White Paper

AccuRev

Software Configuration Management Best Practices

Table of Contents

page

Executive Summary 2
Introduction 2
Best Practice 1: Use Change Packages to Integrate with Issue Tracking 2
Best Practice 2: Merge and Integrate Early and Often 3
Best Practice 3: Build and Release Frequently with Continuous Integration 3
Best Practice 4: Create a Structure for Distributed Development 4
Best Practice 5: Use a Promotional-Based Branching Pattern 4
Conclusion 5

MICRO FOCUS

Executive Summary

Software configuration management (SCM) comprises of factors such as compliance, workflow, security, process management, code review, build management and team work. SCM practices define for development teams how software will be developed and eventually released. Teams can maximize their effectiveness by applying best practices that enable rapid delivery. Choosing the right processes and practices are essential for success.

Gartner analyst Sean Kenefick says "Oftentimes the processes organizations establish to support SCM tools are overly complicated or not complicated enough. Even if a process doesn't work, they continue to use it instead of continually looking at their processes and making improvements."

Introduction

In recent years, software development teams have begun to adopt Agile approaches to delivering software. There is often a conflict with SCM practices because of the difficulty of understanding how much structure will be needed. For example, a company's branching model may or may not match their delivery cycle. The company may want frequent product releases, but will use large complex branching structures that take too long to merge code. Understanding how to maximize this structure based on the team's business goals can help a team deliver software faster.

Two-thirds of all software projects fail, according to the Standish Group's CHAOS study. Improper usage of software configuration management (SCM) is largely to blame. After project management, IT users cite configuration management as the process that most needs improvement, according to Anne Hass in her book/article "Configuration Management Principles and Practice."

This paper will focus on five simple SCM best practices that help teams scale and adapt a development process. These practices will promote a team-oriented process that will help produce better software. SCM best practices help engineers focus on the importance of the development process and not the rigid structure and day-to-day activities associated with an SCM system.

Best Practice 1: Use Change Packages to Integrate with Issue Tracking

Stories, bugs, and requirements drive any process, and what is often overlooked is the linkage between these items and the location of the actual code changes. Issue tracking systems (ITS) are excellent ways to create and assign issues to team members, but this is only the first step in the process. When an issue is in development, there are active changes being made against an existing code base in order to complete it. This link is critical to tracking an issue back to the planning tool, and makes it easy for other team members to help complete the issue or even track its status.

With traditional SCM practices and implementations, developers must manually indicate the linkage between the code and the story it is associated with as the code is checked in. At the end of the development cycle, teams are faced with the task of determining which stories are fully completed and which stories are only partially done and need to be re-targeted to the next release. Rooting through comment fields to find completed stories is not only inefficient, but fraught with errors.

One best practice is to use a tighter integration, such as syncing issues from your issue tracking system to your SCM system using tools and scripts. Some commercial systems already offer this type of integration. The industry term for this is called a "change package" or "change set."

A change package links changes in files and directories with a reason for the change and data to give full visibility into history. This makes life easier for developers so they no longer have to remember revision numbers to pull together a release. It's easy to look back in history and see what changes were made and what tasks were associated with them.

The key to making this work is to have a tight integration that makes it easy for developers to associate their code changes with issues. If the integration is tedious and time consuming, it will just be an unnecessary overhead for the team.

Change packages can create a task-driven process out of this linkage, and manage change effectively. As you move issues through the different stages of their lifecycle, it can be easy to follow that code's status during the process. A single issue may go through many different stages, including DEV→QA→UAT→PROD. Having your code match those statuses, at the same time that the issues migrate through the process, is the key to pushing a successful release out the door.

Best Practice 2: Merge and Integrate Early and Often

Merging early and often is the practice of bringing together branches or changes of code frequently to prevent a large and potentially failed merge process. The idea is that if change is committed incrementally, the large "big bang" merge process will not fail.

"Integrating early and often" becomes a challenge based on the size and time of software projects, even if developers are integrating on a frequent basis from their local workspaces. Some development teams even try to avoid parallel development out of fear of merge problems later.

Teams must choose the right SCM approach for fostering merging and integration. It ultimately comes down to several factors:

- **Choosing the right branching and merging pattern**—branching patterns should make it easy for developers and team members to move and integrate code between different branches. It should be easy and straight-forward to know where a change belongs at any given time.
- **Enforcing merge rules and schedules**—developers should be able to create private workspaces and branches to allow them to create builds, releases and tests of code before they push those changes to other team members.
- **Gaining visibility into the merge process**—a straightforward pattern should give development teams the ability to know where to merge to and from, and when to push changes from one branch to another.

Ultimately it's up the team to decide on the frequency in which changes are integrated. There are SCM systems that automatically integrate a team's changes together in real time, this can be a huge benefit for teams that have parallel development needs. Ultimately it's the team's responsibility to have visibility into that process and be responsible for code changes that should be shared with other team members.

Best Practice 3: Build and Release Frequently with Continuous Integration

Taking several days to merge changes from one developer sandbox to another isn't fun or practical. Kent Beck, the creator of eXtreme Programing, developed the concept of Continuous Integration to address this very issue. With Continuous Integration (CI), code is integrated several times a day rather than just every night or once a week. Code doesn't have a chance to get stale in a private workspace.

