# CRM QA

## Table of Contents



# Define testing type per environment Monthly Release

**INT/QA**

Functional Testing :: Test the scenarios Identified under the Stories

Happy path scenarios :: This confirms the system's ability to successfully complete transactions and workflows without encountering errors, thereby validating the basic functionality and reliability of the integrated system

(new) Alternative Flows :: Alternative flows testing in an integration environment involves validating the system's behaviour when users take non-standard paths or encounter less common scenarios. This type of testing ensures that the system can handle variations in user actions or data inputs that deviate from the primary, expected workflow (the happy path)

(new) Negative (-Ve) Testing :: Identify Scenarios related to -Ve testing. Ex ::Boundary Values , Invalid Formats , Empty or Null Values testing , Calling API's with incorrect endpoints , Concurrency issues etc…

Exploratory Testing : Conduct to identify edge cases and unexpected behaviours. (no scripts necessary, only applicable if dedicated testing team - recommended)

UAT Readiness :: (to be included into UAT Internal Readiness) Stories moving into UAT are bug free and all tests under the stories are passed. All bugs raised in the user stories have been resolved. If any bugs remain unfixed, they should be discussed with the Product Owner (PO) and added to the project backlog in JIRA.

**UAT**

Functional Testing :: Same as above

NRT /Regression Testing (Functional Happy Path) ::  Continuous execution of regression tests to ensure that new changes do not introduce new defects.

Regression test suite's should be updated regularly to cover new features and fixed defects. (These tests scripts should be highly sophisticated such that these can be automated in future) .

Happy Path test cases from all previous functional testing can be incorporated into the Regression test suite. This ensures that critical functionality is consistently validated and that any changes or updates do not disrupt existing features. Integrating these test cases into the regression suite enhances test coverage and reliability, providing greater confidence in the stability of the system.

Prod Readiness :: Ensures that system is fully prepared for deployment to the live environment, with all critical functionalities tested, performance optimized , All bugs are tested and passed, if any bugs are not fixed , then they are created as Project backlog's in Jira.

**E2E : (Optional)**

Definition: E2E testing simulates real-world user scenarios to validate that the entire application, including its interfaces and interactions, performs seamlessly. This testing is crucial for identifying integration issues, ensuring data integrity, and providing confidence in the system's readiness for deployment.

Frequency: E2E tests should be less frequent due to their complexity and extensive coverage, while regression tests should run more frequently to validate continuous code changes.

Performance Testing <Topic to be discussed with Hugo>

Prod Readiness :: Ensures that system is fully prepared for deployment to the live environment, with all critical functionalities tested, performance optimized , All bugs are tested and passed, if any bugs are not fixed , then they are created as Project backlog's in Jira.

**PROD**

Sanity Testing :: Testers should use specific test cases designed to validate essential operations without diving into extensive testing. The goal is to quickly confirm system stability and functionality, minimizing user disruption while ensuring that the production environment remains operational and reliable.

Dedicated Test Accounts: Create dedicated test accounts that are isolated from actual user data. These accounts should mimic various user roles and permissions.

Data Tagging: Tag test data set distinctly so that it can be easily identified and segregated from real user data. This helps in monitoring and clean-up.

Delete created test records: testers should have the permission to delete data, directly using delete Button (system might need to be reviewed to allow or a person appointed to do this task)

# Define testing type per environment Weekly Release

Assumption: weekly release will not include changes on automations (Flows, triggers), if the case, then a more robust testing should be considered: Happy Path, Alternative Flows, Negative (-Ve).

**UAT**

Functional Testing ::

Test the scenarios Identified under the Stories

Happy path scenarios < This confirms the system's ability to successfully complete transactions and workflows without encountering errors, thereby validating the basic functionality and reliability of the integrated system>

PROD Readiness :: Stories moving into UAT are bug free and all tests under the stories are passed. All the bugs raised in the stories are fixed , if any bugs are not fixed , then they are created as Project backlog's in Jira.

# Define Test scripts repository (JIRA folder) for each testing type

**X-Ray Test Repository ::** This will be our test case Library with following limitation ie test repo is a project level feature, so each test repository can only contain test from its projects (we cannot mix tests from multiple projects into a single repository and organize them together)

All test cases which we create end up in Test Repo by default.

**X-Ray Test Sets ::** Is a group of tests , we can consider this to be the equivalent of a single folder inside the test repository.

