The goal of the distributed Agile workflow is to increase the tempo of software releases. The 10-20% planning overhead of Scrum is simply too much when you're striving for daily incremental deliveries.

But writing the software quickly is only part of the challenge. In order to make sure that each incremental delivery is working correctly, continuous integration (CI) is often used to build, test, and deploy each change.

CI is particularly important at scale, when a manual review-test-deploy for each change just isn’t feasible. If you are working with hundreds of scrum teams on a complex product, automation is a necessary part of the equation.

But building out enterprise CI isn’t easy. Two of the challenges faced by release managers are making sure that CI is building the right set of content for each variation of the product and handling the sheer load that CI will place on the version management system.

In some sense CI is just another ‘user’ of the data, like an individual developer. But while it may be uncommon to have development teams actively working on each variation of a product, the CI system usually is building, testing, and deploying all the active variations. If you have tens or hundreds of product variations, you’ll want to make sure that these configurations are easily stored and used without maintaining a large number of distributed repositories built using subtrees.

Providing the correct view, as we noted in an earlier paper, “Code Reuse and Refactoring: Going Agile on Complex Products,” is straightforward for an enterprise version management system like Perforce (but not so simple in Git). That’s why using Perforce Software Version Management and Perforce Git Fusion as the backbone of your Git development teams will greatly simplify the amount of maintenance work your CI system requires.

Consider a component that is consumed by two main products, each of which may have several customer-specific variations.

A bug fix in the COM component must be pushed to each consumer of that component: the two products and their variations (see Figure 1). Then the CI process can build each consumer to make sure that the bug fix has not introduced a regression.

Figure 1: Simple component model
Supporting the Build Farm

As the matrix of components, products, and variations increases, the CI system will increase its activity accordingly. Scaling the CI system itself is one task, but the version management repository must be able to stand up to the increased activity.

What happens if your CI system is overloading a shared repository as it pulls and perhaps pushes data? If you’re using Git, you could keep adding more shared repositories into the mix to support the additional load. But that means you’re adding more repositories that need to be configured and maintained – and you’ll need tooling that keeps them in sync automatically.

Perforce offers a more nuanced way to support the needs of your CI tools as your deployments grow. As Table 1 and Figure 2 show, supporting proxy and replica technologies can be deployed in any configuration depending on your needs.

Table 1: CI Support Options

<table>
<thead>
<tr>
<th>Key Build Characteristics</th>
<th>Tools</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most of the cost is data transfer</td>
<td>Proxy Service</td>
<td>75% improvement in data transfer time</td>
</tr>
<tr>
<td>Cost is data transfer plus read-only operations</td>
<td>Replication Service</td>
<td>95% improvement in data transfer time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>88% improvement in release note reporting time</td>
</tr>
<tr>
<td>Cost is data transfer plus read-only operations, and build writes to the shared service</td>
<td>Replication Service</td>
<td>Purely local read-only activity plus seamless write operations</td>
</tr>
</tbody>
</table>
Figure 2: Flexible deployment