Understanding ClearCase UCM Deliver Dependencies

Joseph Bucanelli

July 14, 2008

Introduction

IBM® Rational® ClearCase® UCM (Unified Change Management) reports a dependency when one or more element versions depend on one or more other element versions. These dependencies are often discovered when performing a UCM delivery. UCM dependencies are often considered a defect in the operation of normal UCM functionality when in fact they are not. In actuality dependencies are created on purpose to avoid the possible destruction of data.

This white paper further defines what a dependency is and what causes them, as well as some basic workarounds and their ramifications. It will not conclude status of any defects that may or may not be related to dependency issues.

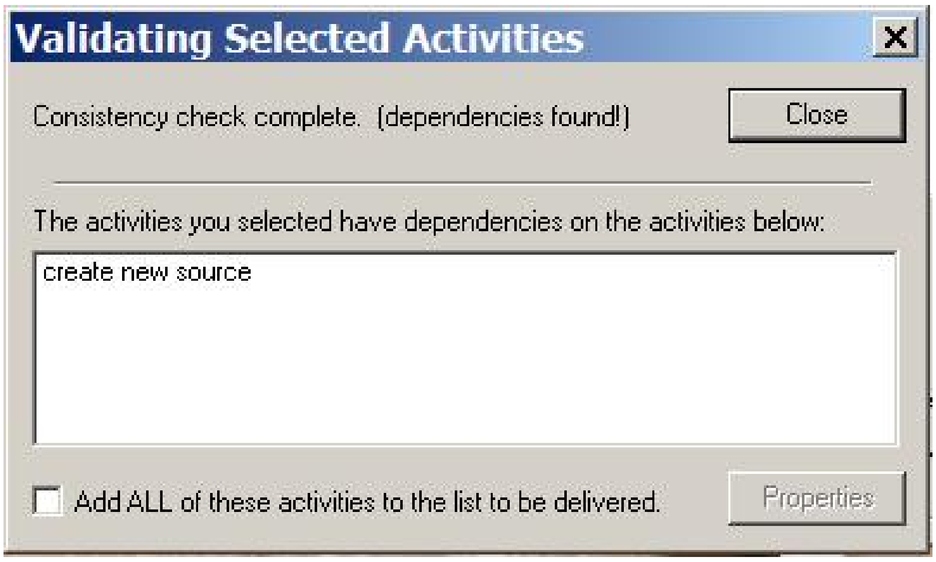
This document is designed to be read by ClearCase UCM Administrators who are responsible for their organizations UCM integration operations.

Before proceeding you should have an understanding of the general UCM concepts covered in the Understanding UCM section of *IBM Rational ClearCase Managing Software Projects*.

Additional References:
About activity dependencies in the deliver operation
How activities are delivered
Dependency relationships in composite baselines of ordinary components
Dependency Relationships in pure composite baselines

Activity Dependencies

Activity dependencies come about when selective delivers are performed, as a result of a consistency check (see Figure 1), which often poses a problem.



Validating Selected Activities

Consistency check complete. (dependencies found!)

The activities you selected have dependencies on the activities below:

create new source

Figure 1 Dependencies Found

All UCM dependencies fall under the scope of an activity dependency. There are two categories that activity dependencies can fall into: change set dependencies and baseline dependencies.

Users who work on the same files simultaneously across different activities in the same stream can create dependencies knowingly. These dependencies can also be forced on users by rebase operations that are performed while work is still pending in a source stream. These are more commonly known as change set dependencies or overlapping change sets.

In multi-tier stream strategies, baselines may be purposely created in development streams (that act as intermediate collection points). But, under the covers, baselines are created by deliveries out of the stream. These dependencies are unfortunate, and misunderstood. This is a known function of UCM and can cause a pain point when encountered. These are more commonly known as **baseline dependencies**.

Change Set Dependencies

Change set dependencies are probably the most common (and most understandable) form of activity dependencies.

