JOIN OVER 100,000 PROS! Join Log In | Member Benefits

STICKYMINDS

Agile Testing DevOps Security AI

Balancing Individual versus Collective Code Ownership

ARTICLE

By Steve Berczuk , Brad Appleton Robert Cowham | April 14, 2006

SUMMARY

The subject of individual -vs- collective code ownership is often the bane of many heated discussions about code change authorization/access and concurrent -vs- serial development. Opponents of collective ownership often claim that it results in "no ownership" of the code and that individual code ownership is better for managing attempts at concurrent changes. Oppenents of individual ownership often counter by saying individual ownership inhibits refactoring and goes against the team ethic of XP and other Agile methods.

One of the commonly touted programming practices of several agile methods is something called **collective code ownership**, where anyone on the team can make any authorized functional change or design quality improvement (e.g., a "refactoring") to any file within the scope of their task. The Agile method known as Feature-Driven Development (FDD), featured in another article this month, is one of the few agile methods that uses the more restrictive model of **individual code ownership** to restrict changes to a class/module to be made by its assigned "owner."

The subject of individual -vs- collective code ownership is often the bane of many heated discussions about code change authorization/access and concurrent -vs- serial development (e.g., **copy-modify-merge versus lock-modify-unlock**). Opponents of collective ownership often claim that it results in "no ownership" of the code, and that individual code ownership is better for managing attempts at concurrent changes. Oppenents of individual ownership often counter by saying individual ownership inhibits **refactoring** and goes against the team ethic of XP and other Agile methods.

The problem with both of these is that they are each extreme positions. Individual Code Ownership, in its purest form, means exclusive access control: no one but the "owner" is allowed to edit the particular module or class. When code ownership is interpreted so rigidly as to exclude allowing others to to make updates, we have definitely seen it lead to the following problems on a regularly recurring basis:

We use cookies to ensure you get the best experience on our website.

Accept

Read More

- Holds changes/code "too close to the vest" and leads to defensiveness and/or local optimization at the expense of the "whole" team/system
- Increasing the project team's "truck factor" by limiting the number of people with intimate knowledge of that module/class within the overall system

At the same time, we have seen cases where "Collective Code Ownership" degrades into "no ownership" when there is no collective sense of team ownership or accountability. So what is the right balance?

We think there is a successful middle ground, one that is sometimes referred to as **Code Stewardship**, where the code-steward is both guardian and mentor for the body of knowledge embodied by the module/class. The Code Steward's job is not to safeguard against concurrent-access, but to instead safeguard the integrity+consistency of that code (both conceptually and structurally) and to widely disseminate knowledge and expertise about it to others.

- When a portion of a module or class needs to change, the task-owner needs the agreement of the code-steward
- The task-owner explains what they are trying to accomplish, the code-steward helps them arrive at the best way to accomplish it, and provides any insights about other activities that might be related to it
- The task-owner makes their changes, seeking advice/counsel from the code-steward if needed (possibly thru pairing or dialogue or other means)
- Before committing changes to the codebase, the code-steward is involved in review or desk-checking (and possible coordination of concurrent activities)

This can be accomplished by pairing with the code-steward, or simply by seeking them out as an FDD programmer would a Chief programmer/architect. The code-steward is like the "editor-in-chief" of the module or class. They do not make all the changes, but their knowledge and expertise is still applied throughout. The benefits are:

- Concurrent changes can still be made and wait-times avoided while still permitting notifications and coordination
- Knowledge is still disseminated (rather than hoarded) and spread-around the team
- Collective ownership and its practices, such as refactoring, are still enabled
- Pair programming can still be done, where pairing assignments can be based in part on who the "steward" is for a piece of code. (At some point stewards can even hand-off-the baton to another)

The bottom-line is that collaborative ownership and authorship is still essential. Code ownership isn't supposed to be solely about controlling concurrent access (and is very suboptimal as a concurrency-strategy, even though some merge-a-phobic shops will swear by it). Code ownership is also about safeguarding the integrity and consistency of the design and implementation, and also about disseminating knowledge of the owned module/class, as well as knowledge of practices and patterns for its succful "care and feeding." If we take "ownership" to either extreme, and keep it there—the result is impractical and imbalanced.

When Brad presented the notion of Code-Stewardshop on several agile forums, he

We use cookies to ensure you get the best experience on our website.

token" for permission to edit a class/module.

That is not how it works. The code-steward serves as a mentor and trail-guide to the knowledge in the code. Consulting the code-steward is **not** about getting permission, it *is* about receiving knowledge and guidance:

- The purpose of the protocol for consulting the code-steward is to ensure two-way communication and learning (and foster collaboration). That communication is a **dialogue** rather than a mere "token" transaction. It's not a one-way transfer of "control", but a two-way transfer of **knowledge**!

