

ARONE - SPIRAL

System Development Lifecycle

DOCUMENT APPROVAL:

Author	Job title (project role)	Date	Signature
Y.DARMAILLAC	Technical Manager		
P.ANSAR	Quality Manager		
E.PENE	Managing Director		
MC.PRETRE	Product Manager		

DOCUMENT HISTORY:

Version Number	Version Date	Information	Modified by
1.0		First version of this document.	Y.DARMAILLAC P.ANSAR

TABLE OF CONTENTS

	Page
<u>1</u> <u>OBJECTIVES</u>	<u>3</u>
<u>2</u> <u>REFERENCE DOCUMENTS</u>	<u>3</u>
<u>3</u> <u>ROLES AND RESPONSIBILITIES</u>	<u>3</u>
<u>4</u> <u>ORGANIZATION</u>	<u>3</u>
<u>5</u> <u>DEVELOPMENT LIFECYCLE</u>	<u>3</u>
<u>6</u> <u>CONFIGURATION MANAGEMENT</u>	<u>5</u>
<u>7</u> <u>SPECIFICATIONS MANAGEMENT</u>	<u>5</u>
<u>8</u> <u>SOURCE CODE MANAGEMENT</u>	<u>6</u>
<u>9</u> <u>TESTING MANAGEMENT</u>	<u>6</u>
<u>10</u> <u>VERSION MANAGEMENT</u>	<u>7</u>
<u>11</u> <u>RELEASE MANAGEMENT</u>	<u>7</u>

1 OBJECTIVES

SPIRAL is a platform dedicated to eDC creation and management for studies.

This document describes the quality approach followed by Arone from the design to the release of a new version of SPIRAL to its customers.

2 REFERENCE DOCUMENTS

This document is based on the following ones:

- NA

3 ROLES AND RESPONSIBILITIES

SPIRAL is managed according to an AGILE compliant methodology. The following roles are involved.

Function: Product Manager

Responsibilities:

- Organizes weekly meetings for task prioritization.
- Expresses requirements.
- Designs acceptance tests.
- Ensures communication between involved parties, internal or external to the organization.
- Gives feedback to the development team.
- Follows the opening and closing of tasks.
- Validates functionalities and mark requirements as tested.

Function: Technical Manager

Responsibilities:

- Designs the software architecture.
- Chooses / validates third party softwares and libraries.
- Defines the testing technologies and methodologies.
- Automates the integration and delivery process :
 - test to requirements tracking references validation
 - unit test execution
 - libraries bundling and deployment
 - integration test execution
- Ensures communication with HDS provider.
- Automates and optimizes feedback loop.
- Performs code reviews and proceeds to pull requests.
- Merges the tested requirements to production release.

Function: Developer

Responsibilities:

- Manages tasks assigned during weekly meetings.
- Writes unit test cases.
- Gives feedback on task advancement using comments and labels.
- Implements requirements.
- Fixes bugs.
- Emits pull requests when tasks are completed.

