# Branch / Merge with EA V13

Anyone who has done code development in a team will be familiar with the idea of branching and merging. It's a way to allow programmers to collaborate on a complicated set of code, without falling-over each other. Someone takes a copy ("branch") of some code, modifies it, and merges it with the original copy when they are finished. Good code management systems help to see where their changes and those of others – who may have changed the same code - have caused problems, which is why branch/merge is how almost all coding teams work.

This is only important to EA modellers because it's one of the most common requests we see from new EA users: – ***why can't I do branch/merge with EA?***

Up to now, the answer has been '***because it's hard***', and indeed a **full** branch/merge solution for EA would be way, WAY more complicated than one just for code, and therefore probably un-usable.

**But can we get most of the benefits of a code-style branch/merge with EA, without a 100% fool-proof solution?**

With EA v13 we may just have a solution: not a 100% solution, but one which has 10% of the function but which may deliver 90% of the benefits.

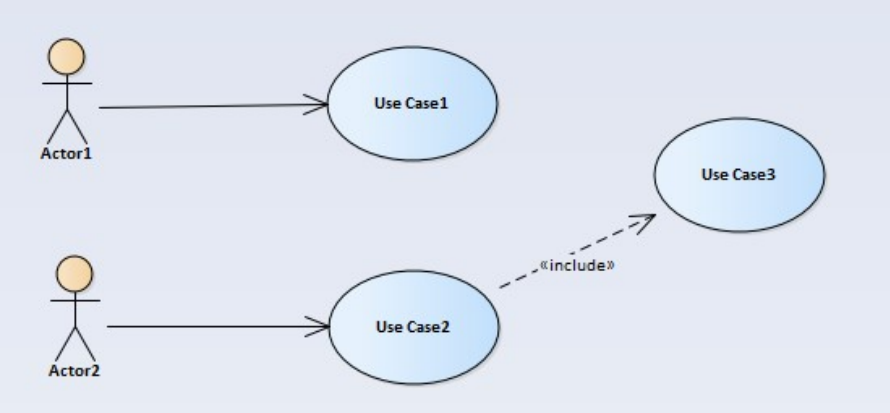
## A real-life scenario

In a recent client engagement (details changed to protect the innocent) we had a small team of modellers (30+) working on a large but quite simply-structured model.

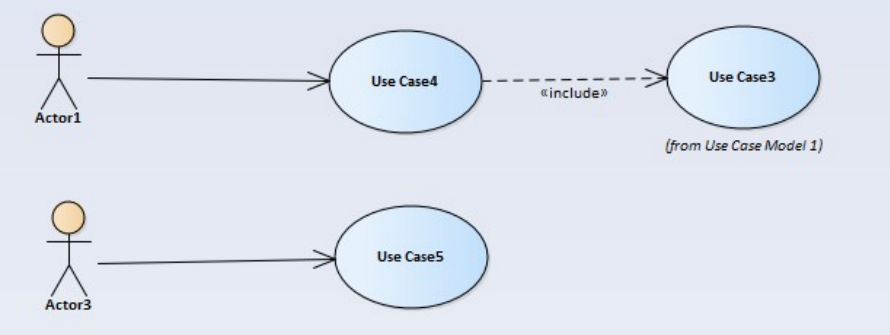
Let's say they were writing Use Cases. What could be simpler?

For readers who don't write use cases, replace 'use case' with your favourite EA element type (Requirement, Business Process, SysML Block – they all have the same issue).

Team #1 created:




Team #2 then produced this:




Let's think about what Team#2 just created:

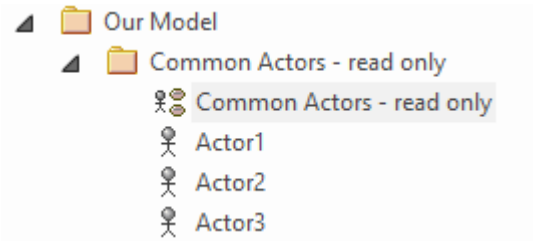
- Team #2 have re-used 'UseCase3' from the team 1 model (you can tell from the namespace below it). What they wanted to say was *'we're using UseCase3 – the exact one which the other team modelled'* – so this is perfect.
- But Team 2 also have an actor called **Actor1**. This doesn't look to have been re-used from Team 1 – (**clue**: no namespace below it). It's just a new Actor which happens to have the same name. So they created a new actor, called it "Actor1", and started modelling.

