# Top 5 Software Development Process Challenges

## Executive Summary

A process framework is a combination of project management, technical practices, and supporting tools. The tools and practices have a profound effect on the day-to-day life of a developer. These types of practices and tools have either helped or hindered development teams deliver software. Specifically, software configuration management (SCM) systems are like the "hub" of a development team. It's where teams artifact important work, integrate changes, save important ideas, and add features for customers. SCM is the management of change; it's the center of the development universe.

One leading SAS-based software company noted that using the proper tools and processes "streamlined our Agile and hybrid process beautifully, saving us at least a day of merges on the small projects and even more on the larger ones. This equates to approximately $500,000 in savings per year for the Web Services team."


*SCM is a supporting tool that must act as the hub to bring together the entire process.*

The top 5 challenges for organizations using SCM are the following:

- Getting to done (i.e., knowing which issues are in which state)
- The complexity of managing software changes
- Attaining rapid release cycles
- Remote and outsourced teams
- Scaling and code sharing among Agile and non-Agile teams



AccuRev

Understanding how to use process and your tools together is crucial to the success of a development team.

Gartner analyst Sean Kenefick says the use of outdated tools is less of a problem than the processes that surround the tools. “Mostly folks need to improve processes. The tools are strong; they do what they are meant to do for the most part. It’s how you use the tools that’s the real problem,” he says.

This paper focuses on a series of best practies and techniques for development teams looking to improve their software development process.

## Getting to “DONE” within Each Stage of the Development Process

Stories, bugs, and requirements drive any process, and what is often overlooked is the linkage between these items and the location of those actual code changes. Planning tools are an excellent way 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 retargeted to the next release. Rooting through comment fields to find completed stories is not only inefficient, but fraught with errors.

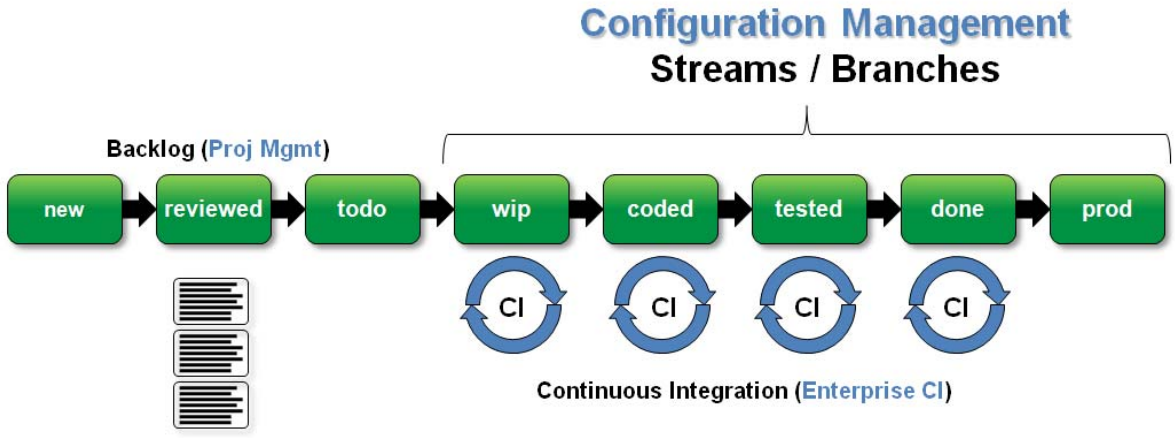
A possible solution is to use a tighter integration, such as synching issues from your issue tracking system (ITS) to your SCM system using tools and scripts. Some commercial systems already offer this type of integration. The industry term for this is called a “change package” or “change set.”

Creating a link between your user stories and your code is the first step. Creating a powerful task-driven process out of this linkage is the key to “getting to done.” As you move issues through the different stages of their life cycle, it is often difficult to follow that code’s status during the process. A single user story may go through many different stages, including DEV->QA->UAT->PROD. Having your code match those statuses at the same time that the user stories migrate through the process is the key to pushing a successful release out the door.

Especially in an Agile environment, the goal may be to have a shippable code increment every two weeks, but unfortunately traditional SCM practices are still designed around single branches for large releases and long and lengthy merge processes. Fast-paced Agile environments expose those limitations as development teams struggle to ship code to customers at the end of iterations.

