Software Configuration Management for Embedded Systems Developers

Overview

Embedded systems developers face complex versions of the problems that confront most software developers. Choosing a robust SCM system can help by enabling the management of large and complex file sets, supporting distributed development, and helping to manage intellectual property risk. In fact, the right combination of SCM system and best practices can help any embedded development project progress quickly and smoothly.

## The Challenges of Developing Embedded Systems

Every development organization requires tools that are fast, easy to use, and straightforward to administer. However, developers of embedded systems have an additional set of demands for their SCM system, such as its ability to handle:

- Complex file sets
- Distributed teams
- Management and versioning of intellectual property

The following sections discuss these issues in more detail.

Complex File Sets

Embedded systems are composed of numerous hardware and software components. For example, silicon system-on-chip designs combine schematics, RTL code, GDS data for foundries, software drivers, middleware, and even operating systems and applications. Each system is complex and can contain variations of required components. Furthermore, systems often have variants that are specific to a region or a platform so the product can be sold in markets with differing requirements. The following diagram illustrates how much of a challenge managing these variants can be.
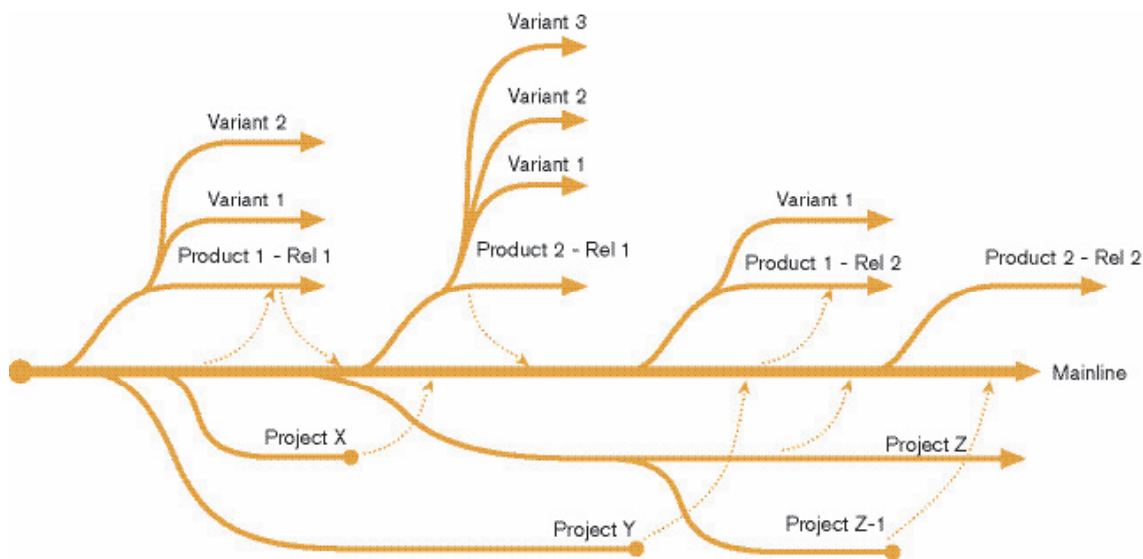
Figure 1: Multiple variants are a hallmark of embedded systems.

Ancillary files, such as test suites, can change as the components being tested are modified. However, full product builds require the correct version of every component. The challenge is to ensure that the correlation between each component and its dependent files is maintained.

Distributed Teams
The components of embedded systems are frequently developed in geographically disparate sites. In some cases, outsourced teams are working on the same component. Collaborative development under these circumstances requires that developers have ready access to each other's work. The challenge is to ensure that development teams working in different sites can stay in sync.

Intellectual Property
Embedded systems often incorporate third-party technologies. In addition to the file management problems described above, third-party developers must be able to update their contributions. These updates must be traceable to ensure that the project contains correct, compatible and stable versions of the vendors' contributions. The final challenge is to enable vendors to check-in their contributions and track changes to these assets.

The Benefits of SCM
The right SCM system is fast and efficient, so developers are not tempted to circumvent it, and it's flexible enough to manage the complexities of developing embedded systems. Here's how a full-featured SCM system, properly deployed, can bring order to a complicated process.

Managing Complex File Sets
To divide components along meaningful lines and build properly, observe the following best practices.

Define component relationships by branching. Use branches to isolate experimental code from production-quality code, or to create a release version of a product. Choose the source from which the files in the branch are drawn, for example, to incorporate third-party technologies into a component.

A full-featured SCM system enables branching and defines the relationships between branches. Branching must be a lightweight operation in order to reduce disk space consumption due to the volume of code and digital asset files which are involved. SCM tools employing the lazy copy technique (see Figure 2) are able to work in this way. Branches containing large numbers of files can be created quickly while minimizing performance and scalability costs.
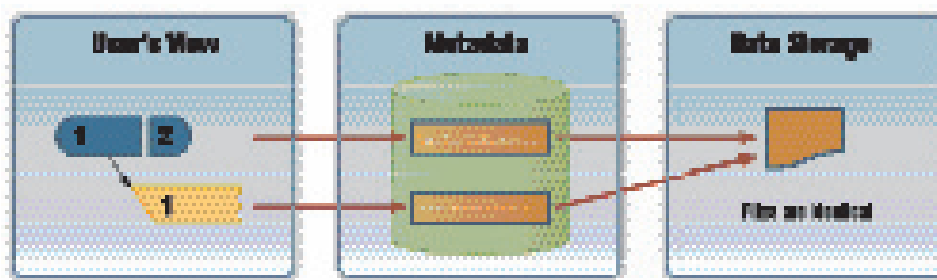

Figure2: *Lazy copying* **allows an SCM system to create variant branches without consuming storage space.**

PERFORCE

Ensure correct builds by using clean workspaces and revision specifiers. Use the SCM system to create a workspace on the build machine, and then populate the workspace with the desired revisions of the components. By ensuring that only the desired files reside in a clean workspace, clean builds are obtained.