Now my client spotted this one a mile away. Carrying on like this is going to cause huge problems down the road. They were predicting 80+ actors (real-life, individual roles) and if every team created their own duplicate with the same name, this would rapidly become chaos. They would lose the ability to say 'what Use Cases does **Actor1** get involved with?' But which '**Actor1'** are they talking about?

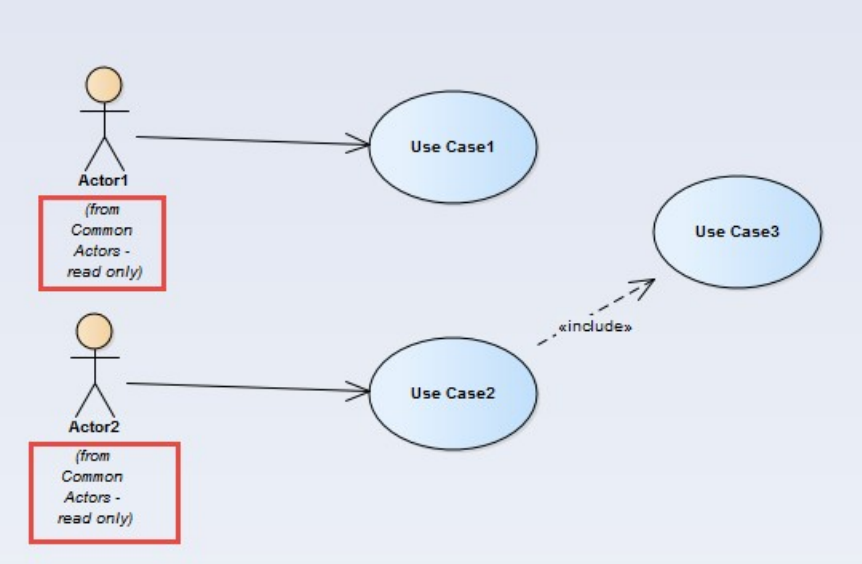
## The “Master Elements” solution

So they did the smart thing, and created a master set of Actors, in a locked part of the model. And told teams to re-use those actors.

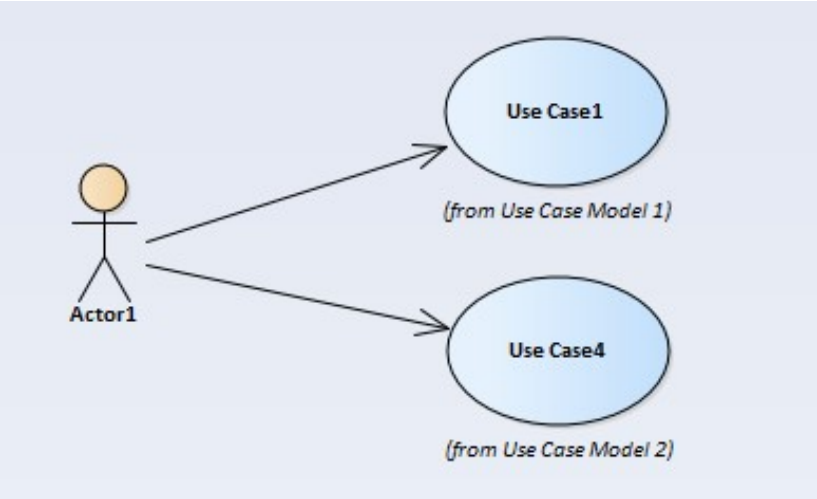
And replace any of their own duplicate versions of those actors with the master ones:




The Team #1 model now looked like this:




