Software Requirement to Release Strategy

An instruction manual

Contents

[1 Introduction 3](#_Toc460199148)

[2 Purpose 4](#_Toc460199149)

[3 Existing Practice and its Constraints 5](#_Toc460199150)

[4 Suggested Best Practice One Pager 6](#_Toc460199151)

[5 Suggested Best Practice Detailed Instructions 8](#_Toc460199152)

[5.1 git branch naming convention 8](#_Toc460199153)

[5.2 Instructions for Developers (Development Phase) 8](#_Toc460199154)

[5.3 Instructions for Developers (Testing in TEST/SIT Phase) 8](#_Toc460199155)

[5.3.1 Resolving merge conflicts on pull-request to develop 9](#_Toc460199156)

[5.3.2 Resolving merge conflicts on pulling from latest release branch 10](#_Toc460199157)

[5.4 Instructions for Developers and Testers (UAT phase) 10](#_Toc460199158)

[5.4.1 Release build creation process 11](#_Toc460199159)

[5.5 Instructions for Developers and Release Co-ordinators (Hot-fix phase) 11](#_Toc460199160)

[6 Further Comments on improving this process 13](#_Toc460199161)

# Introduction

Somewhere in a deep forest in Burma,   
[**Commander Shears**](http://www.imdb.com/name/nm0000034/?ref_=tt_trv_qu): You mean, you intend to uphold the letter of the law, no matter what it costs?

[**Colonel Nicholson**](http://www.imdb.com/name/nm0000027/?ref_=tt_trv_qu): Without law, Commander, there is no civilization.

[**Commander Shears**](http://www.imdb.com/name/nm0000034/?ref_=tt_trv_qu): That's just my point; here, there is no civilization.

[**Colonel Nicholson**](http://www.imdb.com/name/nm0000027/?ref_=tt_trv_qu): Then, we have the opportunity to introduce it.

-------- [The Bridge on the River Kwai](http://www.imdb.com/title/tt0050212/?ref_=ttqt_qt_tt) (1957)

After various production defects arising due to miss outs and staying up late through the night until wee hours of the morning on the D-day (deployment day), DPM Vitality team members felt the need to introduce a formal convenient end-to-end Software Requirement, Development, Testing and Release Management process based upon Agile and Lean thinking.

DPM : Discovery Partner Markets

# Purpose

Have Zero defect GO-LIVE using the best Lean (remove waste) and Agile (fail early, fail consistently everywhere) practices.

It should be easy to pull-in or pull-out a feature from the release.  
It should be convenient to create a tested release candidate build for immediate release.

This document serves as an instruction manual for the all the team members involved in the various phases of software delivery from analysis of Business Requirements to the final Software product release so that they can conveniently manage their delivery.

# Existing Practice and its Constraints

[Git](https://git-scm.com) – the distributed version control system has been in use for a while now.  
The way software development works is as follows:

1. Each software developer creates a git FEATURE branch in a code repository where he/she implements the code changes for the required feature. Changes made to the code in one git branch are isolated from other changes made in another git branch in the same code repository.
2. Since there is a single development/testing environment shared among all developers for the same application, they have to merge each of their individual feature changes into a common git branch so that the single executable runtime (called as build) can be deployed onto the development/testing environment and all these features can be tested at the same time.
3. However, when it comes to PROD deployment, not all of these features are scheduled for GO-LIVE at the same time.
4. Thus, there is a need to add or remove features as per the business approval and thereby re-create the final release candidate executable build and again test the same.
5. If the above single development/testing environment constraint is removed by, say, using [Docker](https://www.docker.com/), then this document needs to be modified accordingly.

git branching strategy v1.0 achieved:

Standardisation in branch creation from common parent branch called **release** which was the latest GO-LIVE branch. This ensured all developers started with same base and their features were incremental over the common codebase.

Yet, there were production defects arising due to feature miss-outs. Hence, the need of a comprehensive strategy which covers the process of deployment and testing features was felt.