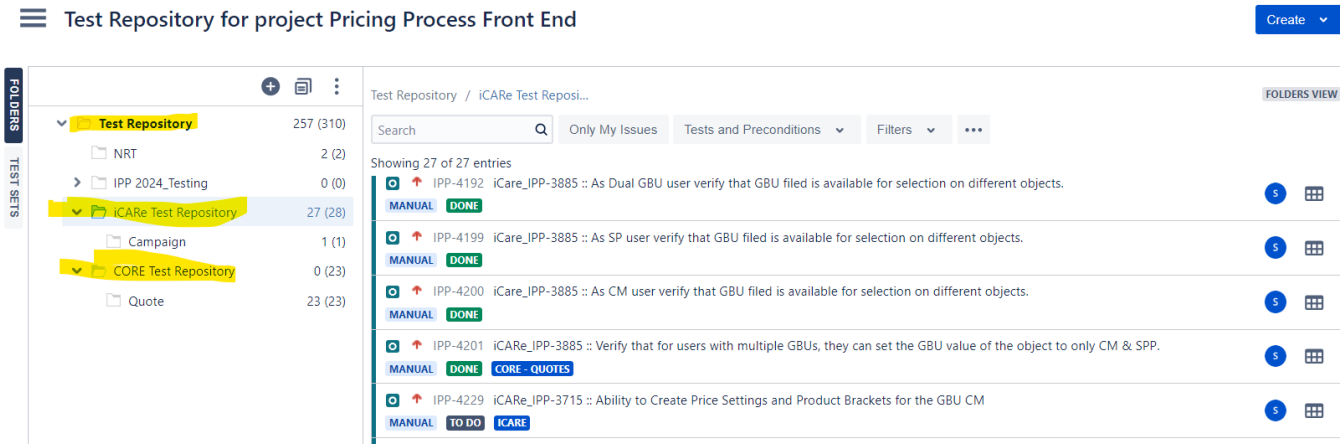
Test sets are more flexible than test repository and that a test set can contain tests from multiple projects and a single test case can live in multiple test sets.

Under test sets we create All E2E , Regression tests , Test Release plans.

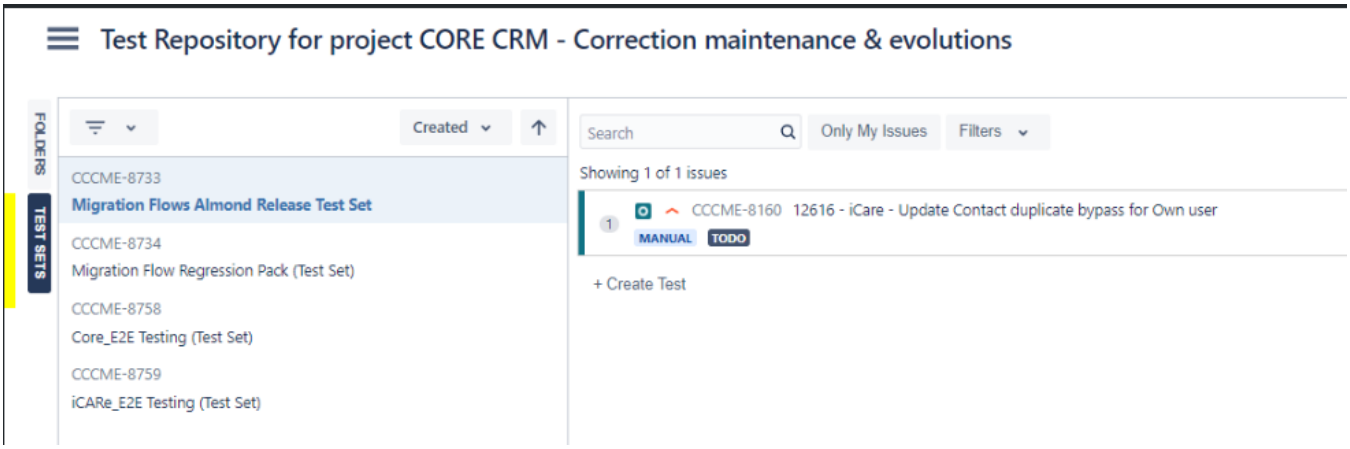
**Following are Agreed by Dina ::**

Test Repository Decision:

- Folders - will be used to structured the repository by Application >> Process >> Test Type - this folders will contain all test scripts that will be created by BAs, QA Lead will maintain the structure, and Test Scripts cannot be cloned.




