metrics & Dashboards**
   * 7.1. Gatr Dashboard (Link)
8. **FAQ (Frequently Asked Questions)**
9. **Implemented Features (Gate Details)**
   * 9.1. Overview Table of Implemented Gates
10. **Quality Gates Deep Dive**
    * 10.1. CDQR Quality Gates
      + 10.1.1. Overview and Purpose
      + 10.1.2. Soft-gates vs. Hard-gates
      + 10.1.3. Process Flow Diagrams
      + 10.1.4. Entitlements (Rule Lifecycle, Exception, JIRA)
    * 10.2. GEMS Quality Gates
      + 10.2.1. Overview and Purpose
      + 10.2.2. Identifying and Resolving GEM Warnings
      + 10.2.3. GEM Information (Scanned Policies)
    * 10.3. SonarQube Quality Gates
      + 10.3.1. Overview and Rule Elevation
      + 10.3.2. Blocker Rule Details
    * 10.4. Application Health Gates
      + 10.4.1. Basis (KI 2013, KI 2014, KI 2015)
      + 10.4.2. Rule Definitions
      + 10.4.3. Finding TRS Scorecard and Remediation
    * 10.5. EOVS Quality Gates
      + 10.5.1. Soft Gates and Hard Gate Status
      + 10.5.2. Sample Messages
    * 10.6. Upcoming COB Test Quality Gates
      + 10.6.1. Purpose and Status
    * 10.7. Gatr Rules Summary Table
11. **Exception Handling Processes**
    * 11.1. Hard-Gate Exception Request Process (OneApproval)
      + 11.1.1. Requesting an Exception
      + 11.1.2. Approval Workflow
    * 11.2. Hard-Gate Exception Approval Process (Legacy Jira & Transition)
      + 11.2.1. Temporary Exception Guidelines
      + 11.2.2. Multi-Step Exception Request (1st, 2nd, 3rd)
      + 11.2.3. Moving to "Plan" Status
    * 11.3. Hard-Gate Exception Approvers List
    * 11.4. OneApproval Based Exception Process
      + 11.4.1. Overview and Phased Rollout
      + 11.4.2. Requesting Roles and Steps
      + 11.4.3. Creating, Viewing, and Approving Exceptions in Gatr UI
      + 11.4.4. Build Log Failure Messages
12. **Internal Processes & Trackers**
    * 12.1. Gatr Internal Processes (e.g., CAMP Process)
    * 12.2. Tracker
      + 12.2.1. GATR API Monitoring (Summary of findings)
      + 12.2.2. ROD Issue Tracker (Summary of scope)
      + 12.2.3. SNOW SAAS Migration Tracker (Summary of scope)
13. **How-To Guides**
    * 13.1. Datalake Access and Usage (MongoDB Compass)
      + 13.1.1. Requesting Access
      + 13.1.2. Connection String
      + 13.1.3. Running Aggregation Queries
    * 13.2. Release On Demand (RoD) for USPB and Wealth
      + 13.2.1. What, Who, What's in it for me, Why?
      + 13.2.2. Key Features & Workflow Comparison
      + 13.2.3. FAQs (Pipeline Name, Jira Validation)
    * 13.3. SNOW SAAS Integration with SDLC Validation API
      + 13.3.1. Overview and Scope
      + 13.3.2. SNOW SAAS URLs
      + 13.3.3. Process Interactions (Standard Changes, Code Versioning, Code Scanning, Mainframe, NBS, ITESS, Incidents, ROD CR Validation)
14. **Adoption Tracking**
    * 14.1. Express Lane Adoption
      + 14.1.1. Service Adoption Status
      + 14.1.2. Completed Service List
    * 14.2. ROD Adoption
      + 14.2.1. Summary Dashboard
      + 14.2.2. Detailed Tracker
    * 14.3. RoD Feedback Summary
15. **Internal Documentation & Future Initiatives**
    * 15.1. AI Use Case - Self Healing "Fix Me" Assistant
      + 15.1.1. Current State, Gap, and AI Opportunity
      + 15.1.2. "Fix Me" Assistance Workflow
      + 15.1.3. Key Components and Example
      + 15.1.4. Benefits

**1. Introduction & Overview**

* **1.1. Purpose and Vision**
  + Gatr (formerly known as Controls as a Service or CaaS) is a pipeline-agnostic gating platform.
  + It is designed to implement gates at various phases of the CI/CD (Continuous Integration/Continuous Delivery) pipeline.
  + The primary goal is to implement automated pipeline controls without requiring changes to existing code or platform scripts.
  + Gatr aims to improve developer productivity by bringing automation, reliability, and reduced risk to the Architecture and Controls Space.
  + It is soon to expand to include release management and code traceability controls.
* **1.2. Key Terminology**
  + **Gatr:** The gating platform itself.
  + **CaaS:** Controls as a Service, the former name for Gatr.
  + **CI/CD:** Continuous Integration/Continuous Delivery pipeline.
  + **Gates:** Checkpoints in the CI/CD pipeline where rules are evaluated to determine if a build/deployment can proceed.
    - **Soft Gate:** Issues a warning but allows the pipeline to proceed. Often creates a JIRA ticket for tracking.
    - **Hard Gate:** Blocks the pipeline from proceeding unless an approved exception is in place.
* **1.3. Target Audience**
  + Application Development Teams
  + Pipeline Technology Managers
  + Controls Product Managers
  + Policy Standard Owners
  + Safety & Soundness Partners