That means when we ask which Use Cases Actor1 is involved with (by the EA function ‘**insert related elements**), EA tells us:

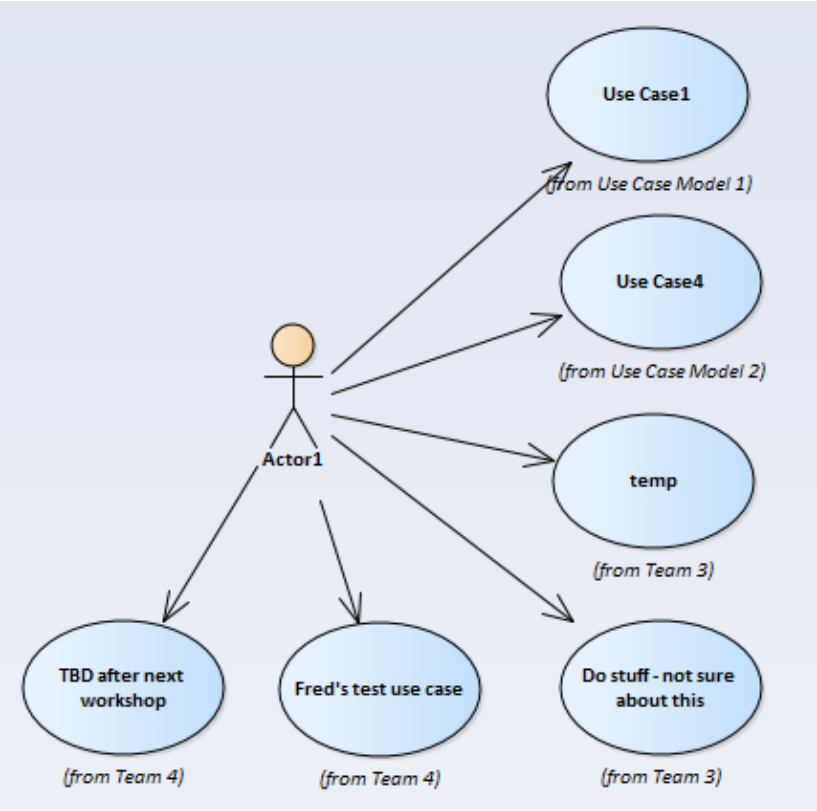



…which is the combination of team#1 and #2s models, as they are both using the same Actor instances from the common set.

Much more useful – and the best argument for 'why EA is 1000% better than Visio'.

So, problem solved? Well not quite.

These diagrams and documents which show 'related elements' are useful for understanding what happening in a project, so they started creating more of them, but started to get things like this:



Now we can say this isn't great use case practice, but it's what happened when they enforced the 'use the common actors' rule. The master actors get connected to **everything** –

- completed use cases,
- partially completed ones,
- … and ones which are just some ideas the modeller is playing with.

If I want to find out the use cases which **really** use Actor1, which have a status which means they are **suitable for me to think about**, then we need something in between the chaos of 'invent your own' and 'always re-use'.

We need an intermediate state of '**using Actor1'**: something like – '*I'm **intending** to use Actor1, but I'm not ready to tell the world what I'm doing*'.

In the case of this customer – using EA12 – they had to create duplicate Actors with the same name, then, once they were happy, manually merge their 'temporary' and 'reusable' Actor1s. An alternative would be to give each Use Case a status, but this means we can't use 'insert related elements' quite as simply.

So not a great solution.

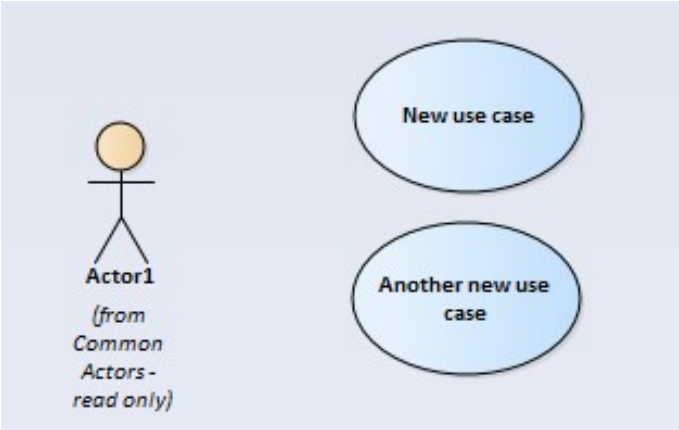
## Then along came v13

EA V13 has introduced the idea of 'time-aware EA'. Looking past the rather grand title, it does provide a small amount of function which should make this problem a lot simpler.