- Test Sets - will be used to include all scripts needed for a certain release, this will be managed by QA Lead
- Test Sets will contain the regression pack, EtE test per release, this will be managed by QA Lead

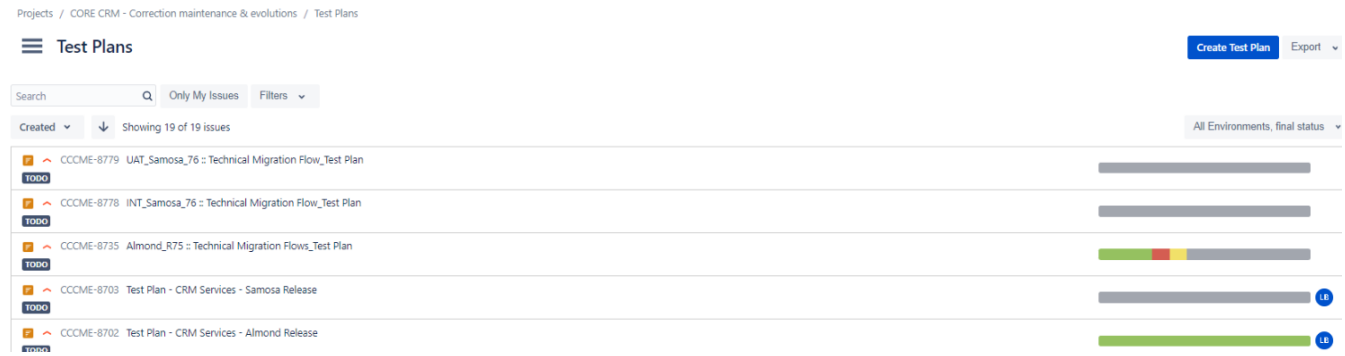


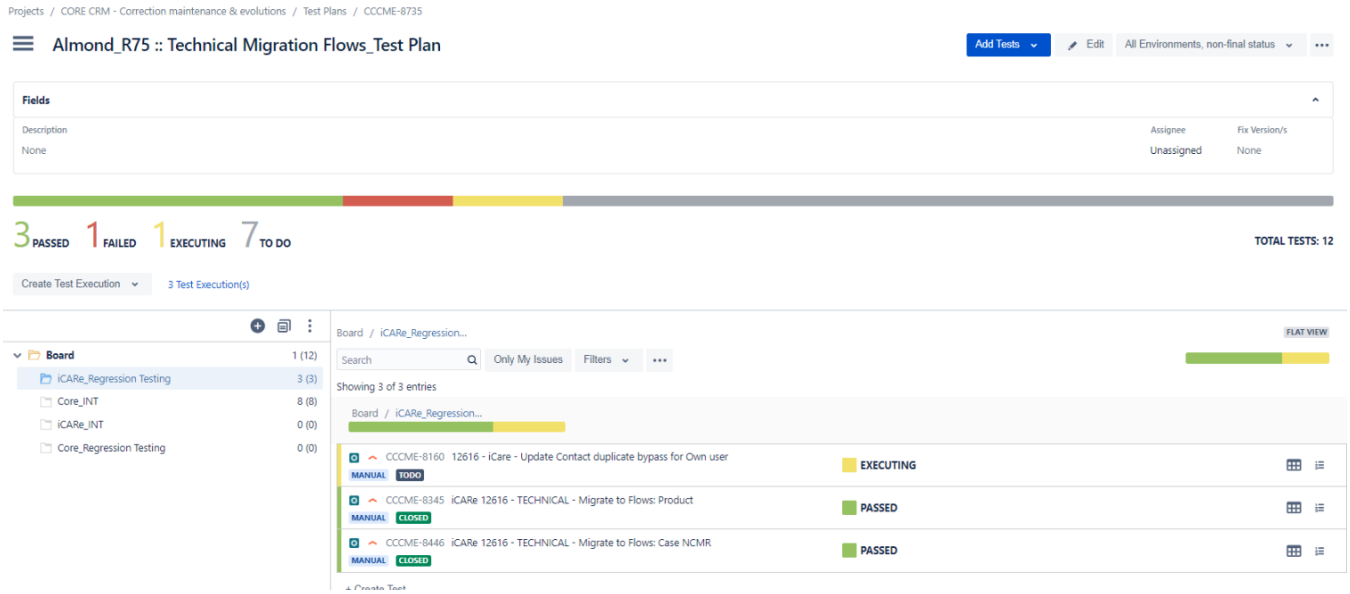

# Define how to Manage test Plans and Executions and Progress Reporting

- Test Plan - Create test plan according to Monthly Releases for INT , UAT & Prod testing

  Naming Convention = **Release Name :: Project Name_Test Plan <Almond_R75 :: Migration 2 Flows_Test Plan>**

  Test Plan contains the sub folders with test cases which specifically targets the specified testing i.e INT_Regression , INT_Core Testing and INT_iCARe Testing