**Branch Nomenclature:**

1. **master** branch contains the code that is currently LIVE.
2. **develop (**or **build)** branch is the shared branch where all developers merge their FEATURE code to deploy to testing/SIT environment.
3. **RELEASE\_QA** branch is used to prepare the Release test candidate build to be deployed in the shared QA/UAT environment for **next immediate GO-LIVE**.
4. **release** branch is same as **master.** It is like a backup for **master.** About 2-3 days before GO-LIVE, **RELEASE\_QA** is merged into **release** and deployed to QA/UAT for regression testing if required. In VSP (Vitality Service Portal) application repository, this branch is a versioned branch. For example: **VSP.2.32.0**
5. On GO-LIVE day, **release** is merged into **master** branch from which the LIVE artefact is deployed.

# Suggested Best Practice One Pager

|  |  |
| --- | --- |
| Branch | Environment to which the artifact built from the branch is deployed : |
| **master** | PRODUCTION/LIVE environment |
| **develop** (or **build**) | SIT / TEST environment |
| **RELEASE\_QA** | QA / UAT environment |
| **release** | QA / UAT environment 2 to 3 days before GO-LIVE. |
| <FEATURE> | Your own development environment or DEV |

|  |  |
| --- | --- |
| Description | Suggested best practice |
| Agile team responsibilities | **Forming** the team for implementing the business requirements in the BRD (business requirement document), brain **storming** on the problem statements and finding solutions, **norming** i.e. resolving conflicts, if any, and exchanging service level agreements (SLA) i.e. proof of concepts of the solution and then **performing** tests on implementation to check whether business requirements are being fulfilled.  Business analyst to document the scope, processes and solution feasibility with SLA contracts WSDL/Rest services in the PRS (Product requirement specification) document. |
| Developer: Create your FEATURE branch | Create your FEATURE branch in ALL CAPITAL letters from previous GO-LIVE branch. |-- **release**          |-- AIAB-1028\_SG119\_COLD\_STORAGE\_VMP\_V1 |-- **VSP.2.30.2**         |-- AIAB-1028\_SG119\_COLD\_STORAGE\_VSP |
| Tell about it to others | Mention your FEATURE branch name in JIRA  (In case you are unavailable, others can pick up your work) |
| Code review | Raise pull request on BitBucket from your FEATURE BRANCH to **develop** (or **build** in VSP) |
| Resolve merge conflicts | Do not pull **develop/build/RELEASE\_QA** into your FEATURE branch.  Create a temporary branch from **develop/build/RELEASE\_QA** and switch to it. Then, merge your FEATURE branch into the temporary branch. Resolve merge conflicts and raise pull request from this temporary branch into **develop/build/RELEASE\_QA**. |
| Testing / SIT phase | Deploy your artefact from **develop** (or **build**) into the shared SIT/TEST environment. Testing team must test the feature end-to-end in this environment itself. |
| QA/UAT phase | **Prerequisite 1:** SIT testing is approved. **Prerequisite 2:** Ask for Business approval for your feature for the **next immediate** GO-LIVE.  If approval is received for a later dated GO-LIVE, then stay put and do not merge your feature branch into **RELEASE\_QA**.  If both prerequisites are met, then merge your FEATURE branch into **RELEASE\_QA** for deploying into QA/UAT environment. |
| GO-LIVE preparation phase | If UAT Testing is approved, your FEATURE is ready for GO-LIVE.  Merge **RELEASE\_QA** into **release** for regression testing.  On the day of GO-LIVE, merge **RELEASE\_QA** into **master** |
| Deployment check-list | BA has uploaded the latest BRD and PRS documents in the sharepoint/confluence/JIRA.  CMS (Content Management System) changes, if any, are communicated to CMS team.  Deployment checklist is signed off by all impacted teams participating in GO-LIVE and uploaded in JIRA. |

# Suggested Best Practice Detailed Instructions

## git branch naming convention

The purpose here is to identify a feature by a unique git branch name which should be searchable by others and work consistently on all operating systems (Unix/Linux/Windows/Macintosh)   
Your git feature branch name should be IN ALL CAPITAL LETTERS and must start with JIRA ID. Optionally, you may mention the CR (change request) name into your branch name.

