TFS Branching and Merging

This document describes the approach and implementation of source control management

**Contents**

[Introduction 2](#_Toc6629)

[Basic Branch Plan 2](#_Toc6630)

[Structure 2](#_Toc6631)

[Flow 3](#_Toc6632)

[Branch Merge Criteria 4](#_Toc6633)

[Merging from Dev to Main 4](#_Toc6634)

[Merging from Main to Release 4](#_Toc6635)

[Merging from Release to Main 4](#_Toc6636)

[Merging from Main to Dev 4](#_Toc6637)

[Update Releases 4](#_Toc6638)

[Release Process Summary 5](#_Toc6639)

[Major Releases 5](#_Toc6640)

[Update Release Summary 6](#_Toc6641)

[Hot Fix Releases 6](#_Toc6642)

[Non-Hot-Fix Releases 6](#_Toc6643)

[Action Items 7](#_Toc6644)

# Introduction

This document is intended to guide members of the Development team with respect to building and branching the TFS code base. The objective of this process is multifold:

1. That code releases are identifiable and stable outside of the development environment
2. That code releases can be related to the whole of the TFS history so that feature and bug statuses can be known with respect to any given code release.
3. That code releases can be associated with bug creation and feature requests in order to provide context to developers, the company at large, and possibly to external partner stakeholders.

# Basic Branch Plan

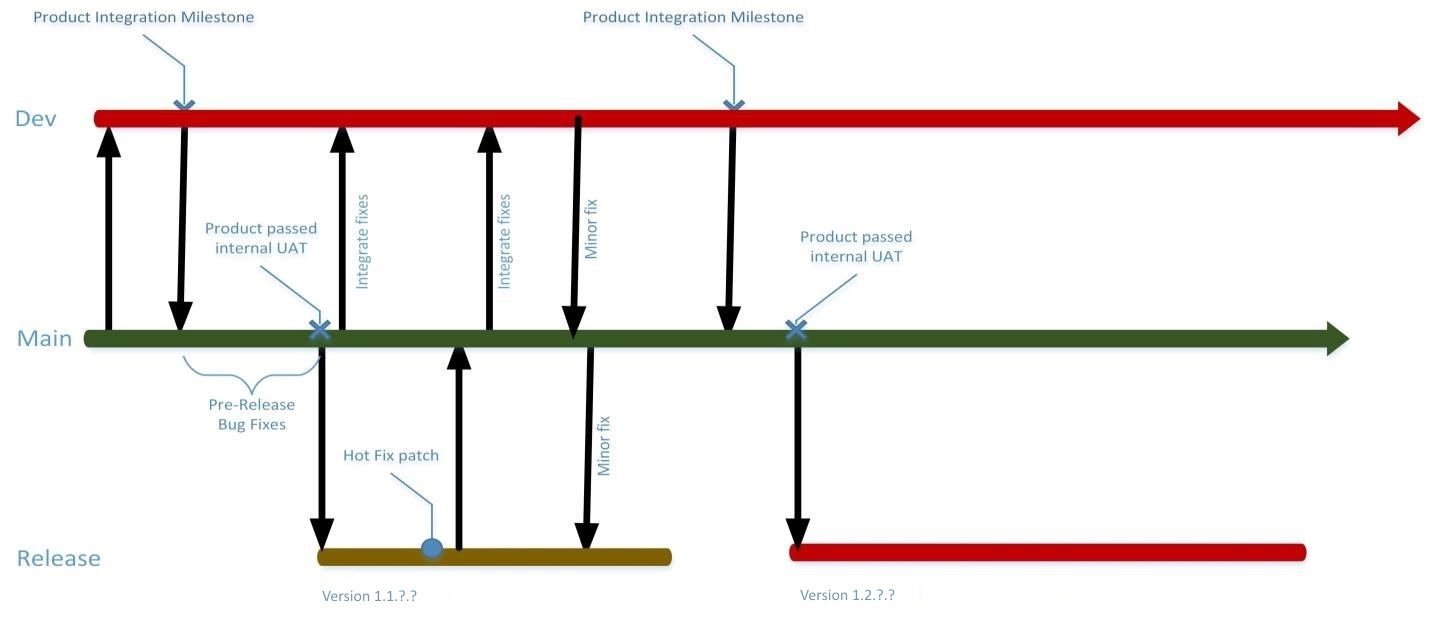
All companies which ship software need a plan for version management. Version management is accomplished via TFS branch/merge functionality. This section describes how that is accomplished.

## Structure

This section presents the plan for concurrent development and deployment. The basic branches are a) The main branch

1. The development (“dev”) branch
2. The release (“release”) branch (one created for each major/minor version combination of the product).

The branches and the actions that relate them to each other are presented in Figure 1 below.



**Figure 1: TFS Branching with Interactions**

## Flow

* Developers submit code to peer review
* Developers check in changes into the Dev branch which employs a *gated check-in* policy.

This means:

o All change sets are converted by TFS into a shelf set o All changes must merge, build and test successfully before they are accepted o TFS provides a mechanism to reconcile the changes checked in with a local, developer TFS workspace.

* Dev branch reverse integrates to the Main branch o This integration indicates that the code base has been released by development o Just like the development branch, merges into the main branch are gated o At this point the development branch is ready to move forward to the next release (no code freeze).
* Quality Assurance and internal User Acceptance Testing (UAT) operate from Main branch
* The main branch now operates as the candidate for the next release. Bug fixing for the release continues on the main branch.
* Once a Release has been approved by all stakeholders, a branch is made to a named release (e.g. Release\_4.7) and any mainline fixes are merged into the development branch. The release branch will contain files specific to that release including release notes and release-specific documentation.
* Bugs found in the release branch are fixed then propagated back to Main and then forward to Dev
* Builds of release branch will be related to unique build numbers which will be reflected in all histories tracking issues and deployments. These build numbers must be unique product-wide.

