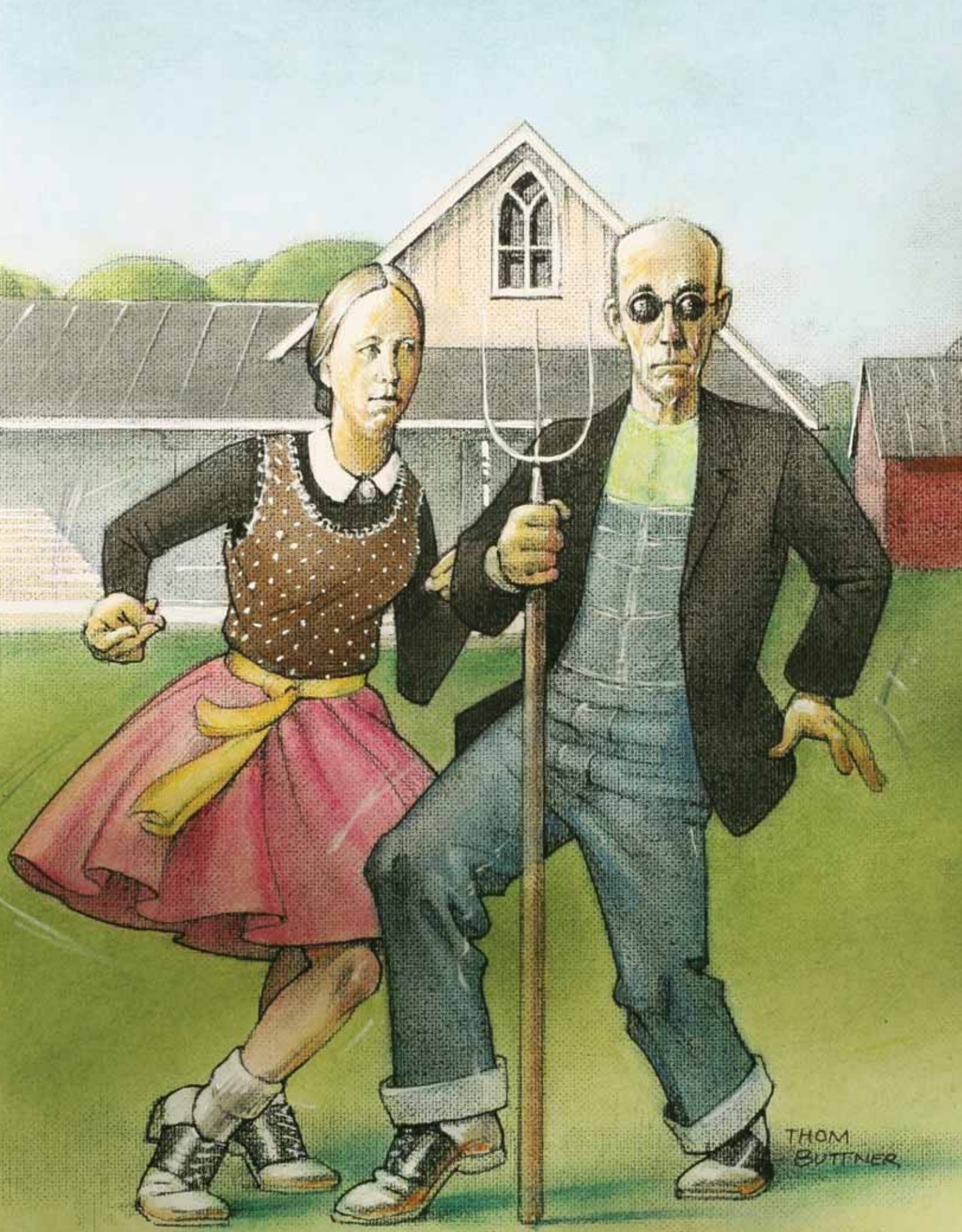
February 2006 $9.95 www.StickyMinds.com

BETTER SOFTWARE

The Print Companion to StickyMinds.com

A Critical Line of Defense

ARMING YOUR SOFTWARE DEVELOPMENT PROCESS


THOM
BUTTNER

BREAKING With Tradition

An Agile Spin on Software Configuration Management

by Steve Berczuk

We do not usually associate the words "simple," "dynamic," or "agile" with software configuration management (SCM), but in many cases a simple, dynamic, and agile SCM process can greatly improve the productivity of a team. SCM gives you the ability to identify and control changes to your source code and reliably build a version of an application from a point in time. Agile SCM further enables you to integrate frequent changes, adapt to these changes, and get feedback on how they affect the quality of the system that you are building. To support feedback, agile SCM processes emphasize frequent integration and testing at all levels, in addition to the traditional SCM areas of build, release, and version management.

COMMON Mistakes

Software configuration management is definitely beneficial, so why are people often frustrated by it? The answer is that teams tend to misapply SCM in one of two ways: too much process or too little process. Ironically, one is often the consequence of the other.

With too little process, there's no mechanism in place to correct mistakes. The codeline quickly degrades, so rules and procedures are added to reduce risk. When too many are added, the workflow is disrupted instead of supported. Despite a sense of security, errors still make it past development and are discovered much later. Since the context for the error has been lost, it is more difficult and more expensive to fix. See the StickyNotes for a link to the article "The Illusion of Control" that explains how the attempt to gain control by adding rules often backfires.