Mention your branch name in the JIRA site as comments. This is to ensure business continuity, in case, the developer leaves the organisation before his/her feature gets approval for GO-LIVE.JIRA site: <http://plpbugz01.discsrv.co.za:8080/secure/Dashboard.jspa> (Email your Scrum Master for access)

For example:

**AIAB-2965**\_AU220\_FITNESS\_FIRST  
|\_\_\_\_\_\_\_|  
mandatory JIRA number  
Check out the JIRA comments for AU220 feature – [AIAB-2965](http://plpbugz01.discsrv.co.za:8080/browse/AIAB-2965)

## Instructions for Developers (Development Phase)

Create your FEATURE BRANCH from the latest **release** branch following the naming convention.   
    |-- release

        |-- AIAB-1028\_SG119\_COLD\_STORAGE

***Special Note for VSP:***

In Vitality Service Portal (vcs-vsp) repository, we follow the convention of **versioned release branches**.   
For example: [VSP-2.23.0-Release](https://bitbucket.org/discoveryholdings/vcs-vsp/branch/VSP-2.23.0-Release) branch is the versioned branch which went LIVE on 18th June 2015. If this is the latest VSP release, then create your git FEATURE BRANCH from the above latest **versioned release** branch.

For example: Assuming that VSP-2.23.0-Release is the latest release branch in the repository,

git checkout –b AIAB-1028\_COLD\_STORAGE VSP-2.23.0-Release

git add /path/to/VoucherFacadeImpl.java

git commit –m “added new loyalty program numbers”

git push origin AIAB-1028\_COLD\_STORAGE

## Instructions for Developers (Testing in TEST/SIT Phase)

Raise pull-request for code review to merge your code from your FEATURE BRANCH into **develop** branch (or **build** in case of VSP) to deploy to TEST/SIT environment.  
Developers need to take care not to select the checkbox option to close/delete the FEATURE branch while raising pull request.

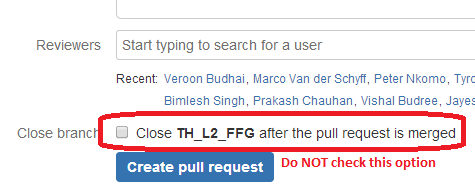


Figure 2: Uncheck the close branch option on pull request

After code-review and comments, use the same FEATURE BRANCH to continue further development.

Kindly do not delete your FEATURE BRANCH as it will be merged into **RELEASE\_QA**.

After testing in TEST/QA, if there is further development to be done for the same feature, or if there are defects to be resolved, then continue to use the same FEATURE BRANCH.

**Suggestion:**   
To reduce the number of temporary commits on your feature branch, you can use **--amend** option parameter.  
git commit --amend –m “committing again (temporary)”

### Resolving merge conflicts on pull-request to develop

This can be explained best with an example.