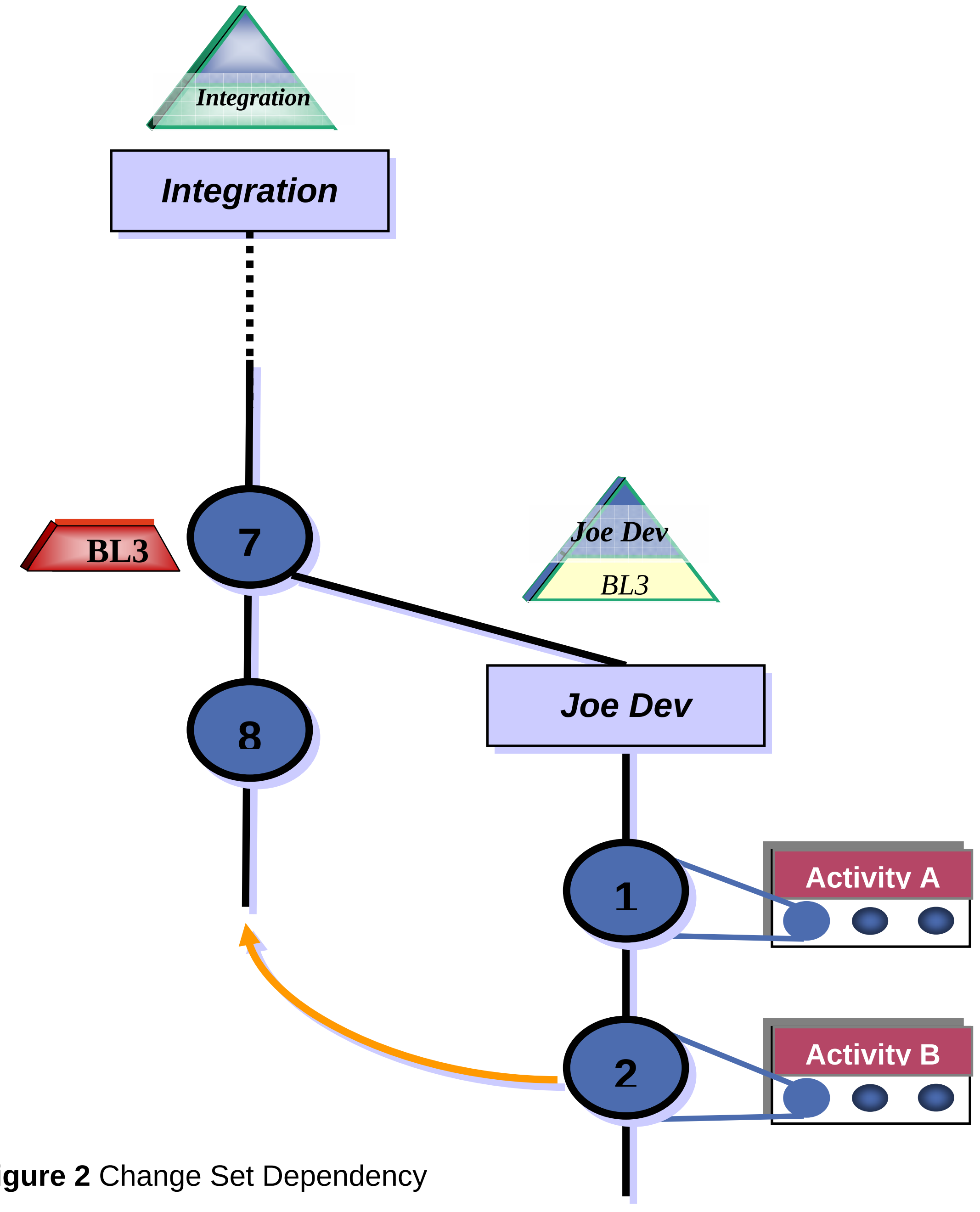


Figure 2 Change Set Dependency

In the case of Figure 2, Activity B cannot be delivered without also including Activity A. This is really no different than a Base-ClearCase findmerge operation from branch to branch. Base ClearCase would also make you include the changes in version 1 when merging the changes from version 2 to the integration branch.

In many cases change set dependencies are self-inflicted and this is very normal. Refer to Figure 3 for an example of a "normal" self-inflicted change set dependency.