**2. System Architecture**

* **2.1. High-Level CI/CD Pipeline Integration**  
  Gatr integrates into the standard CI/CD flow:  
  Commit change -> Trigger build -> Build outcome -> Notify of build outcome -> Run tests -> Notify of test outcome -> Deliver build to environment -> Deploy where necessary.
* **2.2. Core Components**  
  Gatr is an ecosystem of APIs & UIs powered by a rule engine (Drools or OPA in the backend).
  + **2.2.1. Detect Capabilities:** Determines attributes of the build (e.g., Tech Stack, Application, Project/Sprint/Release).
  + **2.2.2. Rules Capabilities:** Defines criteria and logic for gates based on detected capabilities.
  + **2.2.3. Configurable Rules Engine:** The core engine (Drools/OPA) that evaluates rules against build data.
  + **2.2.4. API & Service Layer:** Provides interfaces for CI/CD pipelines and other systems to interact with Gatr.
  + **2.2.5. Data Sources:** The rules engine uses various data sources to evaluate conditions, including:
    - CTC (Citi Technology Catalog) / CSI (Citi System Inventory)
    - JIRA
    - Artifactory
    - CMDB (Configuration Management Database) for target environment
    - Future planned data sources
  + **2.2.6. Action Capabilities:** Based on rule evaluation outcomes, Gatr can perform actions such as:
    - Failing a build
    - Issuing warnings
    - Updating JIRA tickets or other data systems
  + **2.2.7. Govern / Report:** Provides mechanisms for:
    - Managing exceptions to gate failures
    - Generating reports and dashboards
    - Backlog management for Gatr-related tasks

**3. Roadmap**

* **3.1. Overarching Goals**  
  Improving Developer Productivity while bringing automation, reliability, and reduced risk to the Architecture and Controls Space.
* **3.2. Q1 2024 Highlights**
  + OneApproval MVP1: New exception handling tool for all controls - Unify.
  + Increased Gate traceability with Audit Trail Capture for all gating streams implementation.
  + Gatr rule simulation functionality (CLI) MVP1.
  + Improved SDLC traceability with Bitbucket controls to require PTS ID or Jira ID.
  + Enhanced Gatr reporting to include LSE statistics.
* **3.3. Q2 2024 Highlights**
  + Unified pipeline exceptions - Enhancements and Pilot.
  + GATR Platform Architecture Redesign.
  + Expand Snyk Soft-gating to other sectors.
  + Improved Performance/Infrastructure Improvement.
  + Common Audit Log - Output Enhancements, Policy Catalog.
  + RCD Design and Backend APIs.
  + Express Lanes (formerly ATZ) for LSE MVP.
  + Roll-out Bitbucket Webhook.
  + Reporting for business requirement traceability for Unify Pilot.
  + Safety and Soundness.
* **3.4. Q3 2024 Highlights**
  + CI/CD APIs for LSE.
  + Express Lanes Enablement and Roll-out.
  + GitHub Migration for Gate Services.
  + RCD UI Integration (moved to P10).
  + ServiceNow SaaS integration with SDLC API.
  + Event Logger API enhancements for LS UI.
  + Exception Management via OneApproval for Unify.
  + CSI data restructuring.
* **3.5. Q4 2024 & Later Highlights**
  + Architecture Principles Validators into LS-Design.
  + Snyk Annotation API Integration.
  + Legacy CI Integration Support.
  + LSE UI Enhancements for CI/CD gates.
  + Exception Handling for LSE.
  + Express Lane Adoption.
  + Gatr Catalog Design.
  + CDDN Integration for 3 GATR services.
  + !IA&S SDLC exempt.
  + SNOW SaaS Roll-out.
  + Automated Functional Test Suite & Critical Services.
  + Stability for GATR & Critical Services.
  + RCD Integration with Implementation Approved.
* **3.6. Recently Completed Milestones (indicated by green checkmarks on roadmap)**
  + Increased gating transparency and reporting.
  + Increased adoption with standardized exception process.
  + Reduction in Vulnerabilities.
  + Improved Stakeholder Experience.
  + Increased speed in Deploying new gates.
  + Permit Compliance.
  + Improved App Health (Reduced Tier 1 Tier 2).

**4. Product Backlog**

* (As of video recording, the Jira project for the backlog was not accessible or did not exist.)