Perhaps a better way of describing it would be the way **Peter Block** defines *stewardship* in his book of the same name **Stewardship: Choosing Service over Self-Interest**,

- Stewardship is being accountable to the larger team or organization by "operating in service, rather than in control, of those around us"
- "We choose service over self-interest most powerfully when we build the capacity of the next generation to govern themselves"
- Stewardship offers a model of partnership that distributes the power of choice and resources to each individual
- Stewardship is personal—everyone is responsible for outcomes; mutual trust is the basic building block, and the willingness to risk and be vulnerable is a given
- Total honesty is critical. End secrecy. Give knowledge away because it is a form of power

When practiced properly, *collective code ownership* is in fact an ideal form of stewardship (but not the only form). Stewardship may ultimately end-up as collective-ownership if given a sufficient period of time with the same group of people. However, early on we expect stewards to have a more direct role. The form of *code-ownership* that **Feature-Driven Development** (FDD) practices may seem fairly strict at first, but is really intended to be the initial stages of code-stewardship in the first two quadrants of the situational leadership model.

We believe the form in which stewardship should manifest itself is *situational*, depending on the skill and motivation of the team and its members. In **Ken Blanchard's model of situational leadership**, there are four quadrants of leadership-style, each of which should be used on the corresponding combination of hi-lo motivation and hi-lo skill for a given task:

- **Directing** (hi directive + lo supportive, for "enthusiastic beginners")
- **Supporting** (hi directive + hi supportive, for "disillusioned learners")
- **Coaching** (lo directive + hi supportive, for "reluctant contributors")
- **Delegating** (lo directive + lo supportive, for "peak performers")

If we apply the concepts and principles of "stewardship" using the appropriate situational leadership-style, the outwardly visible result may appear to transition from individual code ownership, to code guardians/gate-keepers, then code coaches/counsellors, and finally to truly collective ownership.

We use cookies to ensure you get the best experience on our website.

individual "styles" are the extreme ends of the spectrum, with "code counsellors" as the style in between those two extremes.

Topics: configuration management, version control,

ABOUT THE AUTHOR

Steve Berczuk

Steve Berczuk is a Principal Software Engineer with experience as a manager, Scrum Master and technical lead in Boston, MA. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a Certified ScrumMaster. Contact Steve at steve@berczuk.com or visit berczuk.com and follow his blog at blog.berczuk.com.

Brad Appleton

Brad Appleton is an Enterprise Agile+DevOps leader, coach & manager, and seasoned DevOps/ALM/CM solution architect at a large Fortune 100 company. Currently he helps organizations and teams scale, adopt and apply lean/agile development methods and DevOps/ALM/CM practices and tools. He is co-author of *Software Configuration Management Patterns*, the Streamed-Lines branching patterns, and a former section editor for *The C++ Report*. You can read Brad's blog at blog.bradapp.net.

Robert Cowham

Robert Cowham has long been interested in software configuration management while retaining the attitude of a generalist with experience and skills in many aspects of software development. A regular presenter at conferences, he authored the Agile SCM column within the CM Journal together with Brad Appleton and Steve Berczuk. His day job is as Services Director for Square Mile Systems whose main focus is on skills and techniques for infrastructure configuration management and DCIM (Data Center Infrastructure Management) - applying configuration management principles to hardware documentation and implementation as well as mapping ITIL services to the underlying layers.

We use cookies to ensure you get the best experience on our website.

UPCOMING WEBINAR

LETS HANG!

FEATURED RESOURCES

TESTING
Mobile Software Testing eGuide
White Paper | StickyMinds

TESTING
The Care and Feeding of Your Software Testing Career
White Paper | StickyMinds

TESTING
Test Automation eGuide
White Paper | StickyMinds

TESTING
Internet of Things eGuide
White Paper | StickyMinds

We use cookies to ensure you get the best experience on our website.

USER COMMENTS

0 comments

YOU MAY ALSO LIKE

TESTING

Why Your AI Testing Suite Isn't What It Was Touted to Be

Nahla Davies | August 26, 2026

TESTING

Exploratory Testing Isn't a Junior Skill—You've Just Been Letting Juniors Do It

Nahla Davies | August 21, 2026

TESTING

Choosing the Right Testing Tool: How the Idea-t Framework Can Help

Isabel Evans | August 19, 2026

Join the crew for FREE! Access exclusive content, contribute, and more.

See All Member Benefits

We use cookies to ensure you get the best experience on our website.

We use cookies to ensure you get the best experience on our website.