The notion of a single owner or multiple owners of code has existed as long as we’ve been building software. Code ownership measures the extent to which a developer (or developers) “owns” a single part or multiple parts of a software system. The concept is often expressed in terms of who controls code editorship (who has modification rights). At one end of the spectrum, every contributor to a system has ownership of the code. At the other end, each contributor to a system owns only the code he or she actually wrote. The decision of which ownership model to follow (or even a midspectrum ownership model) is a topical and interesting industry issue. In this, the first instalment of this department, two prominent industry representatives discuss what code ownership means to them. —Steve Counsell

Code Ownership—a Quality Issue

Sigrid Eldh

CODE OWNERSHIP implies a natural sense of responsibility and pride in your work. In my experience, a collective code ownership model seems to work best when it involves two to eight people. Being a single developer and owning the code independently is too much of a risk—the danger is that you don’t see your own code issues. The pair-programming approach has made that clear. A small team can easily discuss, plan, and polish the code, making the best decisions on strategies such as refactoring.

It’s also worth bearing in mind that no two software systems or companies are the same. The difference between the mix of skills, experience, cultures, code “idioms,” and belief systems results in individual coding styles. Some cultures tend to be more individualistic; some are more collective. So, collective code ownership might work great for one team and be lousy for another, even if we’re talking about the same software product and company. Because people are different, teams and organizations differ in how they take responsibility for their quality, planning, code development, and bug corrections.

I’m a strong believer in fast, efficient feedback and correcting your own problems. This fosters the value of making a quality effort early—because it pays back. When code ownership grows to larger teams, following a collective set of common design rules must come into force—as does the notion of having “code police.” This might be a necessity if the software is too large or is the result of a set of poor initial architectural decisions.

Regardless of whether you develop code individually, in small teams, or in larger teams, what matters from a quality perspective is usually the persistence to measure outcomes beyond the simple pass or fail of test cases, and to care about the quality of your own work. Equally important is to have effective ways to communicate issues to others who are directly or indirectly impacted by your code. This is where we usually fail as a development community. Follow-up on all aspects of quality improvements and code changes is of the utmost importance to achieve code quality. Leaving things half done is a recipe for failure. If we don’t have time to or can’t be bothered to address quality in organizations with more than one team, we soon experience code decay.
the source code. Here is an area in which good architecture and common test-design-code rules could make a huge difference in efficiency. Otherwise, the test suite in an agile continuous integrated build and test quickly becomes costly, where no designer maintains test cases but adds new ones. This behavior often results in a waste of effort through overlapping test cases. Furthermore, you are led into a false sense of security through the abundance of test cases, instead of focusing on how well tested your code is. Code ownership also means ownership of the test code—and this is often forgotten.

SIGRID ELDH is a senior specialist in software test technology and researcher at Ericsson in Stockholm. Contact her at sigrid.eldh@ericsson.com.

**Code Ownership—More Complex to Understand Than Research Implies**

**Brendan Murphy**

A NUMBER of researchers, including me, have published papers on code ownership from a number of perspectives, all identifying that the greater the code ownership by individuals or organizations, the greater the code quality. Most of the research measures ownership on the basis of which developers edited the code. The logical conclusion from the studies is that a project should optimize its product development through a rigid adherence to Conway’s law (an organization that designs systems is restricted in some sense by the structures of the organization itself). An underlying assumption is that only the developer who “owns” the code (understands the code’s functionality and the impact of any changes on the total system) should edit the code. Another assumption is that ownership is a proxy for responsibility, whereby code owners will take greater care when altering the code because any resulting issues might reflect badly on the owner.

When I discussed these results with product teams, although they agreed on code ownership’s importance, they viewed these results and assumptions as too simplistic. They had three primary issues. First, restricting the number of people who can alter specific code will create development bottlenecks. Second, restricting the developers who can alter code can result in bad coding practices, in terms of readability and development of comprehensive unit tests. Finally, several development methods organize teams around developing end-user features, rather than the product architecture, and such organization requires freedom for engineers to change any code they wish.

Further discussions indicated that ownership research should consider code complexity and other definitions of ownership. The product teams believed that any competent developer should be able to update code modules whose functionality is clearly defined and has unit tests that verify the code’s functional completeness. At the other extreme, most teams would restrict the engineers who could edit an OS’s memory management code. So, the importance of code ownership is probably related to either the code’s complexity or its position in the overall product architecture.

Having only one person with knowledge of critical code represents a risk: if the person leaves the group or company, disruption will occur. Product teams should seek to increase the number of people who can alter the code, while managing the risk. This is often achieved through tools that manage code review. Engineers can update any code, but the code owner must agree to changes before they can be checked into the code base. So, you can identify code ownership either by determining who alters the code or identifying which engineers are responsible for signing off code changes.

SOFTWARE development is complex, so a single development factor is unlikely to have a universal positive impact. Ownership research must also take into account other code and development attributes to understand ownership’s overall impact on code development.

**BRENDAN MURPHY** is a principal researcher at Microsoft Research, Cambridge, UK. Contact him at bmurphy@microsoft.com.