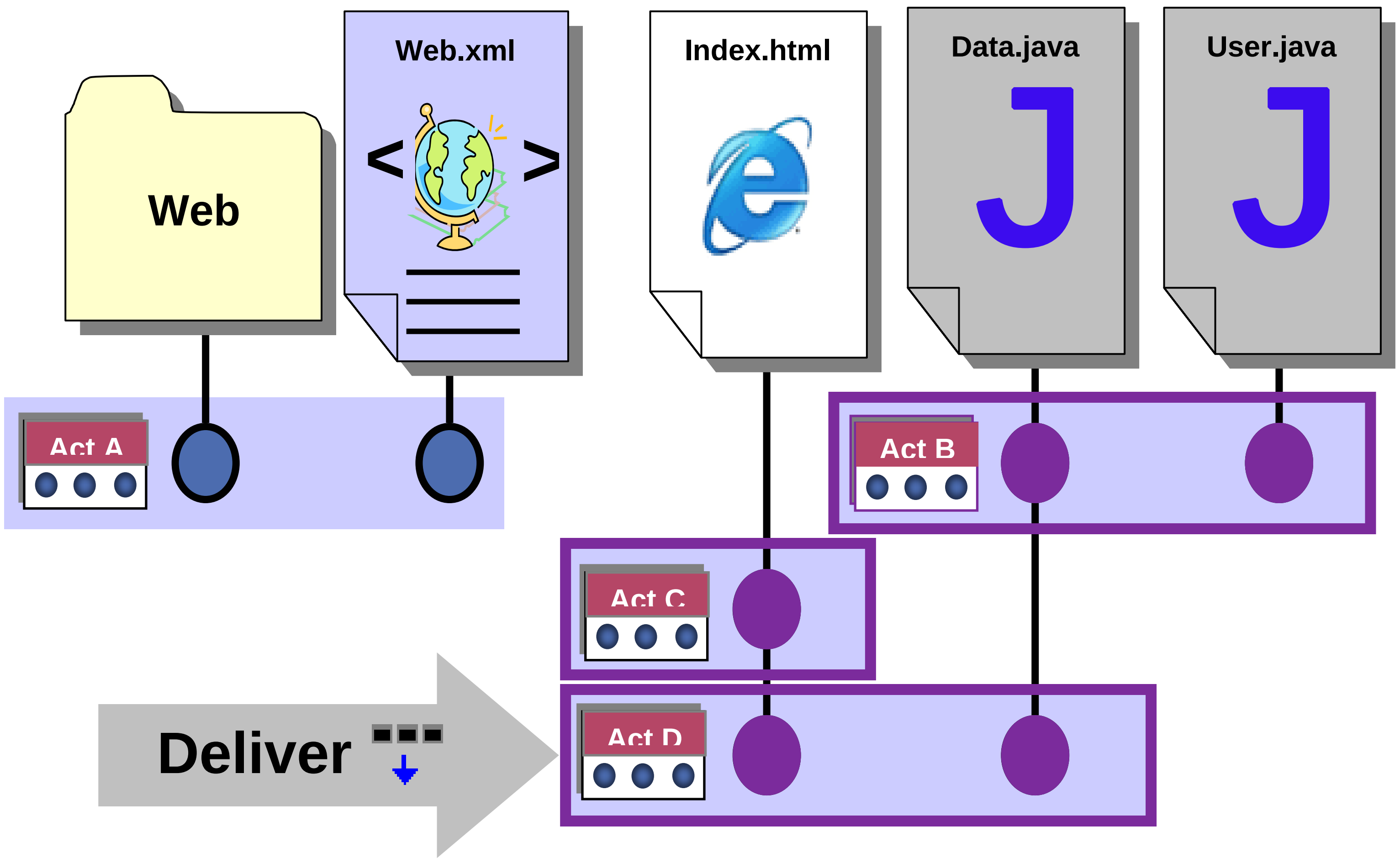


Figure 3 Self-Inflicted Change Set Dependency

In the case of Figure 3 Act D is dependent on Act B as well as Act C, because Act D has a version in its change set that is a successor to a version in Act C and Act B.

In rare cases dependencies can also be forced as the result of a rebase that was performed before all work was finished in the pending source stream. Figure 4 shows an example.