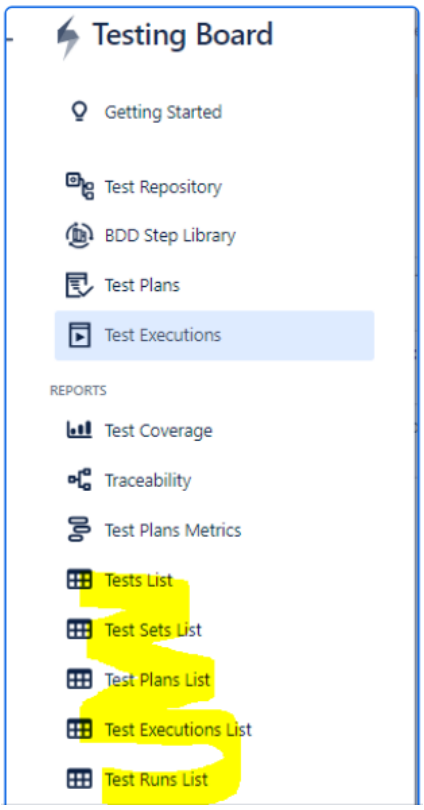



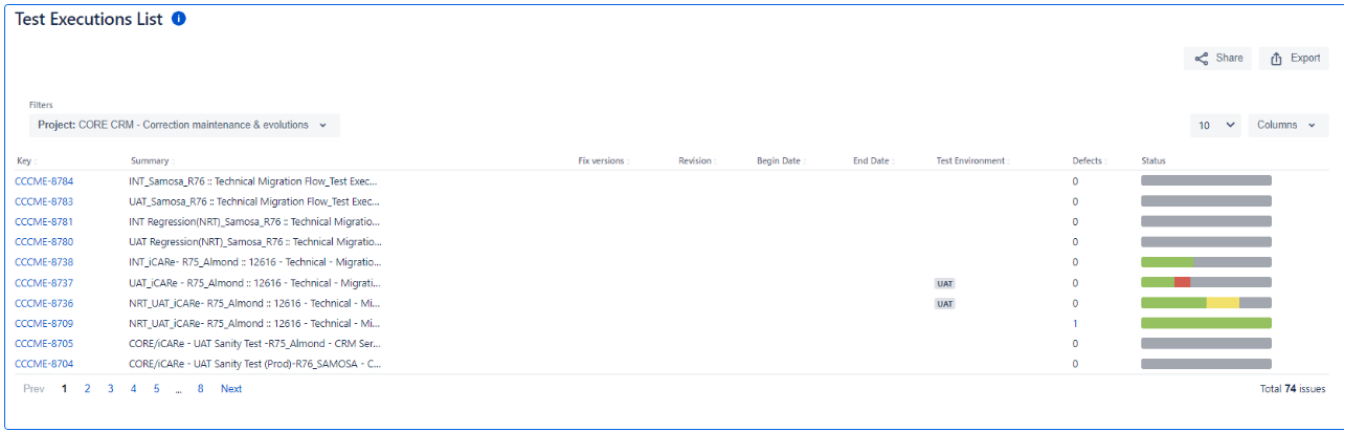

- Test Executions - Test Executions are created as children under Test Plan and will be by Release and Envoirment: INT, UAT, PROD (Sanity PROD), this will be managed by QA lead.

  **Naming Convention ::** Test Execution for Test Plan Name i.e **Test Execution for Almond_R75 :: :: Migration 2 Flows_Test Plan**

The BAs will need to create or update test scripts, identify if the test Script needs to be executed for a certain release by adding or updating the fix Version >> this will allow QA lead to create the Test Sets, Test Plan and Test execution.

**Reporting ::** Reports can be executed reports section of X-Ray based on Tests List, Test Set list , Test Plan List , Test Execution List.




# Define Testing process including Bug creation and fix

Once a story is created on Jira, the testing process generally follows these steps:

**Story Refinement Session:** A story refinement session is a collaborative meeting where the development team, product owner, and stakeholders review and clarify user stories in the product backlog. The goal is to ensure that stories are well-understood, properly scoped, and ready for upcoming sprints. During the session, the team clarifies requirements, estimates effort, discusses technical challenges, reviews and refines acceptance criteria, and prioritizes the stories. The outcome is a set of clearly defined, estimated, and prioritized stories that are ready for development, helping streamline the subsequent sprint planning process.

Here are some good habits for writing stories on JIRA to ensure clarity, completeness, and efficiency in the development and testing processes:

- **Write Clear and Concise Summaries**

  **What:** Keep the story title brief and to the point, summarizing the key action or outcome.

  **Why:** A clear title makes it easy to identify stories in JIRA's backlog, boards, or reports.

  **Example:** Instead of "Fix error," use "Fix payment gateway timeout error during checkout."

- **Use User-Centric Story Descriptions**

  **What:** Write stories from the perspective of the end user, following the “As a [user], I want [feature], so that [benefit]” format.