During that iteration, testers may want to test completed user stories, but need a stable configuration to do so. If you’re using a traditional waterfall branching and merging based SCM tool, you may end up with user stories that are ready for QA and some in DEV. This creates a poor testing environment

and broken builds. As a result, developers often delay committing code so changes can be implemented, tested, and passed through QA before they integrate.




The answer is a promotional-based branching system. This can be scripted as branches in any SCM system. This allows code to be automatically merged between stages as it moves to production. For each stage of the process you can choose “done” issues to move to the next stage. This avoids rework, and the technical debt of trying to cherry pick code into a release.

The key to making this work is automation between the branches. Each branch must be able to inherit changes from its parent branch automatically. By doing this, the flow of information between branches allows for powerful process management, not just better branching and merging.

## The Complexity of Managing Software Changes

With today’s software development methods, teams are distributed and work in parallel, releasing software at different cadences and schedules. Often you will need to take code from one team and merge, integrate, and test those results with another team(s). To do this on a frequent basis means that you now merge your set of code changes almost daily. You still need to keep visibility into which stories are being merged and completed by each team. Trying to do this with a time-consuming merge process can bring progress to a screeching halt.

Traditional SCM practices and implementations require you to solve this multi-team coordination by having everyone work on a single baseline. Having too many teams share a code base slows the process and encourages work in isolation.

This “baseline pollution” should be mitigated by using a development hierarchy. A development hierarchy is a representation of the dependencies between groups, including process steps such as integration, quality assurance, and code reviews. A separate code configuration is used for each stage in the hierarchy. The development hierarchy is a natural extension of private branching.

Using the promotional branch hierarchy increases code stability. As a completed issue is pushed from one stage to the next, the particular change as well as the system as a whole reaches a higher level of maturity.

## Attaining Rapid Release Cycles

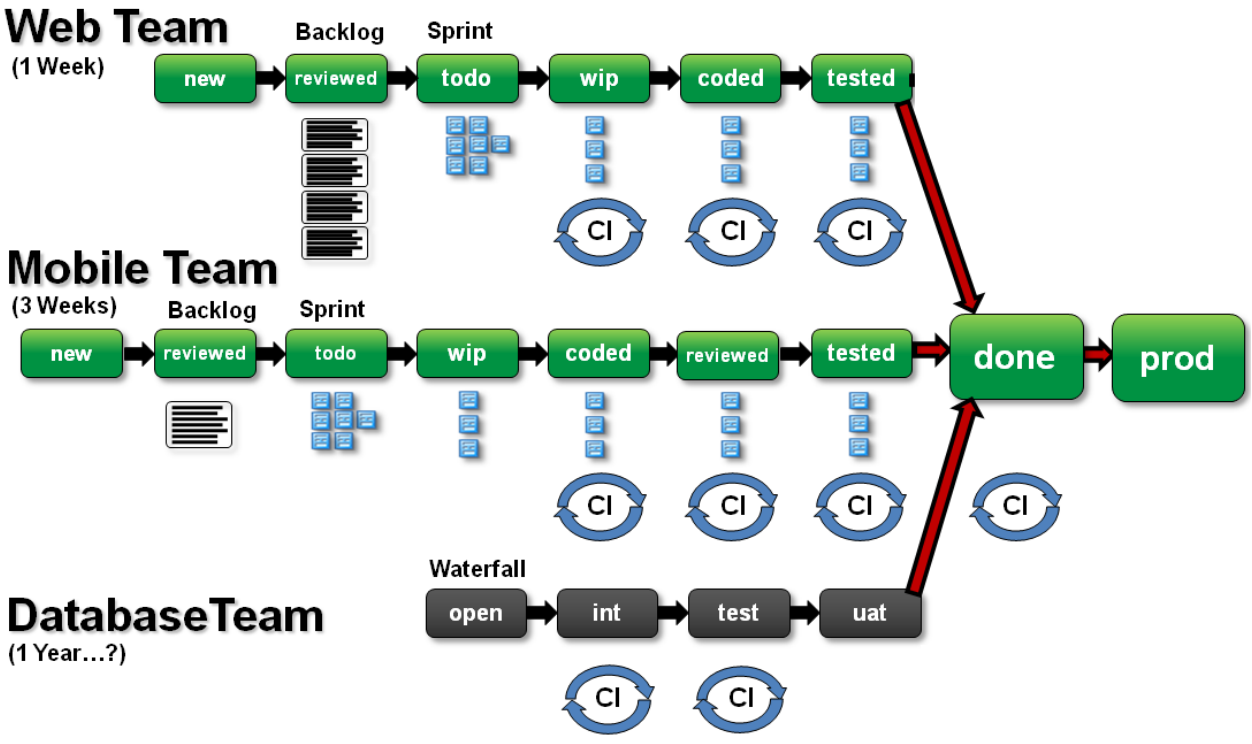
Today's software development world is a lot different than even five years ago. We are asked to deliver more quickly, with greater accuracy and faster time to market.