**5. RACI (Responsible, Accountable, Consulted, Informed) & Governance**

* **5.1. Governance Model Overview**
  + Rules are based on controls. Controls are mapped to a policy, standard, or decision.
  + If a new standard or decision is needed (e.g., version debt), a DAG (Decision/Version Debt anD Governance) is formed to create the standard/decision.
  + The rule written for a control will be owned and authorized by the **Controls Product Manager**.
  + The **Gater Product Manager** will review the rule to ensure proper impact analysis is done and it's mapped to a policy, standard, or decision.
  + When appropriate, the Gater Product Manager will approve the rules and include the newly added/approved rule in communications.
  + The CI/CD pipeline team will add the new rule group(s) to the pipeline.
* **5.2. Key Roles and Responsibilities**
  + **5.2.1. Policy Standard Owner:** Owns the policy/standard for the enterprise and acts as the SME for policy/standard.
  + **5.2.2. Controls Product Manager (CPM):** Owns the controls definition and rules. This is most likely the person responsible for enforcing the standards for each sector, and writing/submitting the rules in Gater.
  + **5.2.3. Gater Product Manager:** Defines the features of the Gater platform, sets the product strategy, develops the product roadmap, engages stakeholders, and defines the governance processes around the tool. This role approves rules to be added in Gater.
  + **5.2.4. Gater Technical Product Owner:** Creates and manages User Stories in Jira, runs the daily scrum calls, tests and signs off on feature releases. Reviews new rules before they are approved and is also responsible for impact analysis (in addition to the CaaS Product Manager).
  + **5.2.5. Gater Engineer:** Builds and deploys functionality within Gater and integration to the CI/CD Pipelines, Jira, Drools, and the entire Gater Tech stack. Functionality Gater engineer can make rules in Development environment.
  + **5.2.6. Pipeline Technology Manager:** Owns the CI/CD platform, sets technology & architecture roadmaps, and defines the development process & platform standards.
* **5.3. Rule Authoring & Approval Process (Maker-Checker)**
  + The Controls Product Manager will be entitled to the Gater tool's UI to author the rules.
  + The rules workflow has an author and an approver role.
  + Rule author and rule approver are separate roles and can't be the same person.
  + The Gater Product Team (Gatr Product Owner or Product Manager) will be entitled to the Gater tool's UI to approve rules.
  + Rule requests can be made through CMP Request (e.g., W5S9369427, W5S9361721).
* **5.4. Rule Makers and Checkers (Examples from table)**
  + **CDQR:**
    - Maker: Brandie Ricketts / Jeanette Osbaldeston
    - Checker: Wendy Hansory / Hridb Choudhury
  + **GEM:**
    - Maker: Wales, Adrienne / Ray Fung / Victoria Jitnic
    - Checker: Wendy Hansory / Hridb Choudhury
  + **APM:**
    - Maker: Avi Martinez
    - Checker: Wendy Hansory / Hridb Choudhury

**6. Customer Process**

* **6.1. Engaging with Gatr**
  + Gatr serves two main sets of customers:
    1. Application Teams: Impacted by controls and partners.
    2. Partners: Help create and implement new controls.
* **6.2. Requesting Exceptions**
  + Application teams should reach out to their Safety and Soundness Partners with any concerns with gating.
  + To submit an Exception request in Jira, refer to:
    1. Hard-Gate Exception Request Process
    2. Hard-Gate Exception Approval Process
    3. OneApproval based exception process (becoming the standard)
* **6.3. Requesting New Controls/Gates**
  + Partners wanting to implement a new control or Gate utilizing Gatr should refer to the governance process Rule Governance (RACI) detailed in section 5.

**7. Metrics & Dashboards**

* **7.1. Gatr Dashboard**
  + A Gatr Dashboard is available (link not captured in detail, but page exists).

**8. FAQ (Frequently Asked Questions)**

* **Where can I find more information about Gatr?**
  + Refer to "Gatr and CI/CD HardGates FAQ".
* **Are Tech Chats recorded?**
  + Yes, recordings for Gem Gating (CitiGatr\_TechChat\_3) and Application Health Gating (CitiGatr\_TechChat\_4) are available.
  + TechFest 2022 recording: CitiGatr\_TechFest\_2022.
* **When did CDQR Hard Gating start for Group 1 Applications?**
  + March 2022.
* **When did CDQR Hard gating expand to all applications on CI/CD?**
  + August 1, 2023.
* **When did GEMS Hard-Gating Start?**
  + November 28th, 2022.
* **When did Application Health Gating start?**
  + MVP1 was at the end of June (year implied as recent, likely 2023).