So now, here is the same story, using EA V13.

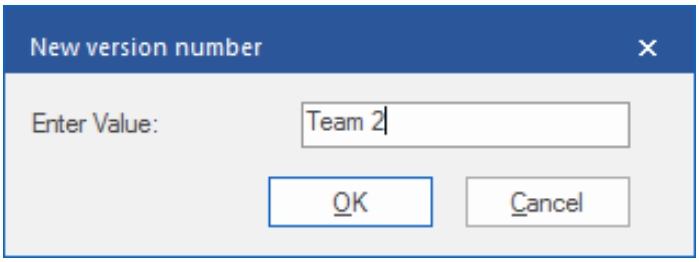
1. The first team creates their model as before, and, as today, adds their new Actors into the 'shared actors' - having first checked with the model manager that they haven't been used before.

2. The next team wants to use Actor1, but they aren't ready to use the 'real' one: they will only do that once they have passed some future milestone. So they drop the shared 'Actor1' onto their diagram, then create a clone of it:

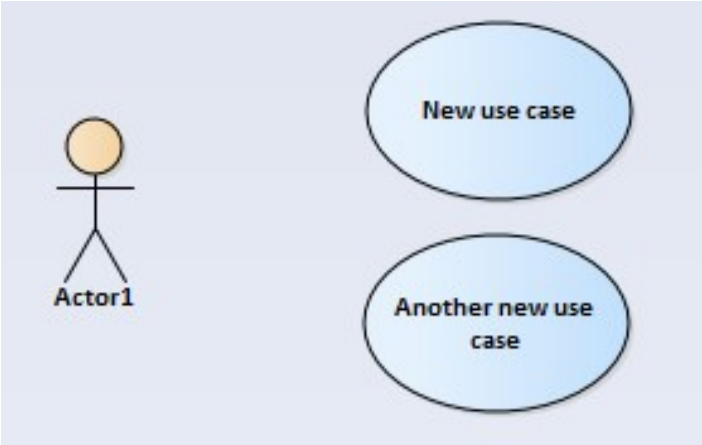



3. Right-click on the actor, and select 'Clone element as new version'. Note this is EA v13 function, not available in previous versions.

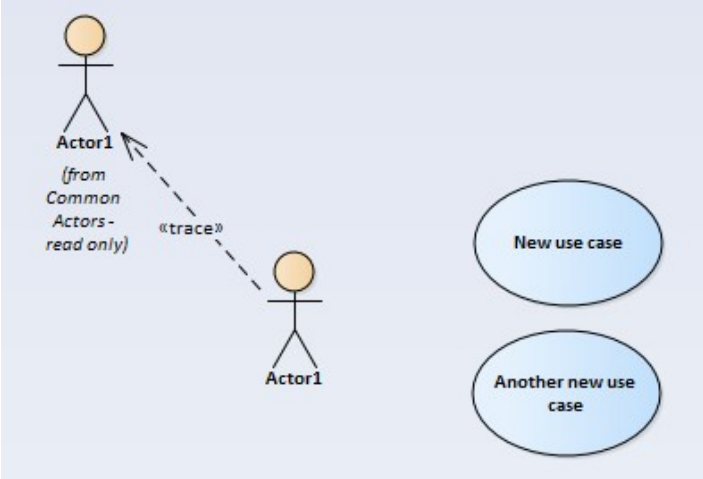
   Using numeric versions doesn't seem useful here, as EA doesn't check for duplicate version numbers, so we suggest using the name of the team, or some other string which lets people identify why this clone is being created:



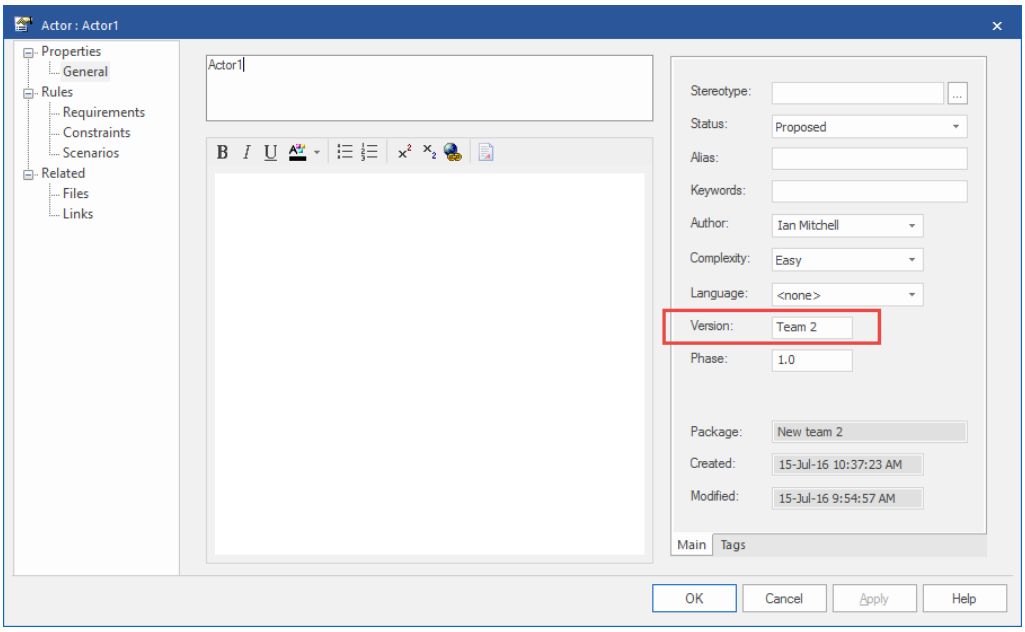

4. This diagram now looks like:




5. To see the magic which EA has done in the background, try dropping the original Actor1 back onto the diagram:




EA has created a new element, with all the same characteristics as the original, but with two differences:
- it has a ‘version’ set to ‘Team 2’:




And has a <<trace>>connector back to the original Actor.

The nature of this connection is worth explaining:

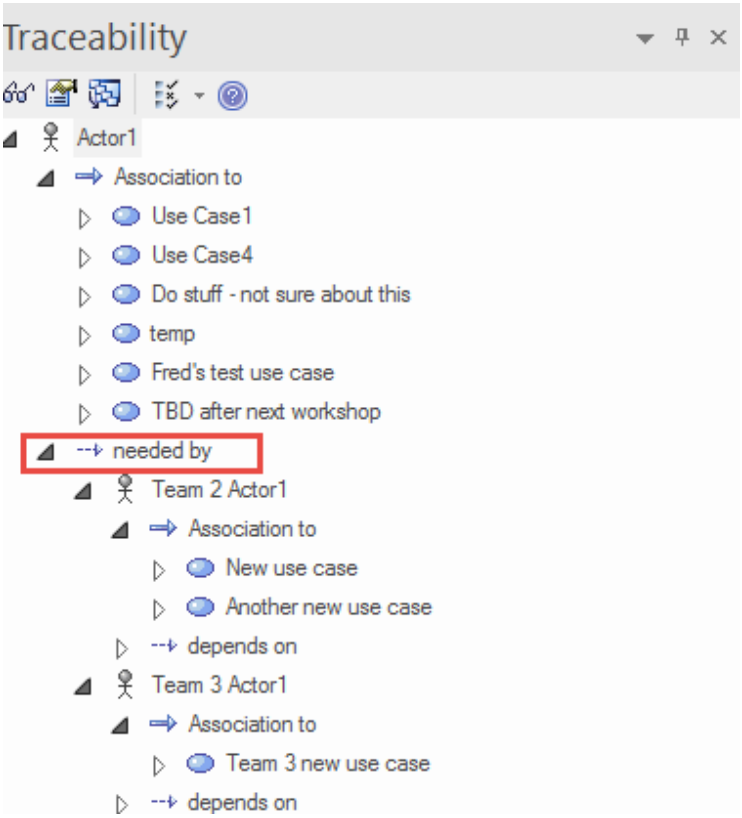
- **In an EA diagram**, it's a dotted-line ("*dependency*") with a stereotype of *<<trace>>*
- In the **EA traceability view**, the connection is called *"depends on"*
- In the **EA database**, and in eaDocX which use the **EA API**, the connector type is *<<trace>>Abstraction*