# Branch Merge Criteria

This section describes the criteria for moving fixes from one branch to another.

## Merging from Dev to Main

Merging from Dev to Main indicates that a deployment candidate exists upon which User Acceptance Testing may begin. Ideally, all changes so merged are associated with TFS bugs or TFS stories so that the changes may be rolled up into the release notes for the product.

Once the code base has been merged to the main branch, the Dev branch is free to continue on, unimpeded into development for the next release. This is the “branch and continue” moment which defines the most immediate future release apart from the latter ones.

## Merging from Main to Release

Merging from main to release follows a successful UAT battery along with all release-specific documentation such as release notes, documentation (online, support, and offline). There may yet be other builds off the release branch as online documentation and the like are refined before deployment of the release to a production service.

## Merging from Release to Main

Hot fixes made on a release branch will need to be reflected in the main branch. No other changes should require this. This synchronization may be by a mechanical reverse merge from the given release branch to the main branch \*or\* the changes may be independently made to the main and/or development branches as deemed prudent.

## Merging from Main to Dev

Merges from main to development will be for those bugs fixed between the “branch and continue” (development  main) moment and the creation of the new release branch.

# Update Releases

In the four-level hierarchy of release numbering (W.X.Y.Z), the components are generally recognized as

* W – the major version number
* X—the minor version number
* Y—the fix number
* Z—the patch (or build) number

Release branches are named for the first two components (e.g. Release\_4.7). Updates to that release branch which elevate the fix number (Y) or the patch (Z) may well require additional mergers between the main and release branches—perhaps even involving the development branch (although, one would expect the more common scenario that the development changes are one in parallel to the main  release merges).

Changing the Y or Z components is an *update release* and the subject of this section.

Changes for an update release may come from the customer or internally. If from the customer and of a sufficient severity, then a “hot fix” will be executed directly on the release branch, that hot fix will be validated internally and then released in short order. That hot fix will then need to be migrated back to the main and development branches.

If the update release comes internally, one would expect that to come from the main or development branch. Such updates are then tested internally using the main branch (merged up from development if necessary) before being merged down to the current release (W.X) .

TFS Labels should be used to mark the preceding update release should it be necessary to roll back or compare against that version.

# Release Process Summary

## Major Releases

These are the steps taken to create a major or minor release of the product:

1. Conclude that all engineering work to complete the release objectives have been reached. This is understood to include preliminary test deployments and automated test runs.
2. Label the point on the Dev branch which will be used for the release.
3. Development may continue with post-release (or next-release) development work at this point.
4. Perform a TFS Merge from the label created in step 2 on the Development branch to the Main branch (Dev  Main)
5. Using the document deployment process, deploy from the Main branch to one or more nonengineering environments for UAT testing and any additional QA activities
6. Fix all necessary bugs in the main branch uncovered by the QA activities.
7. Conclude that the product is ready for release.
8. Create a release branch which establishes the major and minor release numbers (e.g. “4.7.0.0”).
9. Merge into the release branch (Main  Release\_4.w.x)
10. Create TFS builds and deployments for the release branch just built.
11. Conduct the final release QA test suite.
12. Fix any issues discovered in the main branch and merge them down to the release branch.
13. Repeat steps 10 & 11 as necessary.
14. Announce the release ready to ship.
15. Merge any changes made in the main line between steps 5-13 to the development branch ( Main  Dev) as necessary[[1]](#footnote-1).

## Update Release Summary

### Hot Fix Releases

1. Label the release branch.
2. Implement the hot fix in the release branch and verify fix in test deployments
3. Increment the product patch (z) version number
4. Conclude that the hot fix is ready for release and causes no significant regressions
5. Conduct the final release QA test suite.
6. Announce the hot-fix ready to ship.
7. Merge any changes made in the release line to the main line (Release\_4.w.x  Main)
8. Merge those changes to the development branch ( Main  Dev) as necessary.

### Non-Hot-Fix Releases

1. Conclude that all engineering work to complete the release objectives has been reached. This is understood to include preliminary test deployments and automated test runs.
2. Label the point on the Dev branch which will be used for the release.
3. Development may continue with post-release (or next-release) development work at this point.
4. Perform a TFS Merge from the label created in step 2 on the Development branch to the Main branch (Dev  Main)
5. Increment the product fix (y) version number
6. Using the document deployment process, deploy from the Main branch to one or more nonengineering environments for UAT testing and any additional QA activities
7. Fix all necessary bugs in the main branch uncovered by the QA activities.
8. Conclude that the product is ready for release.
9. Merge into the release branch (Main  Release\_4.w.x)
10. Create TFS builds and deployments for the release branch just built.
11. Conduct the final release QA test suite.
12. Fix any issues discovered in the main branch and merge them down to the release branch.
13. Repeat steps 10 & 11 as necessary.
14. Announce the release ready to ship.
15. Merge any changes made in the main line between steps 5-13 to the development branch ( Main  Dev) as necessary.

# Action Items

This section describes the actions which, at the time of this writing, would seem necessary to implement this process:

1. Create Main-branch-based TFS build configuration
2. Create Main-branch-based TFS deploy configuration
3. Create a Release-branch-based TFS build configuration[[2]](#footnote-2)
4. Implement release check-off process for approving releases
5. Implement visible product versions[[3]](#footnote-3)
6. Implement the product version update process

1. Note that some bug fixes may either be inappropriate to be brought forward (due to refactoring, etc.) or may have already been fixed in the development branch. [↑](#footnote-ref-1)
2. This would be cloned and modified for each release branch [↑](#footnote-ref-2)
3. These can be either visible in the UI or via start-up log file entries. [↑](#footnote-ref-3)