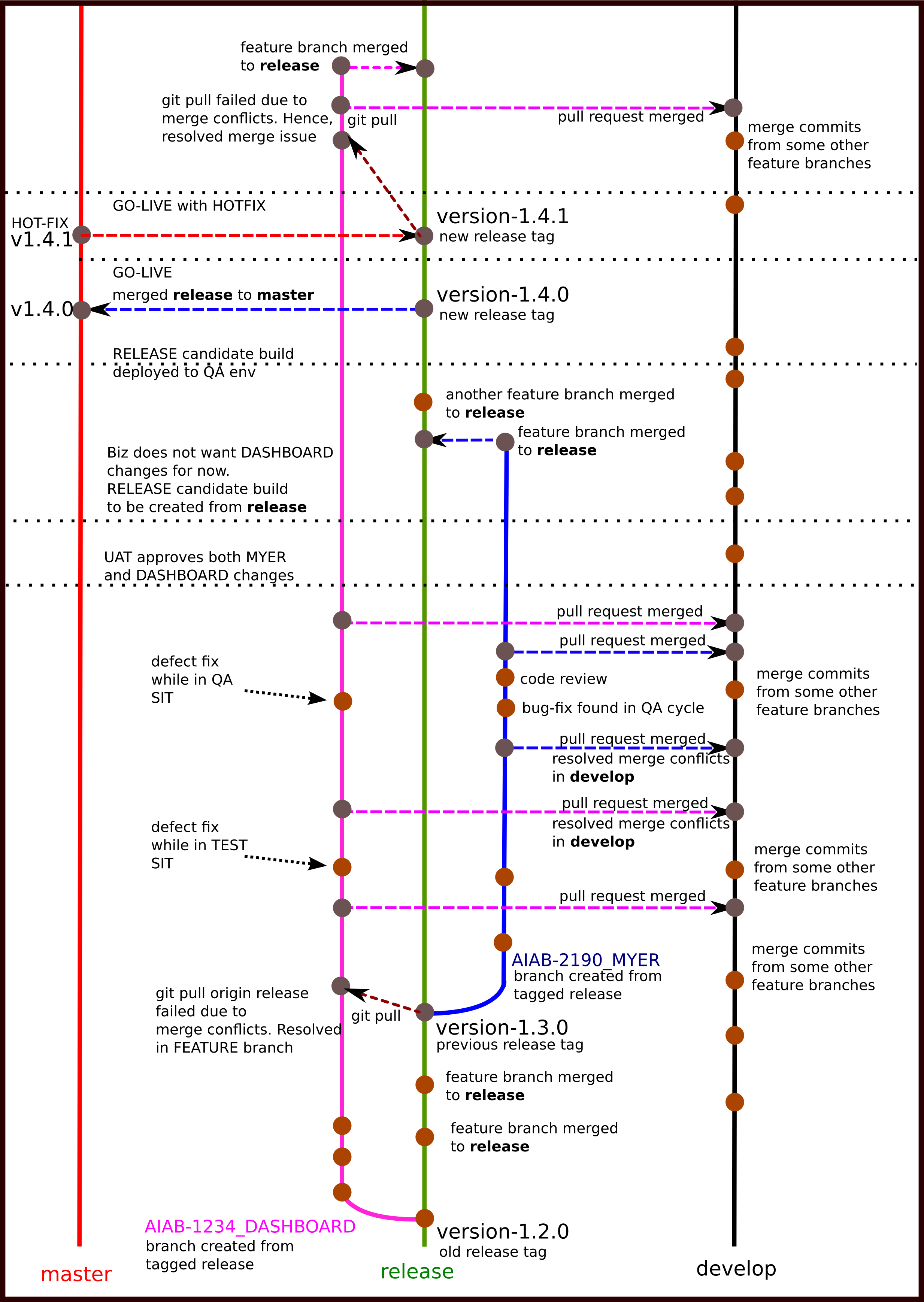


Figure 3: Resolving merge issues in **develop** branch

On raising a pull-request from FEATURE BRANCH (AIAB-2190\_MYER) to **develop**, there are merge conflicts.

1. Decline your pull-request, if you haven’t already done so.
2. On your git bash, go into your repository and get the latest **develop** branch.

git checkout **develop** && git pull origin **develop**

1. Create a new temporary branch from latest **develop**, say, TEMP\_MYER\_MERGE and switch to it

git checkout -b **TEMP\_MYER\_MERGE**

1. Merge your FEATURE BRANCH into your temporary branch

git merge **AIAB-2190\_MYER**

1. Resolve all merge conflicts and then your usual git add, git commit, and then git push to the temporary branch.
2. Raise a pull request from temporary branch **TEMP\_MYER\_MERGE** to **develop** branch.
3. **Continue your development in your FEATURE BRANCH (AIAB-2190\_MYER).** You may forget about the temporary branch you had created to resolve merge conflicts. Ultimately, it is your FEATURE BRANCH going into **release** and then **master**. So, there is no guarantee that the same merge conflicts will occur when merging into **release** or **master**.
4. **DO NOT MERGE develop or build branch into your FEATURE BRANCH in the attempt to resolve conflicts in your FEATURE BRANCH.** Doing so will bring all the commits on develop/build branch into your FEATURE BRANCH. This is exactly what we are trying to avoid by this new process.

### Resolving merge conflicts on pulling from latest release branch

If your feature is under development over many releases, then, you must keep updating your FEATURE branch with your parent branch i.e. the latest **release** branch.   
git checkout **AIAB-1234\_MY\_FEATURE**

git pull origin **release**  
If merge conflicts occur during updating your FEATURE branch with latest release or master, then, you can resolve the merge conflicts into your FEATURE BRANCH itself. There is no need to create temporary branch here as your FEATURE BRANCH is ultimately going to be merged into **release** and then **master** branch.