4 DEVELOPMENT LIFECYCLE

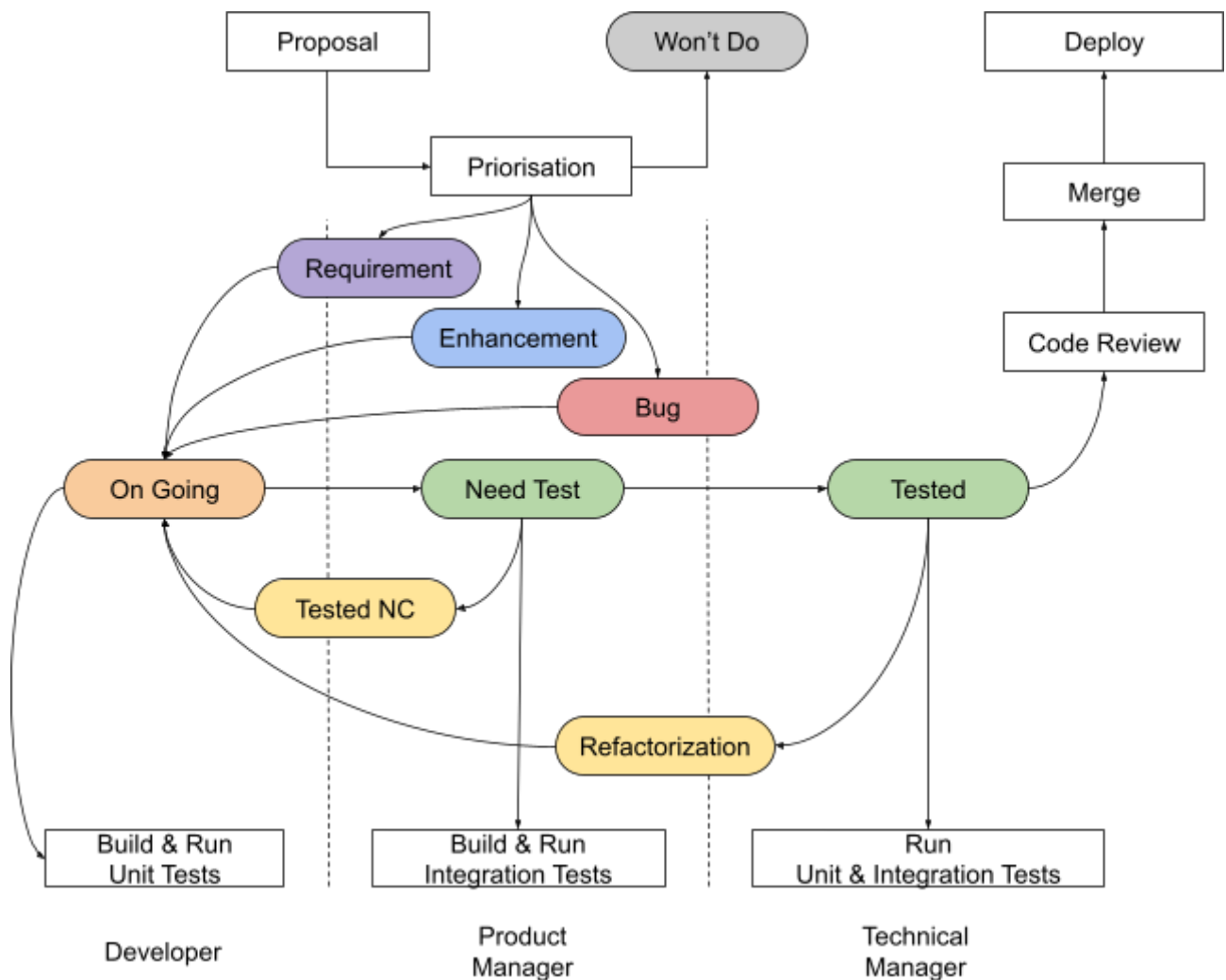
The following tools are used for the development of SPIRAL:

- Integrated Development Environment used by developers
 - o Visual Studio Code (stable version)
- Core developments
 - o JavaScript and TypeScript
 - o Node.js (JavaScript Runtime Environment)
 - o Vue.js (JavaScript Framework for user-interface management)
- Graphical User Interface
 - o HTML/CSS
 - o SASS, SCSS
- Integration testing
 - o Action Test Script
 - o TestNG framework
 - o AgiliTest software
- Unit testing
 - o tape framework
- Source Code management
 - o GitHub
- Database Management System
 - o SQLite : eCRF / ePRO / authentication data (development)
 - o PostgreSQL : eCRF / ePRO / authentication data (integration, preprod and preprod)
 - o LevelDB : quality data (requirements, test results, bugs)
- Project Management System
 - o Github

Our source code is spread among multiple repository in order to properly split:

- technical goals
- role actions

The development lifecycle of SPIRAL is managed with Github using the following workflow (states are described below):



