



# Spot the difference

*How software configuration management can help track code variations. By Dave Robertson.*

**W**hat do virus writers and commercial software producers have in common? Not much, at first glance. But, in reality they share a preoccupation with rewriting and extending code into many different versions. When a virus writer produces a piece of malware, researchers find many variants appearing in a very short time.

Luckily, virus writers are notoriously bad at managing those variants. Code is often duplicated and program structure becomes unnecessarily complex. Before long, inexperienced coders 'enhancing' other people's viruses produce nightmarish collections of spaghetti code that are inefficient, inelegant and bug prone.

Commercial software developers must also manage many variants. Commercial pressures force developers to evolve their ideas to solve slightly different problems and to meet varying business requirements. Different versions may be needed for different operating systems, cpu types, local languages and applications. The problems they face as they try to document and manage many variants are similar but, unlike their misguided counterparts, commercial developers have to make their software work properly. They must apply discipline and rigour to the problem.

Where does variant management start? Finding a baseline – an element of the code base common to all variants – is the first step. This also lets developers infer



the variant specific parts of the code. To do this, development teams need a good software configuration management (SCM) tool, but these are not created equal. Various SCM tools take different approaches to the problem and solve slightly different challenges.



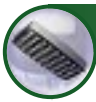
## Managing variation

An SCM tool with good variant management credentials will let developers track those variants without breaking step; it should manage the variant code smoothly without getting in the way of production.

A good variant management tool also needs to store more than lines of C++ code. Its repository may encompass everything from test scripts to product documentation and marketing material. All those assets are subject to change in a similar way to the source code and their variants must be managed in the same methodical way. It must be easy for the team to match variants of different assets together to produce the finished whole.

For that to happen, the SCM tool must allow for the involvement of multiple stakeholders. If the repository is designed to hold more than just code, then it must be usable by the different types of people responsible for those assets. Marketing executives and support personnel may think about their content in an entirely different way to the software developers producing the code.

It is therefore important to choose an SCM tool that supports multiple views, enabling the team responsible for a product's creation to work together. This will reduce the management and administrative overhead and will help to smooth and simplify processes within the team.



*“It must be easy ... to match variants of different assets together to produce the finished whole.”* **Dave Robertson**, *Perforce Software*

This ability to draw together and support multiple stakeholders complements another, related, capability: remote collaboration. With development work often outsourced to other parts of the world, it is important for SCM tools to facilitate work between physically separate developers.

When all team members have access to the same network, they should be able to work against the same core repository. Because they won't be able to shout at each other across the office, it is important for SCM software to provide them with real time snapshots of variant and file status. Is a file checked out? If so, who is working on it and when did they begin?

The role of the SCM tool as an invisible facilitator becomes increasingly important as development teams face increasing pressure to change. Changes in the competitive landscape, alterations to corporate strategy, outsourcing agreements – even mergers – can all have a profound effect on software development processes. Teams have to be more agile and SCM tools can impose an unwanted rigidity. Consequently,

they need SCM software that is fast, scalable and easy to use. It must also be standards based, enabling it to integrate with other enterprise systems.

There can often be thousands of variants in a system, ranging from versions of documentation in multiple languages to development releases and bug fixes. To manage all of these effectively, the SCM product must manage its internal repository efficiently. Each time a variant from the main branch of content is created, it will vary from the core files by a small amount. The method the SCM tool uses

to handle this is crucial and there are two approaches from which to choose.

Firstly, it could copy the whole base of content used in that variant into a new part of the repository, duplicating vast quantities of data to create a new, complete code base for the variant which can then be changed without affecting the main code base. The problem here is that each new variant will increase the size of the overall repository dramatically. When thousands of variants are created, that becomes a problem.

The second, more scalable, approach involves copying only the deltas – the differences between the main content and the variant. In many cases, a variant

will only differ from the main content in small ways. Documenting only those differences is more elegant and leads to a smaller data set for each variant in the repository.

### Size isn't everything

Reducing the space taken up in the repository also serves to increase the performance of the SCM tool. However, repository size isn't the only thing that a good variant management system will handle. It must also deal intelligently with developments as they take place over time as variants are frequently and regularly merged with each other to maintain a consistent base of code.

As these merges and updates take place, it is important for the SCM tool to remember what has been merged before. Otherwise, the whole history of code propagation may have to be repeated from scratch each time a code merge takes place, which reduces the system's performance over time.

If these challenges can be solved effectively, software developers may begin using the repository for personal 'throwaway' development branches simply to test ideas informally. This is perhaps the sign of true quality in an SCM implementation. If developers begin using the tool in unexpected ways, automating tasks that were previously manual, then the tool can be said to have succeeded. If they begin to think of innovative uses for it – wanting to use it, rather than being forced to – the development team will have 'bought in' to the product and will integrate it intimately into the development process.

Before that can happen, the tool must be checked for these key characteristics – performance, scalability, platform support, user friendliness and flexibility. Choose the right tool, implement it well and the return on investment will begin. ■

### Author profile:

Dave Robertson is director of European operations for Perforce Software.