Too little process is often a reaction to witnessing the perils of too much process. Teams avoid anything that looks like SCM and lose the coordination advantages that SCM provides. It appears that they are making progress, but they have no way to verify that the changes made in one place don't interfere with work elsewhere.

To establish an appropriately agile SCM process, you need to overcome your fears and adopt practices that fit your development process.

AGILE SCM Practices

To maintain a quality codeline and to be more productive, agile teams rely on communication, frequent integration, and the resulting feedback—using just the right amount of process. Some agile practices that you can use even in a non-agile environment are:

- Working on a single active development line
- Working in private workspaces
- Using continuous integration
- Performing frequent testing
- Deploying Frequently

Some of these practices cause worry among those who are used to more traditional approaches to SCM ("Don't you lose stability by committing changes so frequently?"). Agile SCM allows more errors to occur but reduces risk by catching and correcting them more quickly, before they can do harm.

ACTIVE DEVELOPMENT Line

Some teams allow developers to work on branches in an attempt to preserve the integrity of the main codeline. The branches cause deferred integration, which makes development more difficult since many problems surface during integration, long after they could have been detected. In contrast, an active development line is a single codeline to which every team member commits changes.

For a single active development line, you need:

- A culture that encourages fine-grained, frequent check-ins
- Optimistic locking
- The supporting practices of private workspaces, continuous integration, and frequent testing

ILLUSTRATION/THOM BUTTNER

PRIVATE Workspaces

Team members do need some degree of isolation to allow uninterrupted work. Developers should have a workspace where they can control when to accept changes. To create a private workspace:

- Provide resources (disk space, database space, server software) that allow a developer to run a complete system. Some teams use shared components either in an attempt to save space or to prevent people from delaying integration. Space is cheap compared to developer time, and forced integration disrupts developer flow, hurting productivity.
- Set up a mechanism to allow a team member to quickly create a workspace from a repository. The process should be as automated as possible, so that all developers' environments are consistent.
- Provide a private build process that mirrors the integration build so a developer can evaluate his changes before committing them. Define the build process early in the development lifecycle.
- Have a policy in which developers resynchronize their code and build and test before committing changes. These precheck-in tests should run quickly and be automated to ensure they are run consistently.

CONTINUOUS Integration

A continuous integration system runs a build moments after anyone's check-in, executes more exhaustive tests, and notifies the team if something is broken. This allows team members to detect integration issues that might have slipped in despite their earlier best efforts. There is a variety of continuous integration tools available, and most are fairly straightforward to set up. In the absence of an automated tool, you can start an integration build manually every few minutes. The cost of setting up this process is small relative to the lost productivity caused either by a lingering mistake or by ill-conceived attempts to prevent mistakes.

FREQUENT Testing

An active development line can degrade quickly if continuous integration checks only that the codeline compiles. To identify problems the precheck-in tests would miss, continuous integration must also run a test suite more thorough than the precheck-in suite. In addition to being more thorough, these tests must run quickly enough so that a developer gets feedback while he still remembers what he was doing.

One concern is that tests are difficult to write, particularly if you are working with an application that does not have a modular architecture. Don't let your concerns about not being able to do complete testing cause you to delay performing *any* testing. A simple test can tell you a lot about the health of your system. See the StickyNotes for some testing resources.

FREQUENT Deployment

Working software is only useful if it is installed or deployed. Frequent deployment of a working system gives you an opportunity to work out the bugs in the deployment process, provides a mechanism for feedback on the status of the application, allows you to detect architectural issues that make it difficult to deploy easily, and reduces the risk that an installation or deployment issue will delay release. Frequent deployment validates that we have working software rather than an assemblage of modules that "do stuff." The more frequently you deploy, the better you will become—and the sooner you will catch errors.

The key steps are:

- Create a deployment process early in your development cycle.
- Deploy the application frequently to developer and production-like environments.

Setting up a simple installation/deployment process at the beginning is quick, and it is easier to modify the process as you go than to create a complex process from scratch.

CONCLUSION

SCM is often viewed as the least agile of all software development disciplines, but that is because SCM is often applied incorrectly. By applying the principles of agile SCM, all teams can work more effectively, even if they do not embrace an agile process. In fact, an appropriate SCM practice is a prerequisite for effective agile software development. As with all aspects of agile software development, the tools and processes should be chosen with the goal in mind to develop in a predictable manner a quality software application that provides business value. See the StickyNotes for more on establishing agile SCM and other practices. **{end}**

Steve Berczuk is a software developer, consultant, and author who focuses on applying agile methods to developing software systems at startup companies. Steve is the author (with Brad Appleton) of the book Software Configuration Management Patterns: Effective Teamwork, Practical Integration *published by Addison-Wesley. He is a senior software engineer at Fast Search & Transfer and is a regular contributor to StickyMinds.com. Steve's Web site is www.berczuk.com, and you can email him at steve@berczuk.com.*

Sticky Notes

For more on the following topics, go to www.StickyMinds.com/bettersoftware

- "The Illusion of Control"
- Testing resources
- Establishing agile SCM