- Proposal : an enhancement or a fix is proposed.
- Priorisation : proposal are passed to one of the 4 following states :
 - Requirement : it's the main type of issue, it represents a functional requirement.
 - Enhancement : a functionality that would enhance a requirement
 - Bug : some behavior that must be fixed regarding a requirement
 - Won't do : the proposal is rejected
- On Going : the issue is being addressed by a developer on a dedicated branch
 - Unit tests : the developer is responsible for unit testing
- Need Test : the developer thinks the requirement is fulfilled, he asks for integration testing
 - Integration tests : the product manager is responsible for integration testing
 - Tested NC : the requirement is not fulfilled, it must be fixed by the developer
- Tested : the product manager validates that the requirement is fulfilled, he asks for deployment
 - Code Review : the technical manager validates the code correctness
 - Refactorization : the developer is asked to modify the implementation
 - Tests : all tests are passed after merge of master branch to the dedicated branch
 - Merge : the dedicated branch is merged to master branch
 - Deploy : the master branch is pushed to Github repository

All colored states on the diagram exist as labels in Github repository and are used for automatic follow up of issues.

All commits, tests, enhancements or bugs must reference one or more issues using the *#<issue num>* notation :

- The corresponding requirement (mandatory)
- The corresponding bug or enhancement (optional)

References are set using the following guidelines :

- commits : in the message
- unit tests : in the test name
- integration tests : as a comment in the script
- bugs : at the beginning of the first comment
- enhancements : at the beginning of the first comment

Reference correctness is automatically checked during code merge and deployment process.

If the automated integration and deployment chain fails, it is stopped and must be fixed before further attempt.

5 CONFIGURATION MANAGEMENT

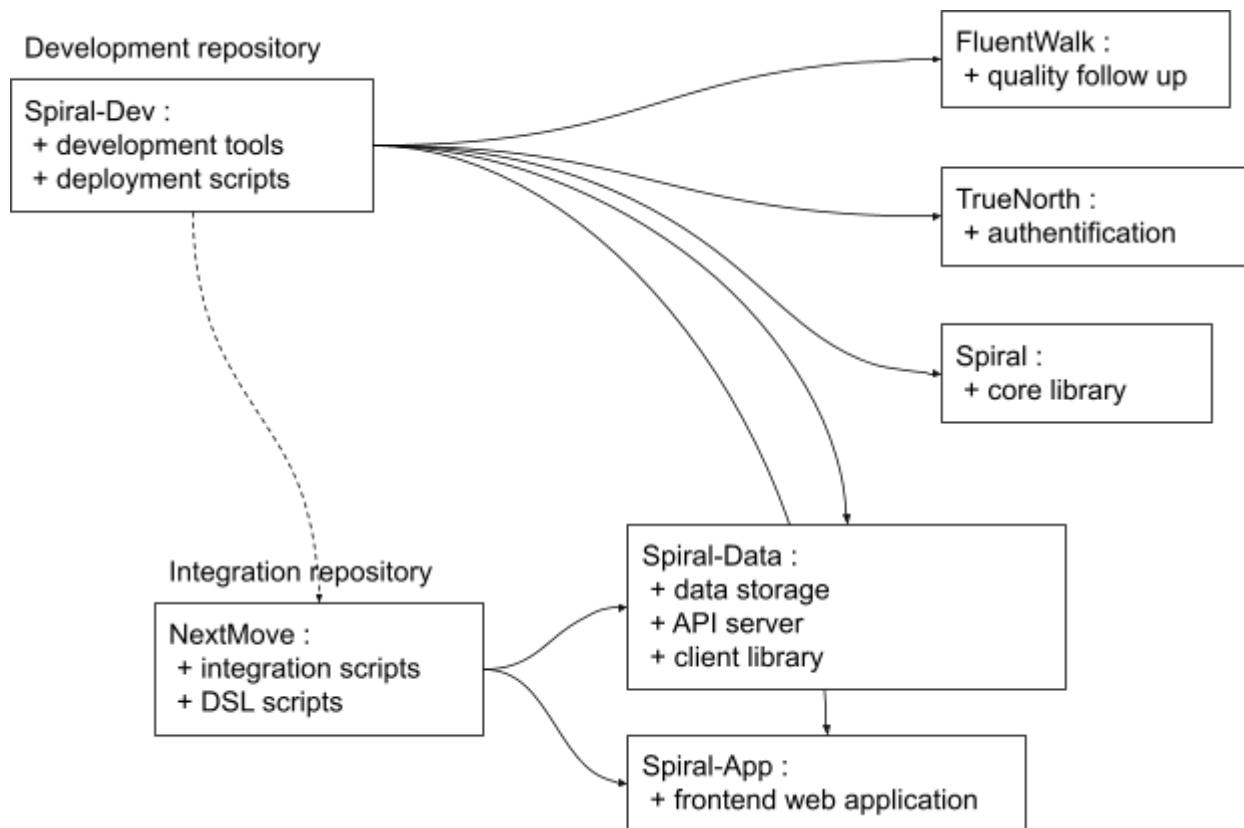
The spiral platform is managed on environments of:

- development
- integration
- preprod
- production

Each developer has a complete **development environment** on its workstation based on SQLite database system and which is entirely set up by cloning our development repository that contains all our libraries as submodules. The development repository has several deployment scripts that enable library deployments between repositories.

The project manager has an **integration environment** on its workstation along with PostgreSQL database system and AgiliTest software. It's environment is entirely set up by cloning our integration repository that contains frontend and backend submodules. Integration chain may be run by a single command from the integration environment.

The technical manager has a development and an integration environment set up. Deployment and integration chains may be run by a single command from the development environment.



The **pre-production** and **production environments** are hosted on private and secure cloud services provided by Microsoft Azure and managed by Oceanet Technology (HDS certified).



All data in relation to the SPIRAL platform is stored on a server located in Paris (EU). Due to security measures, and conformity regulations with international and country-specific standards, Microsoft Azure does not disclose the details of physical addresses to its data centers to any of its customers.

The production platform is hosted on a high availability infrastructure using:

- a redundant **xxx** service

6 SPECIFICATIONS MANAGEMENT

Specifications are managed in Github. Each specification is subject to an issue that must comply with the following guidelines :

1. Have a title that clearly identifies the requirements
 2. Have a first comment that matches the template:
The <role(s)> should/must be able to ... [in order to...]
 3. Have a label **requirement** that will be later use for searching
 4. Have an **issue number**. This number is **unique, automatically generated by GitHub** and uniquely identifies each requirement.
- The issue number is used in unit tests spec and integration test header in order to identify which tests must pass for the requirements to be met.

For example the following picture shows two requirements (#113 and #114)

The screenshot shows the SPIRAL interface. At the top, there is a header with the SPIRAL logo and user information (Pr. Dave Euloper, DÉCONNEXION). Below the header is a search bar with the text "Search for requirements" and a magnifying glass icon. The search results show a list of requirements, each with a title, a progress bar, and a "TESTS" section.

Requirement ID	Requirement Description	Progress	Tests
#124	Administrator should be able to retrieve patient modifications [audit trail]	30% - On going	✓ TESTS
#123	Administrator should be able to retrieve answer modifications [audit trail]	100% - Finished	✓ TESTS
#122	The system should automatically track patient modifications [audit trail]	100% - Finished	✓ TESTS
#121	The system should automatically track answer modifications [audit trail]	100% - Finished	✓ TESTS

Other labels can be used for issues :

- **bug**
- **enhancement**: a functionality that will improve the application but not as strong as a requirement

A dedicated engineering module in the application allows customer's administrators to :

- query issues based on their labels and keywords (e.g. *audit trail*)
- create issues: **bug**, **enhancement** or **requirement**
- read and write comments on issues, about advancement for instance

Each week the development team reviews prioritize the issues, complementary labels are added : **high priority**, **on going**, **on tests**, **done**, **closed**

Users can see the development progress and status :

The screenshot displays the SPIRAL application interface. At the top, there is a header bar with the SPIRAL logo on the left, a user profile icon and name 'Pr. Dave Euloper' in the center, and a 'DÉCONNEXION' button on the right. Below the header is a search bar with the placeholder text 'Search for requirements' and a magnifying glass icon. The search bar contains the text 'CRA'. Below the search bar, there are three requirement cards. Each card has a title, a progress bar, a status label, and a 'TESTS' button.