This is possibly 100% UML compliant, buts it's definitely 99% complicated.

6. With this simple technique, we can now have the best of both worlds:
    a. Team 2 can carry on modelling, adding whatever crazy stuff they need to add to their **cloned Actor1**, and 'real' **Actor1** users won't see it
    UNLESS

b. Those users choose to follow the <<trace>> connector, and find whatever use cases Team 2 are working on.

This is (fairly) easy using EITHER the EA traceability view:




You just need to know that the <<trace>> connector which EA created when we cloned the Actor appears as a 'needed by' relationship in the traceability window. Here, there are two teams which have create clones of Actor1.

OR

c. Tools like eaDocX[1] are OK to print ‘multi-hop relationships’ like this, in lots of different ways. For example, we can document the ‘master’ element and its related elements, then also its clones, and their related elements.

1 eaDocX® is the approved Sparx Systems Extension providing integration with Microsoft Word and Excel, online publishing and collaboration capabilities. For more information visit http://www.eadocx.com

# 1 Actor1

**Details**:

This is the definitive description of this actor.

## 1.1 Accepted use cases for this actor

| Use Case | Ref | Notes |
|---|---|---|
| Use Case1 | UC001 | This use case has been approved, so can be seen by others |
| Use Case4 | | So has this |

## 1.2 Work-in-progress clones of this Actor

| Actor | Description |
|---|---|
| Team 2 Actor1 | This is team 2's clone of Actor1, which will be merged later on when the use cases have been approved. |
| Team 3 Actor1 | This is team 3's clone of Actor1. |

## 1.3 Work-in-progress use cases

| Use Case | Description |
|---|---|
| New use case | Just getting started on this use case |

## Note to Sparx Systems: Next Steps?

Now that we have this starting point, I'd like to suggest that we need some improvements to the EA UI which will allow us to go further:

- **Implement a simple 'merge' function** to move all the connectors from a back-level element to a new one. Saves me having to drop old + new onto the same diagram, with ALL their connected elements, and move the connectors one at a time from old to new.
- **Better diagram filters** to show/hide different versions more simply
- **Associating the current 'version' text field with something more interesting** – another EA element e.g. an Archimate Plateau, or some other project-related 'thing', so that we can find out 'all things associated with version X.Y' without having to start writing SQL
- Native EA support for the function that eaDocX lets us do already (as shown above): '***show the clones for this element***', '***show me what's related to those clones***'. These are now important functions, and shouldn't need me to create a document to see them easily.

## Conclusion

The V13 'time aware' function is limited, but it is a start, and it does show the way to achieving what EA users have wanted for a long time – the ability to have multiple 'versions' of the same thing in a model **at the same time**.

This has many more implications for EA users, as it opens-up new ways of using EA. We can now allow teams to explore new ideas without messing-up the model for other uses, and keep track of those explorations using standard EA techniques.

_____##_____

**About the author**

Ian Mitchell is a Business Analyst by trade, and designer of the eaDocX and Model Expert Extensions for Enterprise Architect. He's been an EA user since v3.5, and in the IT industry in all kinds of roles for 30+ years. He also writes the Artful Modeller blog.