## Instructions for Developers and Testers (UAT phase)

Since, there is no separate QA/UAT and PRE-PROD environment, only those features that have been approved for immediate GO-LIVE should be merged into **RELEASE\_QA** to be deployed into QA/UAT environment.  
Merge **RELEASE\_QA** into **release** after Business gives sign-off on all GO-LIVE items.

Upload deployment checklist in JIRA.  
Upload BRD, PRS, TDD (technical design doc) in sharepoint/confluence/JIRA.

Sample checklist that mentions the impacted items:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| JIRA | CR/CA | Description | Dev | BA | Status | git branch | Impact |
| AIAB-2606 | I55839 | KYH tick issue | Sandesh | Peeyush |  | vcs-vmp-v1: AIAB-2606\_VHC\_TICK\_ISSUE\_FIX  vcs-vsp: AIAB-2606\_VHC\_TICK\_ISSUE | VMP-AU, CMS, VI (activity db entry), Core (assessments, events) |
| AIAB-2649 | AU185 | AFD removal | Ashok, Varun | Byron |  | vcs-vmp-v1: AIAB-2649\_AFD\_REMOVAL | VMP-AU |

Legend:

|  |  |
| --- | --- |
| Status | Description |
| RCB | Release candidate build |
|  | SIT in TEST signed-off. QA in progress. |
|  | QA Sign-off |
|  | QA Reject. Keep item until resolved. If not resolved by QA-end date, remove the item from RCB |
|  | On Hold. Remove item from RCB |
|  | UAT sign off |
|  | UAT reject. Remove from RCB |
|  | Status unknown |

### Release build creation process

The following activity must be performed by Release Co-ordinator in every repository that is going LIVE.

1. Synchronise **master** and **release** branch

git checkout master

git pull origin master

git checkout release

git pull origin release

git merge master

Resolve merge issues, if any.

1. Start your Release Build Creation Process

git checkout RELEASE\_QA

git pull origin RELEASE\_QA

git merge <FEATURE BRANCH>  
Resolve merge issues if any.

git merge <ANOTHER FEATURE BRANCH>

Resolve merge issues if any. And so on…

Deploy the **RELEASE\_QA** branch on QA environment.

1. After the UAT-build has been signed off, merge the **RELEASE\_QA** into **release** into **master.**

git checkout release  
git merge RELEASE\_QA   
git checkout master

git merge RELEASE\_QA

1. [Tag](http://git-scm.com/book/en/v2/Git-Basics-Tagging) the **master** branch with proper version number

## Instructions for Developers and Release Co-ordinators (Hot-fix phase)

If a hot-fix is required, then follow the following process:

Create your hot-fix branch from **master** or **release** branch

git checkout master

git checkout –b AIAB-1028\_COLD\_STORAGE\_HOTFIX

You may fix and create a pull-request to merge it into **RELEASE\_QA** branch**.**

Deploy and test your fix in QA environment. If everything is fine,

1. Tag the **RELEASE\_QA** branch incrementing the hot-fix version.
2. Merge your fix into **master** and tag it incrementing the hotfix version.
3. Deploy your hot-fix to LIVE.
4. Merge your hot-fix to **release, develop, RELEASE\_QA**.

# Further Comments on improving this process

1. On code refactoring or majorly impacting changes: Consider FEATURE A is dependent on FEATURE B. How do we ensure that when FEATURE B get GO-LIVE, automatically, FEATURE A goes along with it?

For long development cycles, say, 50+ BRDs with inter-dependencies, how should this process be modified?

Consider following real case:

1. Consider majorly impacting changes like below: moving au\_rules.xml and sg\_rules.xml.  
   FEATURE X had changes done au\_rules.xml and sg\_rules.xml in April 5.   
   As part of FEATURE Y, these files were moved to a new location in the repo in April 10.  
   FEATURE Z, there were changes again in the same files which merged into **develop** on April 20.

Business wants only FEATURE Y in April–end GO-LIVE.   
It is clear that post-GO-LIVE, for both the FEATURE X and Z, developers have to do re-development.