**9. Implemented Features (Gate Details)**

* **9.1. Overview Table of Implemented Gates (Selected Examples)**  
  | Type of Gate | Date Source | Gate | Implementation Date |  
  | :----------- | :----------------- | :--------------------------------------------------------------------------------------------------------------------------------------- | :------------------ |  
  | Soft | SonarQube | CI Pipeline Unify | WARN if any Blocker Vulnerabilities or Blocker Bugs detected by SonarQube in respective component's New Code Changes | 15 Nov 2021 |  
  | Hard | SonarQube | Fail CI Pipeline Unify if any Blocker Vulnerabilities or Blocker Bugs detected by SonarQube in respective component's New Code Changes | 05 Mar 2022 |  
  | Hard | CSJ / CAMP | FAIL CI Pipeline for aging (>15days) Continuous Data Quality Review (CDQR) errors, fed through CAMP feeds. (Group-1 apps, Error 136 excluded) | 05 Mar 2022 |  
  | Hard (ICG) | GEMS/ BlackDuck | Fail build on LSE Pipeline if a GEM Vulnerability is detected by the GEM Scanner (Black Duck Based) in the application. | 30 Apr 2022 |  
  | Hard (PBMM) | GEMS/ BlackDuck | Fail build on Unify Pipeline if a GEM Vulnerability is detected by the GEM Scanner (Black Duck Based) in the Application. MVP1 | 27 Jan 2023 |  
  | Soft (PBMM) | Archer/TRS | Warn build on Unify and LSE if High risk apps don't have remediation dates or that remediation date is past due. | 06 Jun 2023 |  
  | Hard (Build) | Build | Optional. CD Gates (LSE) for application thresholds to provide the last bit o pipeline parity to LSE compared with Unify | 12 Jan 2024 |