  **Why:** This provides clear context and helps the development and testing teams understand the business value.

  **Example:** “As a customer, I want to save my credit card details, so I can make faster payments in the future.”

- **Define Acceptance Criteria Clearly**

  **What:** List out specific, measurable conditions that must be met for the story to be considered complete.

  **Why:** Acceptance criteria guide development and testing, ensuring alignment on what "done" means.

  **Example:**

  User can add a new credit card.

  The system encrypts credit card information.

  Users are notified of a successful save.

- **Include Sufficient Details and Attachments**

  **What:** Attach relevant screenshots, wireframes, design mockups, or detailed specs.

  **Why:** Visual aids and additional details help developers and testers understand the requirement better, reducing back-and-forth clarifications.

  **Example:** Attach a UI wireframe of how the credit card form should look.

- **Break Down Larger Stories into Smaller Tasks**

  **What:** If the story is too complex, break it down into smaller, manageable tasks or sub-tasks.

  **Why:** Smaller stories make it easier to track progress, estimate effort, and ensure more frequent, smaller deployments.

  **Example:** Instead of “Implement payment module,” break it down into “Implement credit card payments,” “Implement PayPal payments,” etc.

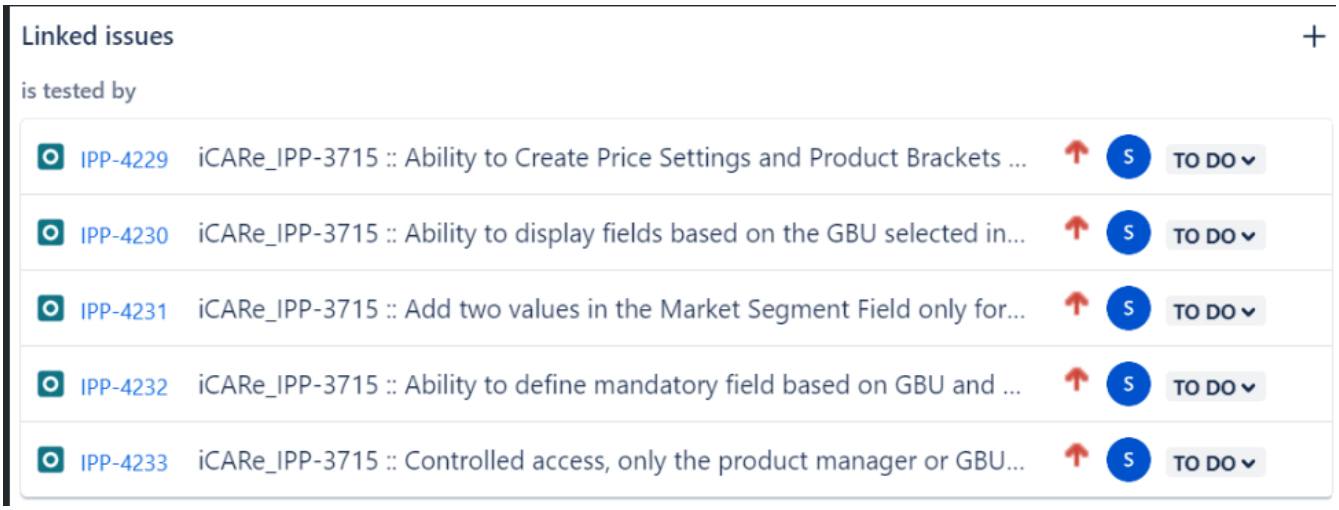
**Test Case Design:** Test Case Design involves QA engineers creating detailed test cases that outline how to validate a story's functionality. These test cases cover all possible scenarios, including both expected (positive) outcomes and potential errors or edge cases (negative scenarios). Each test case is carefully linked to the relevant user story in Jira, ensuring traceability and making it easy to track testing progress and results against specific requirements.

**Naming Convention for Test Description of a test case** (This helps in identifying the test case related to the story in test)

Ex :: iCARe_Story Number :: Description of Test / Core_Story Number :: Description of Test

iCARe_CCCME-1234 :: Verify that the user successfully logged into iCARe application.

Core_CCCME-6789 :: Verify that the user successfully logged into Core application.




**Test Execution:** Once development is complete and the code is deployed to the testing environment, typically during the Integration Testing (INT) phase, the developer hands over the story by providing a high-level demo to the QA and BA teams. This demo helps verify whether all the Acceptance Criteria (AC) have been implemented in the story. After the handover, the QA team proceeds with executing the test cases.