A solid SCM system supports meaningful ways to specify revisions, such as date and time, file version number, or label.

Supporting Distributed Development

To best support teams that are geographically disparate but need access to each others' work, consider these methodologies.

Use cross-platform tools and workspaces to provide access for disparate teams. It's rare for different teams to use the same platforms, so an SCM tool must support all platforms in use, without complicating administration. Users must be able to isolate their "in-progress" work in their own local workspaces, as many as required, checking in changes when ready.

Ideally, the SCM tool functions according to users' expectations. For example, Java users are accustomed to applications that have a native look and feel. The most effective collaborative development is achieved when the SCM tool can be used by all contributors to the project, regardless of their role, or the tools, process and platforms that they use (see Figure 3).
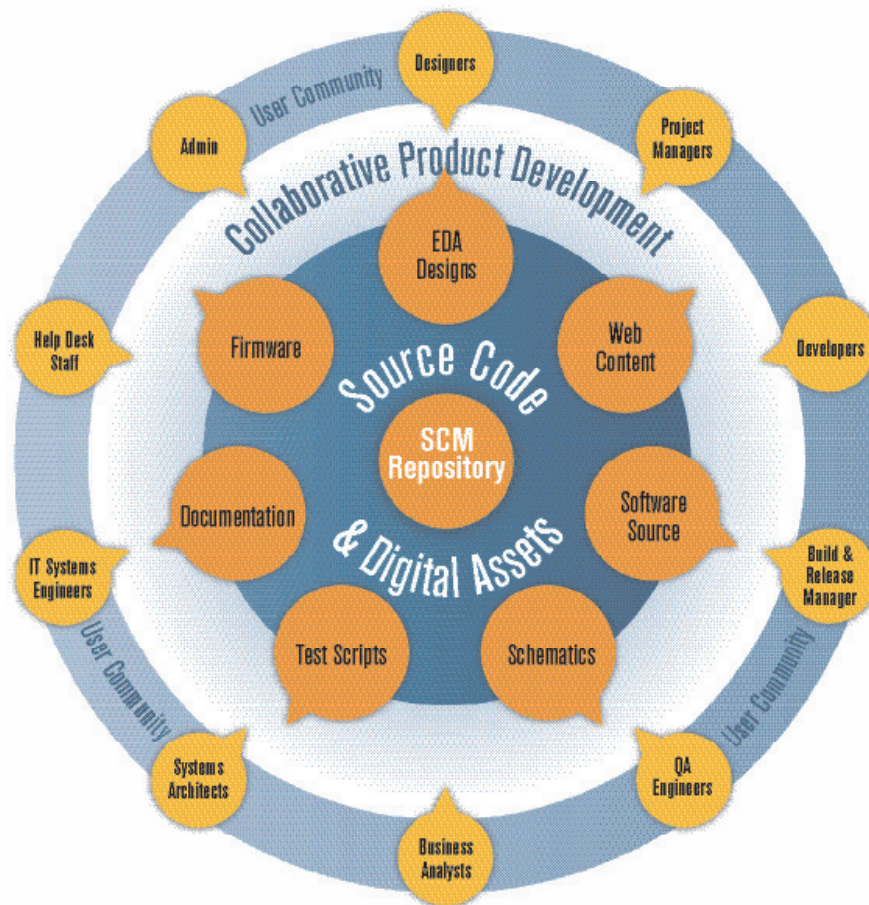
Figure 3: Effective collaboration is achieved when the SCM tool supports all users and the tools and processes in use.

Software Configuration Management for Embedded Systems Developers

Incorporating Intellectual Property

Use codeline policies to insulate components. To prevent inadvertent modifications and to protect proprietary code from being viewed by outside developers, define roles and privileges that specifies who can modify files in each codeline. An SCM tool must support a fine-grained protection mechanism that provides full control over access to files in a corporate repository.

Use branches to support agile development and rapid prototyping. An SCM tool that permits low-cost branching and configurable views provides excellent support for all development methodologies, including Agile and the V model. Low-cost branching enables developers to quickly take snapshots of code lines for reuse, experiment with them, and merge changes back when ready without disturbing ongoing production work. Configurable user views allow a variety of branches to be incorporated into a project. The ability of user views to reference multiple variant branches reduces the need for creating branches to support new projects.

Add *all* project assets to the repository. Many of the digital assets that are used on the project may be in formats other than plain text. Binary files, such as chip schematics or PDFs, are as important for the SCM tool to control as any source code being developed. Plus, new versions of assets provided by a third party may be delivered in frequent intervals as a series of compiled files. An SCM tool supporting lazy copy insures that multiple variants of binary files can be supported and without effecting disk space consumption.

Conclusion

Embedded developers face complex SCM issues on a daily basis. The following best practices can help any project progress quickly and smoothly:

- Define component relationships with branching
- Use clean workspaces for build environments
- Use cross-platform tools and workspaces
- Use codeline policies to insulate components
- Use branches to support various development methodologies
- Add all project assets to the repository

www.perforce.com

PERFORCE

North America
Perforce Software Inc.
2320 Blanding Ave
Alameda, CA 94501
USA
Phone: +1 510.864.7400
info@perforce.com

Europe
Perforce Software UK Ltd.
West Forest Gate
Wellington Road
Wokingham
Berkshire RG40 2AQ
UK
Phone: +44 (0) 845 345 0116
uk@perforce.com

Australia
Perforce Software Pty. Ltd.
Level 10, 100 Walker Street
North Sydney
NSW 2060
AUSTRALIA
Phone: +61 (0)2 8912-4600
au@perforce.com