**10. Quality Gates Deep Dive**

* **10.1. CDQR Quality Gates**
  + **10.1.1. Overview and Purpose:** Allows process owners/controls product owners to write rules to control against Continuous Data Quality (CDQR) errors within CSI. The pipeline detects the application being built to determine if any CDQR errors exist, executes rules, and takes appropriate action.
    - Useful Resources: CAMP for CSI CDQR errors, Information on how to resolve CSI CDQR errors.
  + **10.1.2. Soft-gates vs. Hard-gates:**
    - **Soft-gates:** Implemented in Nov-21. Warn developers and create JIRA issue/tickets.
    - **Hard-gates:** Implemented in March-22 for Group-1 applications. Expanded to all applications on CI/CD Pipelines on June 28th, 2023 (not just Group-1). Block builds unless an approved exception is received.
    - **Ageing Criteria:** Effective Dec 2023, the ageing criteria for CDQR hard gate has been reduced to **15 days**. Applications with a past due CDQR Error, ageing more than 15 days, would now be blocked.
  + **10.1.3. Process Flow Diagrams:**
    - **Soft Gates:** Developer Submits Build -> Determine impacted CSI -> CSI has CDQR errors?
      * YES: For each CDQR error -> Does JIRA issue already exist?
        + YES: Soft Gate Warning
        + NO: Create JIRA Issue\*\* -> Soft Gate Warning
      * NO: Pass Build
    - **Hard Gates:** (More complex flow involving exception checks and ageing validation leading to Pass Build, Fail Build, or Execute other rules).
  + **10.1.4. Entitlements (Rule Lifecycle, Exception Approval, JIRA Issue Lifecycle)**
    - **Create Rule:** Controls Product Owner (S&S for CDQR) via Gatr UI.
    - **Approve Rule:** Gatr Product Owner / Product Manager via Gatr UI.
    - **Approving Exception:** Users part of 'Exception Approver Security Group' via JIRA (links to CDQR Exception Approver, SonarQube Exception Approvers).
    - *Note: It can take up to 3 days to flow CAMP updates in Gate.*
* **10.2. GEMS Quality Gates**
  + **10.2.1. Overview and Purpose:** GEMS (Global Event Management) identifies severe vulnerabilities that must be remediated. GEMS Controls using OICD Pipelines (LSE and Unify) are Hard Gated via Gate.
  + **10.2.2. Identifying and Resolving GEM Warnings:**
    - GEMs found in build dependencies can be seen in the build console, under "GEMs Vulnerability report" section.
    - Resolution involves remediating application demo jars (from code, pom file, dependencies, or plugins). Use latest versions of impacted dependencies.
    - Use maven-dependency-plugin to find the source of GEMs.
    - *Note: Jira exception is no longer supported for GEM; refer to the long-tail GEMS process for exceptions.*
  + **10.2.3. GEM Information (Scanned Policies - Examples):**
    - Apache\_Log4j (L0, 2.17.1)
    - Spring\_Framework (L0,L1, Various)
    - Spring\_Boot (L0,L1,L2)
    - Apache\_Struts\_S2-001\_S2-120 (L0,L1,L2)
* **10.3. SonarQube Quality Gates**
  + **10.3.1. Overview and Rule Elevation:** In Q4 2023, a new version of SonarQube rules was rolled out, including Blocker Bugs and Vulnerabilities that trigger the hard gate.
    - 167 rules were elevated to Blocker status after review by Safety & Soundness with CSO.
    - A 30-day grace period was provided before the rules took effect.
    - Warnings were elevated to Blocker status on **Jan 12th, 2024 for UAT** and on **Feb 2, 2024 for production**.
  + **10.3.2. Blocker Rule Details (Selected Examples):**
    - AWS IAM policies should not allow privilege escalation (Language: Javascript, Python, CloudFormation, Terraform, Typescript, Java; Type: Vulnerability)
    - Basic authentication should not be used (Language: XML, Python, Java; Type: Vulnerability)
    - Cipher algorithms should be robust (Language: Swift, Typescript, C, PL/SQL, C#, PHP, C++, JavaScript, Java, VB.NET, Kotlin, Objective-C; Type: Vulnerability)
    - Cryptographic keys should be robust (Language: Python, C#, JavaScript, C++, PHP, Typescript, Kotlin, Objective-C, Java; Type: Vulnerability)
    - Hashes should include an unpredictable salt (Language: Python, Java, Kotlin; Type: Vulnerability)
    - Weak SSL/TLS protocols should not be used (Language: Java, CloudFormation, PHP, Terraform, VB.NET, Kotlin, Python, JavaScript, C#, Typescript, C++; Type: Vulnerability)
* **10.4. Application Health Gates**
  + **10.4.1. Basis:** Implemented for Tier 1 and Tier 2 applications based on KI 2013, KI 2014, and KI 2015.
  + **10.4.2. Rule Definitions:**
    - **Rule 1 (KI 2013 Hard-gate):** Apps with committed retirement plans in CSI due within 60 days.
      * Conditions: App Status = Production, Retirement Commitment Level = Committed, Target Retirement Date <= 60 days, Rule-based Retirement = Yes.
    - **Rule 2 (KI 2014 Soft-gate):** Application Residual Risk Score (ARR) of last 2 months = Tier 1 or Tier 2 AND Residual Risk Remedial Action Target Date < Today's date.
      * Conditions: App Status = Production, ARR of last 2 months = Tier 1 or Tier 2, If Current Date > Remediation Target Date.
    - **Rule 3 (KI 2015 Hard-gate):** RR Rating T1/T2 without a remedial action in CSI.
      * Conditions: App Status = Production, ARR of last 2 months = Tier 1 or Tier 2, RRAction Target Date = Null.
  + **10.4.3. Finding TRS Scorecard and Remediation:**
    - TRS Scorecard can be found at: https://citiriskgroup.net/... (link provided).
    - Remediate TRS by: Remediate EOVS Component, Remove all non-compliant FID, Fix Open Risk Items.
* **10.5. EOVS Quality Gates**
  + **10.5.1. Soft Gates and Hard Gate Status:**
    - EOVS Soft gates or warning messages for applications on the OICD Pipelines (Unify & LSE) were enabled on 10/3/2023 (Unify) and 10/5/2023 (LSE).
    - If an application has a past due EOVS component associated with it, it will receive a soft-gate warning message on the build log, and a Jira record will be created.
    - EOVS Hard Gate rules are being finalized; implementation date TBD.
  + **10.5.2. Sample Messages:** Examples of Jira records and build console warnings are provided.
* **10.6. Upcoming COB Test Quality Gates**
  + **10.6.1. Purpose and Status:**
    - Upcoming COB ARP gates will warn application teams for past due (or approaching) COB test attestation date.
    - Warning messages will be displayed on the build console.
    - Data Source: CSI & COBTRAC.
    - This Gate is **on-hold** due to its dependencies on COBTRAC system, which is scheduled for decommissioning.
* **10.7. Gatr Rules Summary Table (Selected Examples)**  
  | Rule Group | Rule Name | Rule Description | Policy | Remediation Tip | Pipeline Stage |  
  | :---------- | :------------------ | :------------------------------------------------------------------------------------------------------------------------------------------------------ | :------------------------------------------------------------------------------------------------------------------------------------------------------------------ | :----------------------------------------------- | :------------- |  
  | PBMM-RG-LS | CDQR\_SoftGate | WARN for any past-due CDQR error. | MQA-194 (Group1 Applications with CDQR Errors) | CDQR issue resolution guide | Validation |  
  | PBMM-RG-LS | CDQR\_HardGate | BLOCK for an application with any CDQR error other than error number 136 if it is past-due by more than 15 days. | MQA-194 (Group1 Applications with CDQR Errors) | CDQR issue resolution guide | Validation |  
  | PBMM-RG-LS | GEM\_HardGate | BLOCK build for a pipeline by an application GEM. | | Common ways to remediate GEMs (link) | Validation |  
  | PBMM-RG-LS | AppHealth\_KI2013 | BLOCK build for an application if the app status is Production, it's committed for risk-based retirement, and target retirement date is within 60 days. | KI 2013: Apps with committed retirement plans in CSI due within 60 days. (App Status=Prod, RetirementCommit=Committed, TargetRetirementDate<=60, RuleBasedRet=Yes) | Common ways to remediate TRS for Tier 1 or Tier 2 | Validation |  
  | PBMM-RG-LS | SonarQube | BLOCK build for a pipeline with Blocker Bugs or Vulnerabilities detected by SonarQube in respective component's Overall Code. | | SonarQube Blocker Bugs/Vulns resolution guide | Validation |  
  | PBMM-RG-LS | Allow\_ReleasedBranch| Allow build based on release-branch not matching, progressing towards Pipeline Parity on LSE. | N/A | N/A | Design |