Requirement ID	Description	Progress	Status	Action
#119	CRA should list queries on non-checked items	30% - On going	requirement	TESTS
#114	CRA role should be able to make query in order to point out entry errors	90% - Need test	requirement	TESTS
#113	CRA role should be able to check data in order to validate data reported against source documents	80% - Tested with NC	requirement	TESTS

The following picture shows bugs corresponding to a requirement :

#121 - The system should automatically track answer modifications [audit trail] requirement

100% - Finished

✓ TESTS

^ BUGS

🕒 The system should track answer checking [#121#113]

✓ Audit trail logs same entry multiple times [#121]

The following screenshot displays test covering for a requirement :

#121 - The system should automatically track answer modifications [audit trail] requirement

100% - Finished

^ TESTS

✓ Create an audit record for interview item

✓ Store interview change payload

✓ Store interview create payload

✓ Build modification payload

✓ Build creation payload

✓ Build audit payload

Fichier : ARONE - SPIRAL - SYTEMDEVELOPMENTLIFECYCLE - DRAFT3


Page 10/15

The following screenshot show the test covering for a bug :

#121 - The system should automatically track answer modifications [audit trail] requirement

100% - Finished

✓ TESTS ^ BUGS

 The system should track answer checking [#121#113] ^


✓ Interview item label cases

✓ Checking process successes

✓ Build audit trail with checking records

✓ Store process change record

✓ Store process creation record

 Audit trail logs same entry multiple times [#121] v

7 SOURCE CODE MANAGEMENT

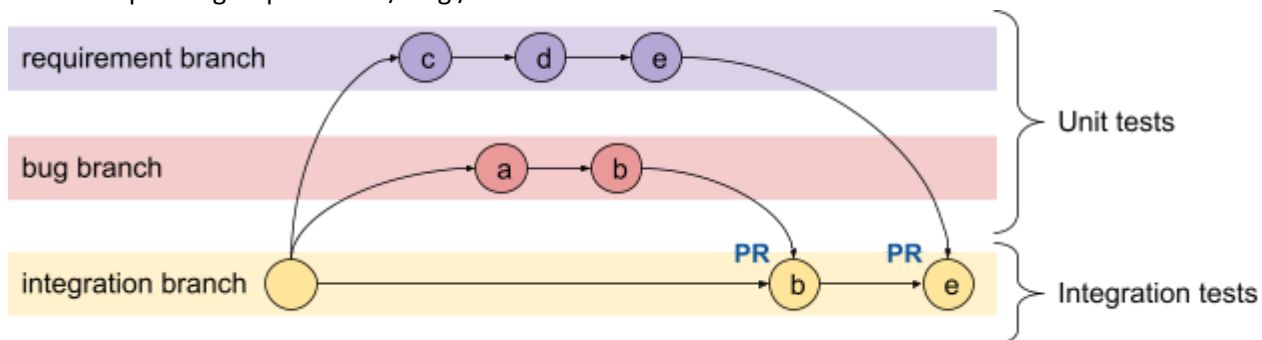
SPIRAL source code is hosted on multiple GitHub repositories managed by the technical manager. The master branches host the code that is pushed to pre-production and then to production platforms. Production releases are marked with tags on corresponding commits.

Each commit follows the following structure:

[SPECIFICATION UNIQUE ID REFERENCE] : <description>

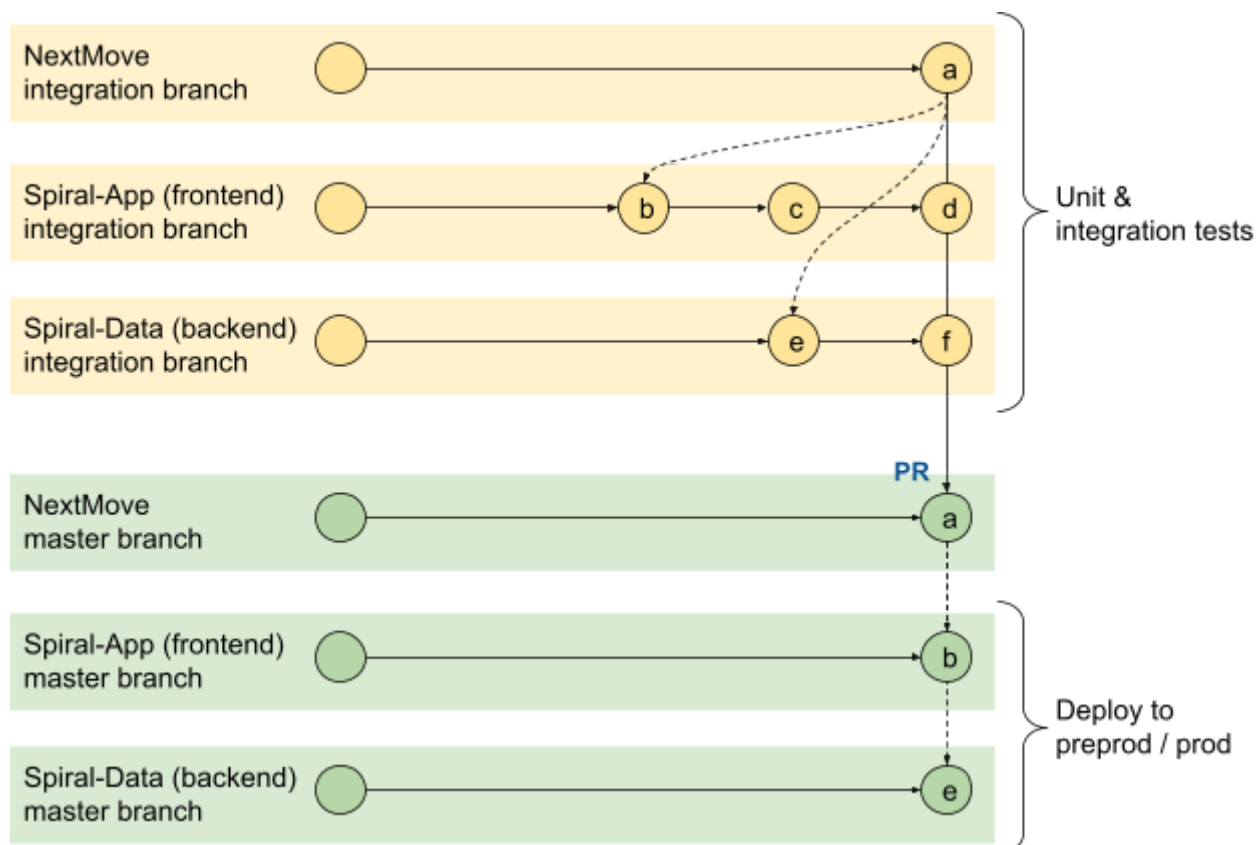
The id reference is the requirement / enhancement / bug github number. Anyway, it always refers to a requirement as bugs and enhancements are linked to requirements.

The `integration` branch contains the ongoing version of SPIRAL. Each requirement / bug / enhancement is developed on a dedicated branch. When development is over, the developer fills a pull request (PR) in order to indicate that the dedicated branch can be merged to the `integration` branch. The corresponding requirement / bug / enhancement is labelled as Need Test.



Integration tests are built and run on the `integration` branch. The NextMove repository contains the integration tests scripts and has two submodules that reference the currently validated commits of the backend (Spiral-Data repository) and the frontend (Spiral-App repository).