If any issues or bugs are found, they are logged as separate Jira tickets. The story is marked as failed, the bug is linked to the original story, and it is assigned to the responsible developer for resolution.

> **Naming Convention for Bug Description** (This helps in identifying in which env the bug was raised).
>
> Ex :: INT_iCARe :: Description of Bug/ INT_Core :: Description of the bug
>
> INT_iCARe :: User unable to log into iCARe application.

**PO Testing ::** PO testing is a phase where the Product Owner (PO) personally verifies that the completed story meets the business requirements and acceptance criteria before it moves to User Acceptance Testing (UAT). During PO Testing, the PO reviews the functionality to ensure it aligns with the expected outcomes and overall product vision. If the PO identifies any issues, they can request changes or adjustments, and any bugs found are logged in Jira for the development team to address. Once the PO is satisfied with the implementation, the story is typically approved to proceed to UAT.

**User Acceptance Testing (UAT):** User Acceptance Testing (UAT) is the final testing phase, where the story is validated by business users or stakeholders after it has passed Integration Testing (INT). In UAT, these users ensure that the implementation meets the defined acceptance criteria and aligns with business requirements. If any issues or bugs are discovered during this phase, they are logged in Jira, where they are tracked, updated, and reassigned to developers for resolution. The process continues until the users are satisfied that the system functions correctly and meets their needs, signalling that the story is ready for production.

**Story Closure:** Marks the final step in the development process, where a user story is formally completed and closed in Jira. This occurs after the story has successfully passed all testing phases, including Integration Testing (INT), Product Owner (PO) Testing, and User Acceptance Testing (UAT). The QA team or product owner verifies that all acceptance criteria have been met, all associated bugs have been resolved, and the functionality works as intended in the production environment. The Jira story's status is then updated to "Done" or "Closed," signalling that no further work is required and the feature is ready for release. Story closure also involves ensuring that any related documentation is updated and that the story is archived for future reference.

**Bug Fixing and Retesting:** Developers work on fixing any reported bugs, updating the status of the bug tickets in Jira (e.g., from "Open" to "In Progress," and then "Resolved/ Done"). The QA team then retests the fixes and updates the status of the bug tickets accordingly.

Once the bug is passed and closed, QA Team retests the associated Story and marks its status accordingly.

Note :: The bugs raised on UAT should also be tested in lower environments i.e INT.

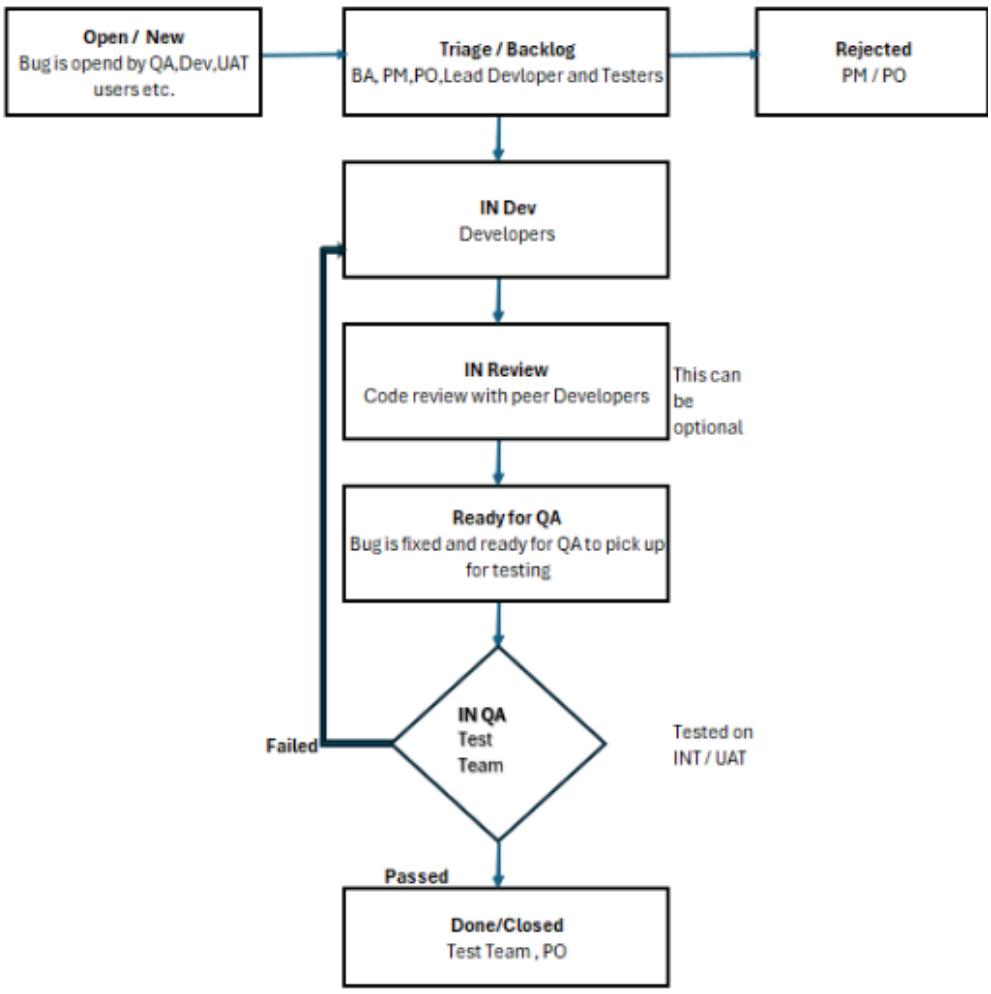
> **Naming Convention for Bug Description** (This helps in identifying in which env the bug was raised).
>
> Ex :: UAT_iCARe :: Description of Bug/ UAT_Core :: Description of the bug
>
> UAT_Core :: User unable to log into Core application.