Our customers are as demanding as ever. If we are slow to meet these demands, competitors will pick up the slack. This means that development teams are now delivering in shorter lifecycles. Instead of delivering software every few months, now they are delivering code every month, week, or in some cases every day.

This can be extremely inaccurate to predict, especially when problems occur. One of the most overlooked problems is the SCM activities and how they fit into the development process.

In any particular release, it's common to take all of the "done" code and merge it to a release branch. There is a set of "un-done" code mixed in with this code base. The problem is identifying the merge and dependencies out of this branch to the release branch and performing the merge. This type of activity is not accounted for in our planning meetings.

In addition, with shorter release cycles, merges that needed to be done once every few months now have to be done every two weeks, or every month. The pain of merging is now persistent throughout the development process.






AccuRev

This means a significant investment in tools and architecture to speed this process. Using the typical branch-per-release mechanism isn't up for this job. Using promotional-based branching and merging can ease this pain, with the right type of automation in place. Many of the merges between teams can happen as soon as each team decides a piece of code is "done" and vice versa so the changes are bi-directional. (Note in image 2, the web team, mobile team, and database team all merge "done" changes when timely and ready for integration with each other, all with the individual schedules and cadences.)

## Co-location Challenge

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

Current best practices tell us that teams should be co-located; meaning each team locally should be able to deliver a segment of work without having to go outside of the team.

These teams should be able to self manage, and have high bandwidth and frequency of communication. They should be staffed with developers, testers, doc writers, and any other member needed to finish an item.

The reality of co-location is that currently we all work in a distributed way. Almost all organizations distribute teams in some manner, whether it be on separate floors, offices, buildings, or countries.