**11. Exception Handling Processes**

* **11.1. Hard-Gate Exception Request Process (OneApproval Integration - Target Q2 2024)**
  + The existing Jira exception process is being replaced by the OneApproval based process.
  + **11.1.1. Requesting an Exception:**
    1. Open the Jira ticket created for the issue.
    2. Click on the "Request Exception Approval" button.
  + **11.1.2. Approval Workflow:**
    1. Exception Approval Status changes to "APPROVAL REQUESTED".
    2. Only approvers can see the button to approve the exception and grant a timebound exception.
    3. Once approved, Exception Approval Status changes to "DECISION MADE". The Approval Decision and Exception Expiry Date will be displayed.
* **11.2. Hard-Gate Exception Approval Process (Legacy Jira & Transition)**
  + Provides guidelines for temporary exceptions of CI/CD Hard Gates using Gatr.
  + **11.2.1. Temporary Exception Guidelines:**
    1. Only provide exceptions for 30 days.
  + **11.2.2. Multi-Step Exception Request (1st, 2nd, 3rd Time):**
    1. **First Exception:** App Manager/Delegate requests in JIRA. Requires: reason, remediation plan, mitigation action (target date <= 30 days), App Manager's email approval, Technology Head (CSI) email approval, PBIO's (C6) email approval.
    2. **Second/Third Exception:** Similar process, with increasing scrutiny. For third exception, provide CAP/Project ID with target date.
  + **11.2.3. Moving to "Plan" Status:** App team clicks "Request for Exception" field in JIRA, clicks Edit, provides Target Date (should align with agreed exception date), clicks Update. JIRA issue changes to planned status.
  + *Note: Jira exception is no longer supported for GEM (refer to long-tail GEMS process).*
* **11.3. Hard-Gate Exception Approvers List (Examples)**  
  | Rule Type | NAM S&S Primary | APAC S&S Primary & Wealth | MEX S&S Primary |  
  | :---------- | :-------------------------------------------------------- | :------------------------ | :---------------------------------------------------------------------------------- |  
  | SonarQube | John Newcomer, Balaji Kaliyamoorthy | Pandey, Vivek1 | Kuri Parra, Ricardo; Jimenez Lopez, Fredy; Corona, Guillermo; Sarabia Aguilar, Jorge Ernesto |  
  | CDQR Errors | Kumar, Rajesh15 (Error #65-68, 161); For the rest: Brandie Ricketts | Pandey, Vivek1 | (Same as SonarQube) |  
  | APH (Tier 1)| Mahansaria, Seemo; Dhruv, Stanlin | Pandey, Vivek1 | (Same as SonarQube) |  
  | GEMS | *Jira exception no longer supported for GEM* | | |
* **11.4. OneApproval Based Exception Process**
  + **11.4.1. Overview and Phased Rollout:** Existing Jira-based process replaced by OneApproval.
    1. Phase 1: Launch for Unify integrated with the Gate UI to go live on **29 August 2024**.
    2. Phase 2: Launch for LSE integrated with the LSE UI is TBD.
  + **11.4.2. Requesting Roles and Steps:**
    1. Request role for Opt Out Service via COIN.
    2. Application teams request exception from Gate UI.
    3. Approvers see request in their OneApproval queue. Multiple approvers assigned.
    4. Approvals: PIIO -> App Manager -> Safety & Soundness Approver.
    5. "Long Tail GEMS Process Soon to be replaced by One Approval."
  + **11.4.3. Creating, Viewing, and Approving Exceptions in Gatr UI:**
    1. **Create:** Login to Gatr UI -> Create Exception tab. Select Gate Type (CDQR, ORM/App Health, SonarQube), enter required details (CSID, Error Detail/Pipeline Name, Justification), Submit.
    2. **View:** View Exception tab allows searching and viewing existing exceptions.
    3. **Approve:** Requests displayed only to approvers in OneApproval for action.
  + **11.4.4. Build Log Failure Messages:** Standardized error messages in build logs guide users to Gatr UI for exceptions (e.g., for CDQR, ORM, SonarQube, GEMS).