In addition to integrating code, CI is the process of performing builds very frequently and doing some basic sanity tests on the build to know if it's good. With this type of rapid build cycle, teams can focus on incremental problems immediately and fix the problems instead of having to wait several days.

Software development organizations can incorporate software configuration management best practices when designing and implementing a continuous integration system to further reap its benefits. These best practices include the following:

- **Using an SCM system to store and version all source code**—this includes libraries, components and tests.
- **Utilizing private developer workspaces**—developers should have a place to check in code, test and build before passing it off to the team.
- **Enabling local developer builds**—local build should function the same as the CI builds to avoid unnecessary problems.
- **Frequently updating code in the SCM system**—make it easy for teams to update code with each other.
- **Establishing a staging and isolation hierarchy**—create staging environments and branches for each environment or step of the process.
- **Automating builds at all stages in the hierarchy**—automate the build process with triggers and events.

Continuous integration is becoming commonplace, with most teams implementing some type of automated build system. This practice should occur frequently enough that no intervening window remains between the SCM commit and the build, so that no errors can arise without developers noticing them and correcting them immediately. This process is certainly easier when utilizing an SCM system with atomic commit capabilities.

Best Practice 4: Create a Structure for Distributed Development

Collaborating and sharing code with distributed teams is more complex than ever. Teams routinely perform software development in many locations and sometimes test or perform other tasks in another location. This distribution of teams strains the development process. There are security auditing and integration problems throughout the process.

Development teams must appear to be co-located while utilizing the same process, across teams with lower complexity. This means code integrations with other projects should happen in real time, so teams can give each other feedback immediately. The process must be visible to everyone to ensure they are all on the same page.

Management of a distributed SCM is challenging, even with the recent addition of DVCS (Distributed Version Control Systems). Keeping track of change and managing the system over LAN/WAN boundaries often with non-local teams can be a challenge. It can become more complex if a remote team asks for its own repository or server and becomes decentralized.

Developers can have difficulty understanding and identifying code conflicts between both locations. The SCM tools in place must be able to show real-time changes between sites.

For reliability and manageability purposes it's often best to go with a single repository between sites, even when using a DVCS to manage changes. This ensures that teams will be on the same page and follow the same process regardless of location. DVCS alleviates some of the LAN/WAN barriers, but in some cases decentralizes the process.

Best Practice 5: Use a Promotional-Based Branching Pattern

Overloading a traditional branching pattern with too many projects and releases can become a burden for the team. The goals of a pattern should always be to manage the software team's development process, and make it easy and straightforward to follow all of the things that need to happen in order to release a piece of software. Creating ad-hoc project and developer branches is not a best practice to follow.

Promotional-based branching patterns differ from the traditional branching patterns because of their ability to map to a development process, instead of just a release or project.

Philosophically, promotional branches can be analogous to the different states of the development cycle. Similar to how issue tracking systems (ITS) move issues through each state, code can belong in different states also. Code might move through different statuses while making its way to production. Teams will start out in development and move the code to QA, UAT, and eventually to production. Along the way, there are processes and policies in place that the code must align with before it moves to each stage.

Promotional patterns are integration points, where transfers of roles, responsibilities, and code integrations can be managed at each stage of the promotional hierarchy. This allows the code to grow more stable as it moves up the hierarchy.

Each branch is based on the previous branch in the hierarchy, meaning that if the parent branch changes, those changes will push down to the lower levels. This allows for easier and more frequent code integrations. While promotional models are a more natural way for teams to work inside an SCM system, implementing the model could require some changes to the SCM system, either with scripting or other tools to optimize the model:

1. **Integration with issue tracking systems to provide change sets, or "change packages":** Change packages are the union of file and directory history to a particular issue. This will allow for easy movement of code through the stream hierarchy.

2. **Automated merging and integration with parent and child branches:** This must be in place so that when a hierarchy is established, most changes start at the top level but changes will flow down to the lower levels with parallel development happening at the same time.
3. **Visualization to manage where in the process a team is and how the hierarchy is set up:** Map and manage the process so that teams have visibility into what needs to be changed and when those changes should happen.

   Promotional models are a great way for teams to manage a complex development process. It enables development organizations to manage effectively every configuration of the SCM codebase while ensuring that a process is followed.

Conclusion

SCM best practices are an important piece of any software development process. Modern development teams deliver code frequently, and have a need to manage a software development process right in the SCM repository. Using SCM best practices based on the team's development needs is a necessity when enabling teams to deliver rapidly with high quality. These best practices can be used to scale an enterprise SCM rollout, and keep development teams focusing on what's important, not the SCM system.

By implementing AccuRev your company could optimize SCM best practices across your organization.

For more information visit: **www.borland.com/accurev**

Micro Focus
UK Headquarters
United Kingdom
+44 (0) 1635 565200

U.S. Headquarters
Rockville, Maryland
301 838 5000
877 772 4450

Additional contact information and office locations:
www.microfocus.com
www.borland.com