**Jira Bug Status and Associated Roles**

- **Open/New**
  - Role: Reporter (could be a QA Engineer, Developer, or User)
  - Description: A new bug is reported and logged in Jira. The bug is in the queue, awaiting triage or assignment.
- **Triage/Backlog**
  - Role: BA, Project Manager (PM), Product Owner (PO), or Lead Developer
  - Description: The bug is reviewed for severity, priority, and assignment. The decision is made whether to work on it immediately or add it to the backlog.
- **In Dev**
  - Role: Developer
  - Description: The assigned developer starts working on the bug fix. This stage involves coding and preparing a solution.
- **In Review/Code Review (Optional)**
  - Role: Peer Developer or Lead Developer
  - Description: The fix is completed and the code is reviewed by another developer to ensure quality and adherence to standards.
- **Ready For QA**
  - Role: Developer / Test team
  - Description: The bug fix is complete and marked as Ready for QA. The resolution is ready for verification, typically by a QA Engineer.
- **IN QA**
  - Role: QA Team
  - Description: The fix is tested by the QA team to ensure it works as expected and doesn’t introduce new issues.
- **Closed/Done**
  - Role: QA Engineer or Product Owner
  - Description: After successful verification and testing, the bug is closed, marking the end of its lifecycle.
- **Rejected/Won’t Fix**
  - Role: Project Manager (PM), Product Owner (PO)
  - Description: The bug is deemed invalid, a duplicate, or is decided not to be fixed. This status is used to close issues that won't be addressed.

**Bug Priority ::** Low , Medium , High , Critical

**Flow Diagram ::**




# Define Guidelines on how to report a Bug

**Guidelines for Reporting a Bug**

Reporting a bug effectively is crucial for developers and QA teams to understand and resolve issues quickly. Below are comprehensive guidelines on how to report a bug:

**Title**

Be Specific and Concise: Summarize the issue in a short, clear sentence.

Include Key Details: Mention the area or feature affected, and the nature of the problem (e.g., "On iCARe error when selecting Products").

**Description**

Detailed Overview: Provide a brief description of the bug, including the expected behaviour and the actual behaviour observed.

Context: Mention where and how often the bug occurs, and whether it’s reproducible or intermittent.

**Steps to Reproduce**

Clear Steps: Outline the exact steps needed to reproduce the bug.

Numbered List: Use a numbered list to make it easy to follow the steps.

Initial Conditions: Include any necessary conditions or prerequisites (e.g., "User must be logged in," "Browser cache cleared").

**Expected vs. Actual Results**

**Expected Result:** Describe what should happen when the steps are followed.

**Actual Result:** Describe what actually happens, emphasizing the difference from the expected outcome.

**Environment / Sprint / Release Details**

System Information: Include details about the environment / Sprint / Release details where the bug was encountered (e.g. INT, UAT , Prod).

**Attachments**

Screenshots: Attach screenshots showing the issue, highlighting the problematic area if necessary.

Video Clips: If a screenshot isn't enough, include a short video clip showing the steps leading to the bug and the bug itself.

Logs and Error Messages: Provide any relevant logs or error messages, either copied as text or attached as files.

Test Data: Include any specific test data used to reproduce the bug.

**Severity and Priority**

Severity: Indicate the severity of the bug (e.g., Critical, Major, Minor). This assesses the impact on functionality. ****

Priority: Suggest the priority level (e.g., High, Medium, Low), indicating how soon it should be addressed.