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

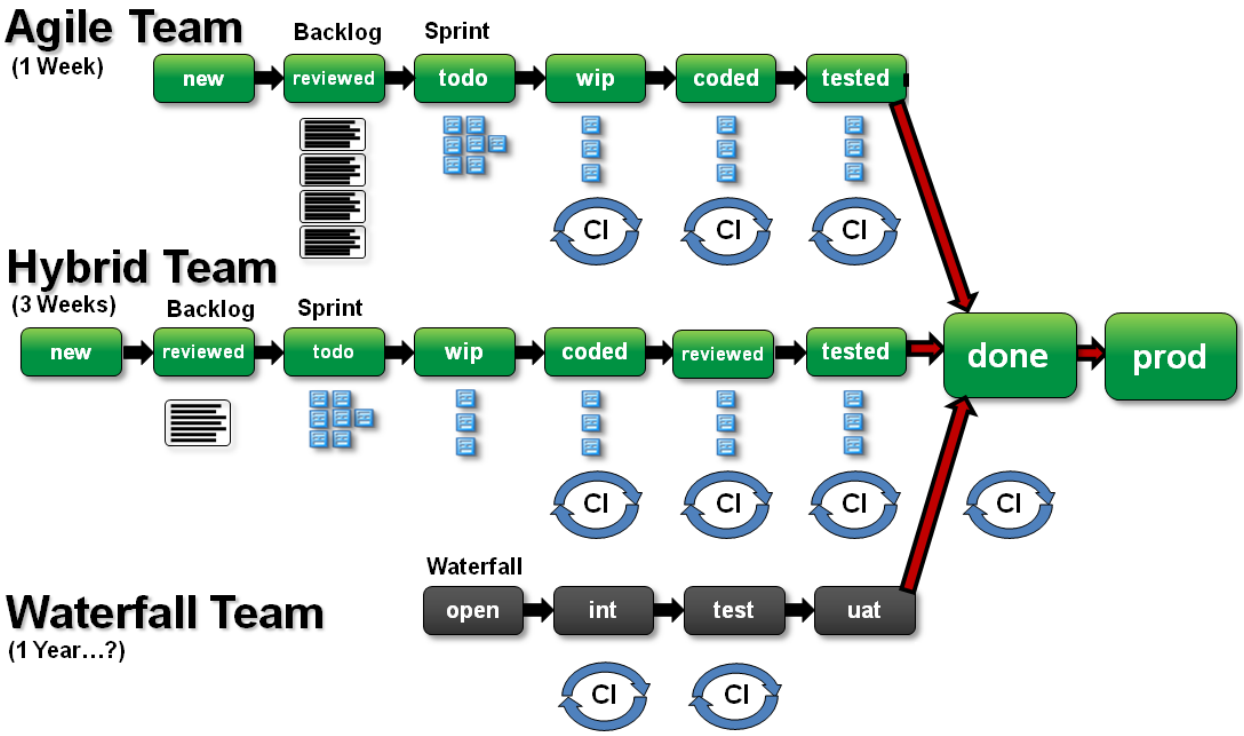
## Scaling and Code Sharing Among Agile and Non-Agile Teams

The reality of mid- and large-size teams is that there are multiple processes within the organization. According to Garter Research, a small percentage of teams are truly Agile today. Rather, these "Agile" teams are using Agile practices, not pure Agile. The fact is that real-world scenarios show teams adopting some Agile practices while not others, meaning a wide variety o f processes and practices will need to be supported. Teams will have to co-exist with each other, sharing code, issues, and visibility into each other's work.

From small Agile pilot teams to fully-scaled environments, there is a variance in how people deliver change. The delivery cycle of change is different for each team, meaning small chunks of code with the potential to release more often.

Legacy SCM techniques are designed for long release cycles. Typically code will be delivered once in a great while, meaning there are long periods of code isolation and an eventual integration point later.

With multiple processes, juggling those release cycles will be difficult. Using a promotional approach gives the benefit of moving “done” code with different cadences with different processes mixed together.




It is important to note that each team delivers code that is “done” when they have determined it’s done. This means that there is not a mix of unfinished work between teams.

Process visibility is very important. Teams need to be able to see this process to understand it. Doing so improves collaboration and allows for teams to deliver quickly and easily.

If we imagine the processes that we have in place today, it may be a combination of scripts, tools, and branches. This cobbled-together solution doesn’t fully represent the processes we have in place. A centralized repository with linked issues, branches, and automation can create a workflow that can accommodate any stage, methodology, or release of the software.

In addition, it gives the teams flexibility to change course when needed. If a team needs to delay or move a piece of code it can be easy to do so with this type of structure.



AccuRev

Collaboration is one of the key aspects of highly-functioning development teams. Delivering “done” code between these teams gives full compliance with the process while customizing what’s needed for each individual team. This brings together everyone - Agile, hybrid, or mixed.

## Conclusion

Traditional SCM branching patterns and systems are designed around decades-old software methodologies. Mapping them to modern practices only hinder and get in the way of the software development process. Using promotional and automated SCM solutions can offer a powerful alternative and allow teams to deliver rapidly and with higher quality. This type of hierarchy is powerful to teams wanting to maintain software configurations for multiple process, releases, and distributed teams.

*At AccuRev, our core expertise is process improvement. We’ve got the expertise, coaches, trainers, consultants, and of course, the tools to help you successfully optimize a development process and best practices throughout the enterprise. In addition to product support, we offer a host of services designed to optimize your process and implement process improvements like multi-stage continuous integration.*

AccuRev

AccuRev, Inc.
10 Maguire Road
Lexington, MA 02421

Phone: +1 781-861-8700
sales@accurev.com
www.accurev.com