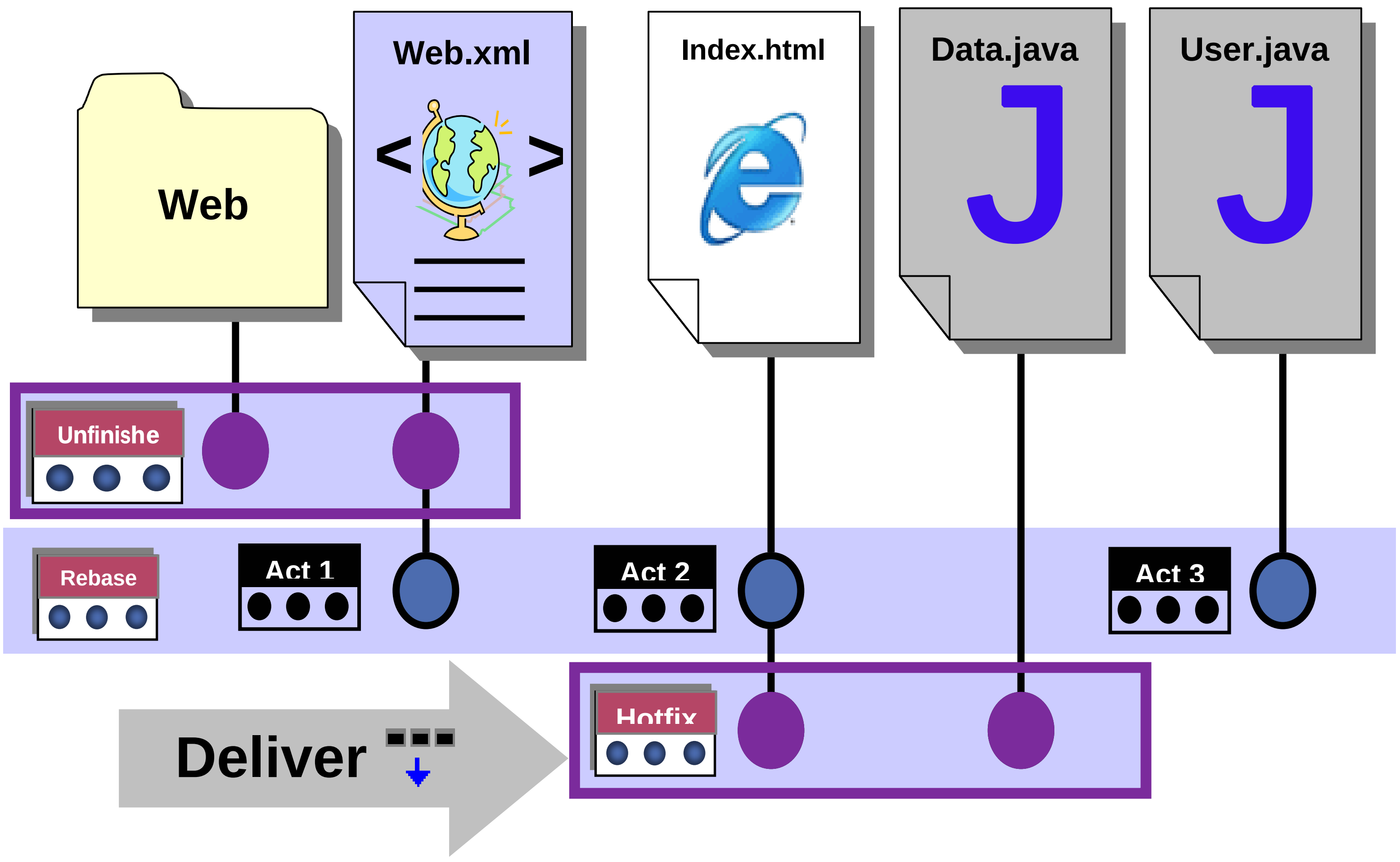


Figure 4 Rebase "Forced" Change Set Dependency

Much like the example in Figure 3 which demonstrated overlapping change sets, activity Unfinished is not ready to be delivered which is holding up the delivery of other activities as a result of a rebase causing change set dependencies. The Hotfix activity is dependent on Act 2 while Act 1 is dependent on Unfinished as a result of having successor versions as change sets. Since Rebase makes Act1, Act 2, and Act 3 dependent on each other, the delivery cannot be performed without including Unfinished.

Baseline Dependencies

The existence of baselines in a source stream will impact which activities can be selectively delivered from that stream. In short, UCM will only allow delivery of selected activities that have not already been included in any baseline on the source stream (not included in any previous deliveries out of that stream).

Sometimes baselines are explicitly created in a source stream. However, many times users may not be aware of such baselines because, in some cases, baselines in a source (development) stream are created under the covers by UCM deliveries and not explicitly by a user. Deliver baselines are unlabeled and don't even show up (by default) in the baseline browser GUI.

Once an activity has been included in a baseline on the source stream, then it MUST be delivered in any subsequent delivery from that source stream.

In some cases, baselines are explicitly made and create a baseline dependency. Sometimes users and/or project managers will intentionally create baselines in a development stream. This isn't common in a relatively flat stream hierarchy, however, this can be very common in a nested stream hierarchy. When activity dependencies occur in these scenarios, they are more understandable to end-users. Refer to Figure 5 for an example of an explicitly created baseline dependency.