**Additional Information**

Related Issues: Reference any related issues or bugs that might be connected to the reported problem.

Workarounds: Mention if there's a known workaround that can temporarily mitigate the issue.

**Review Before Submission**

Double-Check Details: Ensure all information is accurate, complete, and clearly presented.

Clarity: Make sure the report is easy to understand, avoiding jargon or abbreviations unless widely known.

**Example of Bug ::**

**Title ::** INT_iCare Migration 2 Flows :: System generating an error message when trying to add a Relationship to an account.

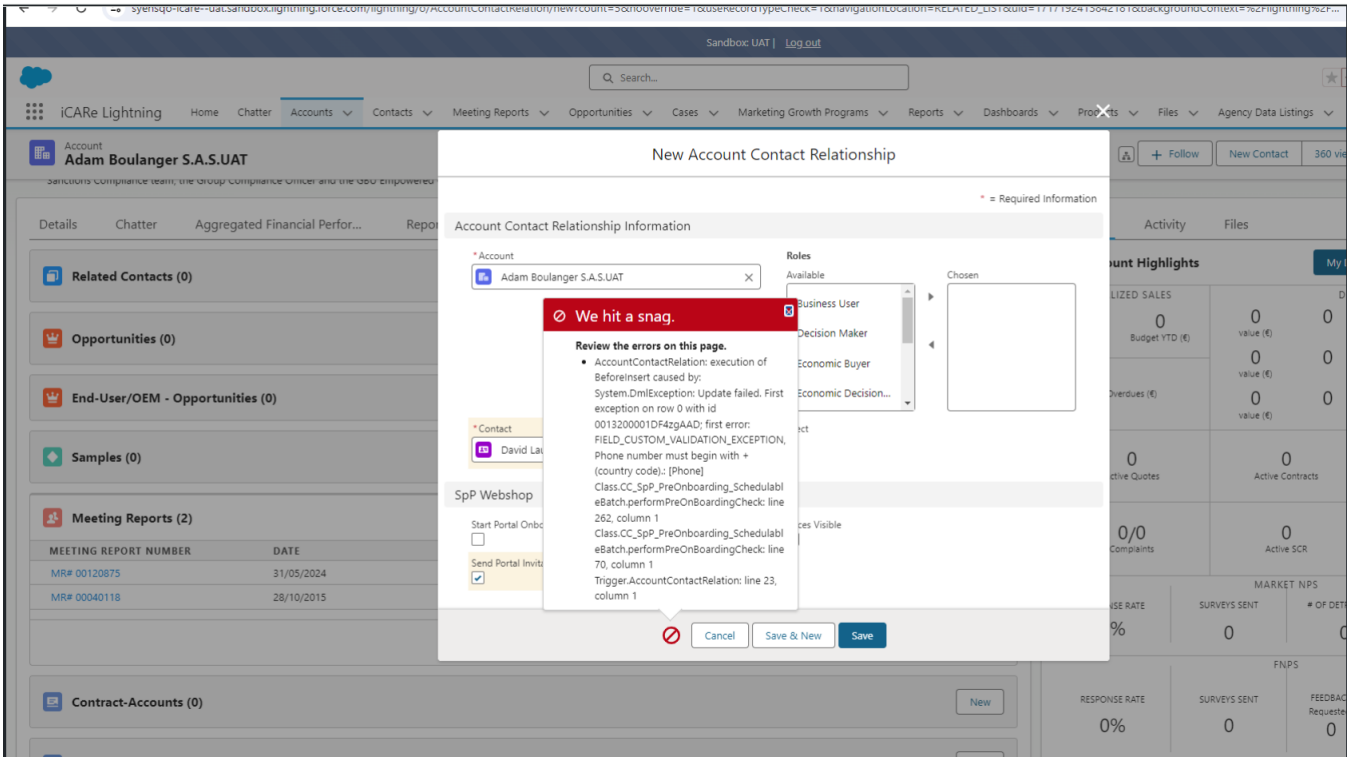
**Description**

1. Log into iCARe as Admin
2. Create a Contact ABC
3. Select an account which has no Account Origin and which is not linked to the above created contact
4. Navigate to Business Tab
5. Click on Add Relationship --> Under Contact select the above created Contact ABC
6. Select Send Portal Invitation check box and click on Save

**Expected Result** :: The record should be saved

**Actual Result** :: System generating bellow attached error message

**Attachments ::**




**Priority :** High

**Environment :** INT

**Sprint :** Sprint 123

**Release Details / Fix Version:** Samosa R76

- **Test Progress Reporting**
    - Key Metrics for Test Progress
    - Creating Effective Test Reports
    - Tools for Reporting Test Progress