**12. Internal Processes & Trackers**

* **12.1. Gatr Internal Processes**
  + **CAMP Process:**
    1. First sprint each month - add functional day (confluence document).
    2. As part of last user sprint each month, check CAMP dashboard for overdue items.
    3. Remediate all overdue and items with due date in the next month.
    4. For services not migrated to github - Ask Gopa for latest tcp version and update in that service.
    5. For services migrated to github - Run dependency tree and update all libraries with the CAMP item to latest version from https://nexusrepository.com/.
    6. Update CAMP tracker in SharePoint.
* **12.2. Tracker**
  + **12.2.1. GATR API Monitoring:** A detailed daily log tracks API performance (load, response time, error rates) and specific errors for "CD Gates" and "CD Results" APIs. Common issues include:
    1. Gate Names in Catalog vs. API Source Code mismatch.
    2. Errors due to old manifest URLs.
    3. java.lang.NullPointerException.
    4. HTTP 503 errors from downstream services (e.g., lightspeed).
    5. Errors due to drools API.
    6. Read timeouts, Manifest API errors.
    7. Lightspeed API timeouts.
    8. Timeouts from greenzone.api.
  + **12.2.2. ROD Issue Tracker:** Tracks issues related to Release On Demand (ROD) implementation and functionality. Examples: PROD Support Task Group approvals, build version validation, rollback build functionality, Tier 0 ROD Draft acceptance, fixing acceptance test issues.
  + **12.2.3. SNOW SAAS Migration Tracker:** Tracks tasks for SNOW SAAS migration, including documentation, training, alternate approval approaches, SDLC reporting, AMA/Office Hours, support models, API testing, bot testing, and RLM validation integration.

**13. How-To Guides**

* **13.1. Datalake Access and Usage (MongoDB Compass)**
  + For data lake aggregation.
  + **13.1.1. Requesting Access:**
    - Request MongoDB Compass via Citi App Store (search "MongoDB", select "Compass").
    - Request CMP for read-only access to MongoDB via Marketplace: https://marketplace.citi-group.net/product/#/USPB\_00153\_GLOBAL.
  + **13.1.2. Connection String Format:**  
    mongodb://<DEV\_SOEID>:<Password>@<MAAS-Instance-Details>/UNIFY?authSource=admin&replicaSet=rs0&readPreference=primary&ssl=true&AllowInvalidHostnames=true&tlsAllowInvalidCertificates=true&directConnection=true
  + **13.1.3. Running Aggregation Queries:**
    - Connect to PROD instance in Compass.
    - Navigate to cicdpipeline database -> staReporting collection.
    - Go to the "Aggregations" tab.
    - Paste the aggregation query (provided in Datalake Confluence link).
    - Click "Run".
* **13.2. Release On Demand (RoD) for USPB and Wealth**
  + **13.2.1. What, Who, What's in it for me, Why?**
    - **What:** RoD is a continuous delivery tool that makes it quick and simple to deliver small incremental changes to production.
    - **Who:** Everybody on LSE (Lightspeed Enterprise)! Requirements: LSE + PTS.
    - **What's in it for me?:** Less Tail, Reduced Risk, Facilitate frequent releases, Improve DORA Metrics.
    - **Why?:** Easy to generate production releases, Auto-generation of release run book and ServiceNow tickets.
  + **13.2.2. Key Features & Workflow Comparison:** Simplified Testing, Faster Feedback, Stabilized Production, Reduced Operational Risk, Fewer Production Incidents, Culture Change. RoD significantly reduces manual steps compared to non-RoD workflows.
  + **13.2.3. FAQs:**
    - **I don't see my Pipeline Name:** In configuration setup, if pipeline name is not found, follow steps to add your pipeline (Start Configuration).
    - **Jira instance not able to validate:** Ensure the "releasedemand" FID is in Jira's serum master. Add the FID if missing.
* **13.3. SNOW SAAS Integration with SDLC Validation API**
  + **13.3.1. Overview and Scope:** SDLC Validation APM (RLM) API is integrated with both old and new versions of SNOW. It identifies if the basis of a change number (CHG) starts with CHGxxxx. If so, it is directed towards SNOW SAAS.
  + **13.3.2. SNOW SAAS URLs:** Provided for DEV, UAT, and PROD environments.
  + **13.3.3. Process Interactions (WhatProcess for Fuji and New SNOW SAAS):**
    - **Standard Changes (Agile/Small Enhancement):**
      * Fuji: Create Reference with ID "Jira Implementation", System "JIRA".
    - **Standard Changes (Waterfall):**
      * Fuji: Create Reference with ID "PTS Number", System "PTS-W".
    - **New in SNOW SAAS (General):** Create Process Interaction with Process "CSDLC", Type "Closure Evidence", Reference "PTS #".
    - **Code Versioning (Unify/LSE):**
      * Fuji: Create Reference with ID "Jenkins Reference" (Unify) or "LSE Reference" (LSE), System "Enterprise Build". *If changes are created through RLM, NO NEED to enter manually for build version.*
      * New SNOW SAAS: Process "CSDLC", Type "Build Record", Reference "LightSpeed Build Record", Summary "LSE".
    - **Code Versioning (Not on Pipeline):**
      * Fuji: Create Reference with ID "UAT CR", System "ServiceNow Project".
    - **Code Scanning (Unify/LSE/Not on Pipeline):**
      * Fuji: Create Reference with ID "Non-Unify", System "Other".
      * New SNOW SAAS: Process "CSDLC", Type "Build Record", Reference "Non-Unify Reference", Summary "Non-Unify".
    - **Mainframe (Mexico - Code Versioning):**
      * Fuji: ID "MEXICO-MAINFRAME", System "Other".
      * New SNOW SAAS: Process "CSDLC", Type "Build Record", Reference "PACKAGENAME#", Summary "MEXICO-MAINFRAME".
    - **Mainframe (Non-Mexico - Code Versioning):**
      * Fuji: ID "InforMan Reference", System "InforMan".
      * New SNOW SAAS: Process "CSDLC", Type "Build Record", Reference "InforMan Reference", Summary "InforMan".
    - **NBS (SDLC Checks):**
      * Fuji: ID "NBS", System "Other".
      * New SNOW SAAS: Process "CSDLC", Type "Closure Evidence", Reference "Nbs-#", Summary "NBS".
    - **ITESS (Skip SDLC Checks if CSI whitelisted):**
      * Fuji: Not-applicable.
      * New SNOW SAAS: Process "CSDLC", Type "Build Record", Reference "ITESS-#", Summary "ITESS".
    - **Incidents:**
      * Fuji: Create an INC, then add the INC in related records.
      * New SNOW SAAS: Create an Incident, then create an emergency CR using that incident.
    - **ROD CR Validation (Testing in Progress):**
      * Fuji/SNOW SAAS: Jira Implementation Approval needs to be added explicitly for USPBW.