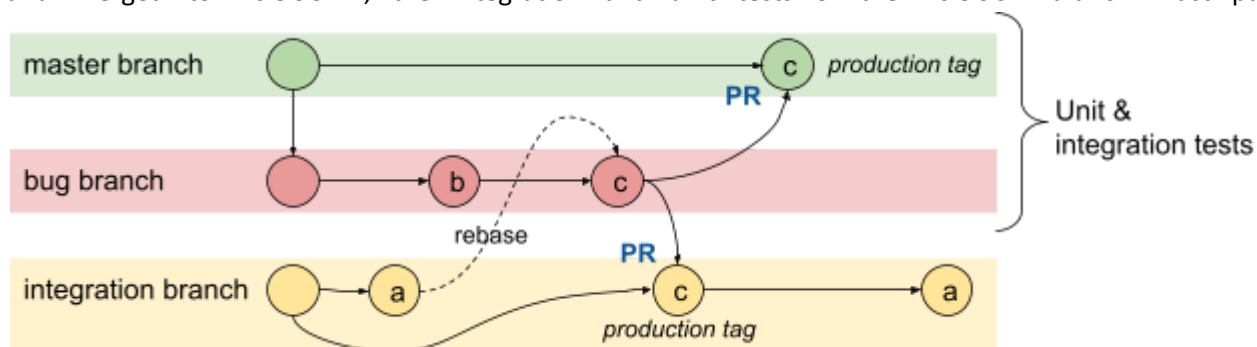
After validation the product manager updates the submodules references to point to the new validated commits and fills a pull request to indicate that these versions can be merged to `master` branches.



The technical manager merges the validated commits into frontend and backend `master` branches, rerun all unit and integration tests and push to the central repository. Every night a Github action will push the code to an Azure repository for each application server (frontend and backend) and finally the versions are deployed in pre-production.

For security reasons, we do not have access to the production environment. Our Azure provider, Oceanet Technology operates an automated deployment chain that will deploy production tagged commits from pre production repository to production.

It may happen that the production version needs to be patched when a critical bug is found on the system already in production. In this case we checkout the `master` branch, the bug is fixed in a dedicated branch and merged to `master`, the integration and unit tests of the `master` branch must pass.



The resulting commit is tagged for production and can be pushed to production as it is just ahead of the current version.

Eventually the `integration` branch is rebased onto the bug branch, thus the bug fix is ported to the ongoing release in the correct order. When the release is delivered in production, it will be ahead of the fixed version.

8 TESTING MANAGEMENT

The following testing activities are performed to ensure the quality of SPIRAL:

Unit tests

- done in development environment installed on each developer machine
- performed automatically by typescript test modules based on the tape package
- Traceability:** the Github issues number are appended to the test label, for example if 95 is the number of the “audit trail” requirement :
The audit trail must log modifications on items #95
The audit trail must log item creations #95
... #95
- Reporting:** the engineering module in our application processes the output of the tape test runner and links the result to requirements using Github API. For each requirement the customer is able to see all corresponding unit tests

Integration tests

- done in development en continuous integration environment
- performed automatically using Agilitest suite
- Traceability:** the Github issues number are prepended as a comment to the ats test source, for example if 95 is the number of the “audit trail” requirement :
// The data manager may access the audit trail to verify item history #95
- Reporting:** the engineering module in our application parses the ats test sources, and links to corresponding requirements using Github API. For each requirement the customer is able to see all corresponding integration tests. Video capture of integration tests are also available.

9 VERSION MANAGEMENT

The version of the solution is managed according to the following rules:

- Preprod environment
Version managed on 1 index and using the date of last deployment (ex: 3 2021-11-23)
A new version is deployed everyday.
- Production environment
Version managed on the 2 following indices
New versions are deployed monthly or when a major bug has to be fixed.

1 (Major)	0 (Patch)
Incremented by 1 when evolutions are implemented on a version installed in production (new needs or important evolution of expectations for existing functionalities).	Applied and incremented by 1 when a version is deployed in a production environment after a bug has been detected and fixed.

10 RELEASE MANAGEMENT

A deployment on the preprod environment is performed automatically every day from an integration environment by the technical manager.

Deployment to the production environment is done once a month or more frequently if necessary (patch). The releases are authorized by the "Release team" (Technical Manager & Product Manager).

- Planning: milestones are assigned to requirements with a due date, thus making it possible to monitor the progress of their achievement. Once finalized, the technical director or the product manager enters the planned production version number.
- Change management: the release number assigned to closed issues allows you to automatically know the content of a each release
- Risk management: Features and corrections deployed were the subject of unit tests and automated integration tests. Deployment is only carried out when all tests have passed.
- All the tests written since the beginning of the developments are executed systematically which makes it possible to avoid any regression.

Documents issued with each release:

- Factory Release Certificate,
- List of requirements and relative tests available on the application for each release