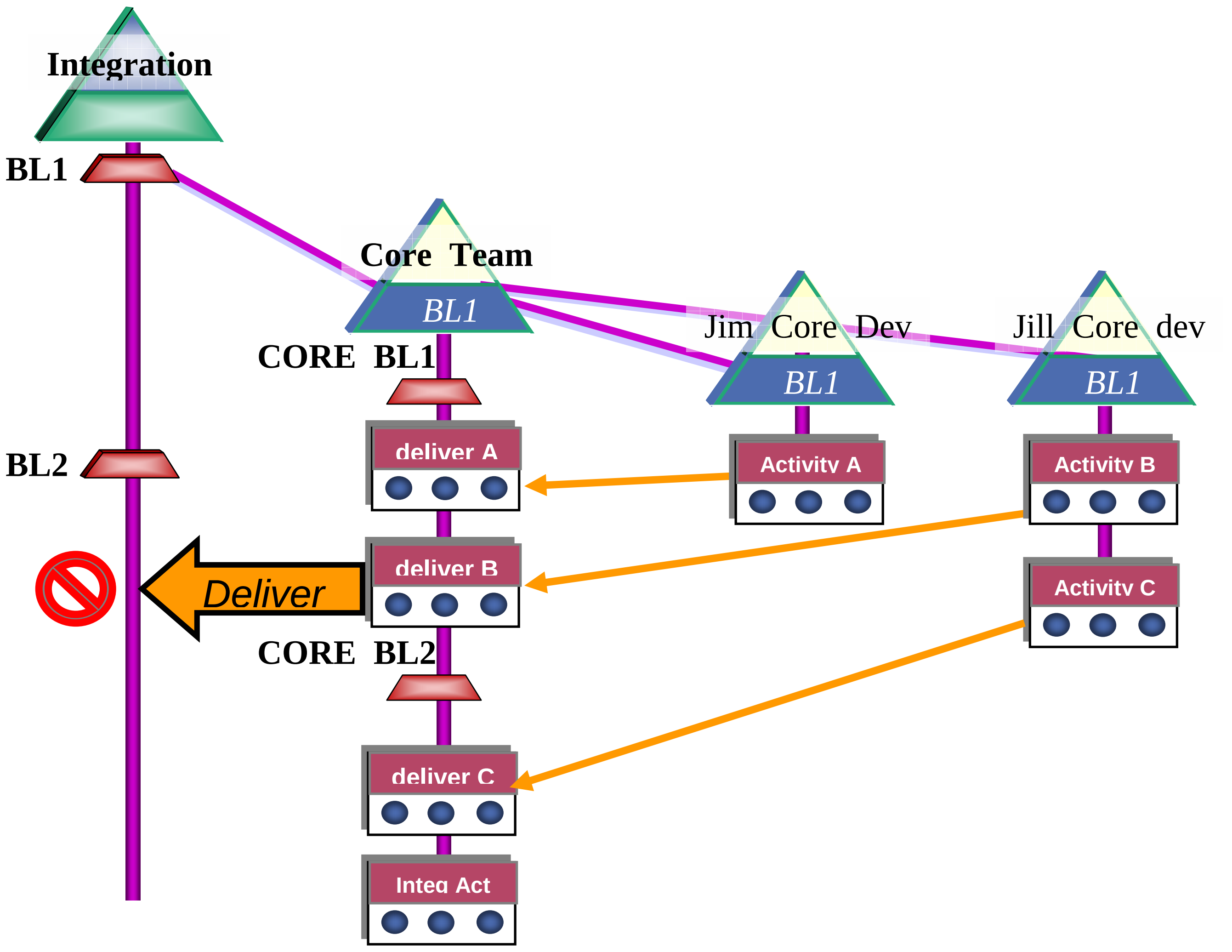


Figure 5 Explicit Baseline Dependency

A baseline is intentionally created in this intermediate stream as a checkpoint (for dev stream rebasing purposes). Only “deliverC” and “Integ Act” (the two activities not already included in a baseline on the Core_Team stream) are eligible for selective delivery from this stream. And even in this case, if you were to try to deliver either deliverC and/or Integ Act to the integration stream, you would still be told that they are dependent upon deliverA and deliverB. If you only tried to deliver deliverB to the integration stream, then UCM would simply tell you that you must also include deliverA (since they are included in the same baseline on the source stream). Once those two activities were delivered to the integration stream, then it would be possible to selectively deliver either deliverC and/or Integ Act to the integration stream (assuming that they are not change set-dependent).