**14. Adoption Tracking**

* **14.1. Express Lane Adoption**
  + **14.1.1. Service Adoption Status:** A detailed table tracks various metrics (security rating, reliability, test coverage, etc.) for services adopting Express Lane.
  + **14.1.2. Completed Service List:** Lists services that have completed Express Lane adoption, along with their status (Done, UAT) and team (OICD).
* **14.2. ROD Adoption**
  + **14.2.1. Summary Dashboard:** A bar chart shows the count of CSIs by adoption status (Ready for Release, Yet to Start, Started, In Progress, Completed) segmented by business areas (USPB, Wealth, OPS\_Tech, etc.).
  + **14.2.2. Detailed Tracker:** A comprehensive table tracks individual CSIs through the ROD adoption lifecycle, including start dates, pilot members, template approvals, CR numbers, expected PROD dates, owners, and comments.
* **14.3. RoD Feedback Summary**
  + Collects feedback from pilot users.
  + **Benefits highlighted:** Easier change request creation, reduced manual effort and errors, faster deployment of small changes, increased confidence in CHG approval.
  + **Improvements suggested:** Streamline template onboarding, allow minor changes directly to approval stage.

**15. Internal Documentation & Future Initiatives**

* **15.1. AI Use Case - Self Healing "Fix Me" Assistant**
  + **15.1.1. Current State, Gap, and AI Opportunity:**
    - **Current State:** Actionable feedback from Gatr is static, requiring manual developer intervention to resolve issues.
    - **Gap:** Manual debugging and fixing are time-consuming, delay pipelines, and increase operational overhead, often due to predictable issues (vulnerable versions, outdated libraries, known bad practices).
    - **AI Opportunity:** Combine Gatr's actionable feedback with AI-driven self-healing mechanisms to automate resolutions for known issues.
  + **15.1.2. "Fix Me" Assistance Workflow:**
    - **Gate Failure Finding:** When a gate fails, the assistant gathers contextual information.
    - **Automated Self Healing Actions:** Applies right fixes with minimal triage. Actions can be:
      * **Generate Patch:** Creates a patch (e.g., adding the correct dependency version) and raises a Pull Request (PR) for developer review.
      * **Retry:** For transient errors (e.g., downstream API timeout), automatically retries after a cooldown period.
      * **Suggestion:** In case of transitive dependency, suggests/removes top-level dependency causing the issue.
    - **Learning from feedback:** Monitors subsequent runs to validate effectiveness and continuously improves its models based on developer feedback (accepting/rejecting PRs).
  + **15.1.3. Key Components and Example:**
    - **Key Components:** Automated Fix Generator (predefined templates or AI models to generate PRs), Feedback Loop (developer actions refine future recommendations).
    - **Example (Log4j vulnerability):**
      * Pipeline fails due to vulnerable log4j (e.g., 2.13.3).
      * Extracts current (2.13.3) and required (2.17.1) versions, identifies vulnerability is linked to log4j. Gatr findings suggest 2.17.1 is a safe upgrade.
      * Updates relevant dependency (pom.xml, build.gradle, etc.) with latest version (2.17.1).
      * Raises a PR with commit message: "Fixed vulnerability in log4j by upgrading to 2.17.1 to resolve CVE-2021-44228. Verified no breaking changes."
      * Learns from feedback if developer accepts or rejects the suggestion.
  + **15.1.4. Benefits:**
    - **Developer Productivity:** Frees developers to focus on complex issues.
    - **Accuracy:** Learns from historical data, improves assistance over time.
    - **Collaboration:** Raising PRs ensures transparency.
    - **Proactive Risk Mitigation:** Avoids delays from common issues.
    - **Improved Pipeline Reliability:** Reduces failure rate for gates.