Another baseline dependency is where someone unknowingly caused an under-the-covers baseline creation, more commonly known as a "deliver baseline" dependency. Refer to Figure 6.

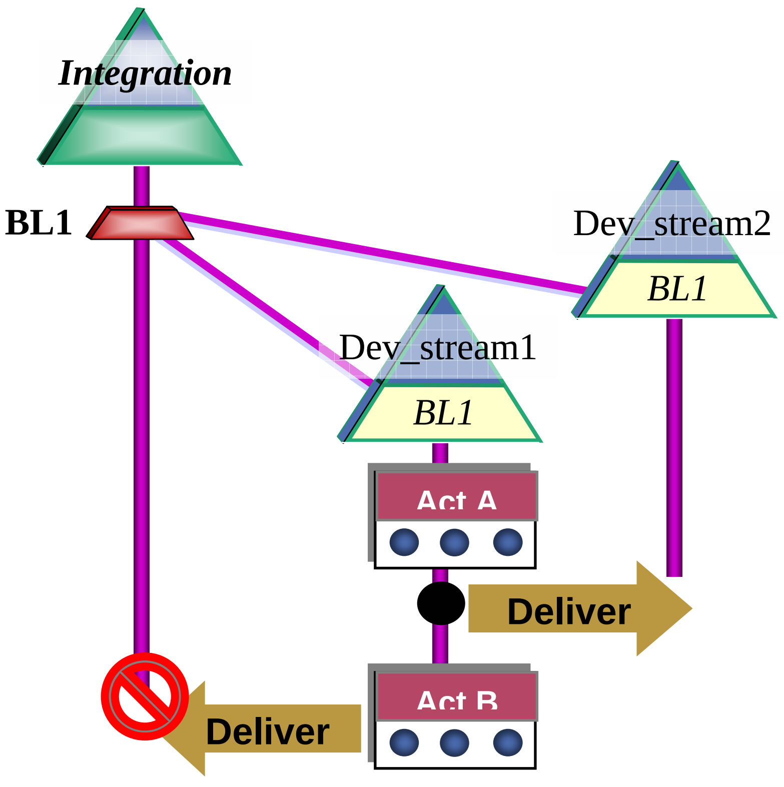


Figure 6 Deliver Baseline Dependencies

In this scenario two sibling development streams exist within a UCM project and Developer 1 works on several activities in the context of Dev_stream1. Developer 1 delivers only a selected activity (Activity A) to Dev_stream2 as an alternate-target deliver. Developer 1 now tries to deliver a different selected activity (Activity B) to the project's integration stream as a default-target deliver.

UCM complains that Activity B has dependencies on Activity A. Developer 1 must include activity A when delivering to the integration stream (because Activity A is now in a baseline). This behavior is expected and occurs even though activity A and activity B have NO CHANGE SET DEPENDENCY!

The existence of this (unlabeled) deliver baseline forces the inclusion of Activity A for any subsequent deliveries out of Dev_stream1.

In this particular case, *any* subsequent delivery out of Dev_stream1 to *any* target must minimally contain Activity A. In other words, only activities that have not been included in any baselines on the source stream are eligible for selected activity delivery. The first delivery of Activity A to the second developer's stream has created a case whereby Activity A must now be included in any subsequent delivery from Dev_stream1 to any other target because it is now included in a deliver baseline in the source stream. Also, after the completion of the second delivery (of both A and B to the project's integration stream), now *any* subsequent deliveries out of Dev_stream1 to any other target stream must include both A and B (since both are now included in baselines on the source stream).

Solving the Mysteries of Activity Dependencies

An obvious solution would be to deliver everything or minimally the selected activity and its dependents (not ideal if that's not what you want to do). Another alternative might be to work in a one-activity-per-stream model. This will always avoid this particular problem by isolating work on each activity in its own stream, although this model introduces overhead and potentially more complexity.

References

Credits

Author: Joseph Bucanelli

Special Thanks: Peter Klenk, Shirley Hui, Yuhong Yin, Johnathan Aibel, Ying Ma, Kent Seith, Ryan Sappenfield, Jim Tykal, Dave Bellagio, Ralph Capasso.