Like this Knowledge Brief? Find more like it at the Rapid Learning Cycles Resource Center:

http://rapidlearningcycles.com

Here are a few other Knowledge Briefs that might interest you:

- Building Blocks of the RLC Framework: How to Write KDs, KGs and Activities for Clarity and Focus
- The MVP for the Physical World: How to Adapt the MVP for Products That Are Hard to Change
- The Power of "Learn Before Design": Six Reasons to Use the Rapid Learning Cycles Framework

About the Author

Katherine Radeka has a rare combination of business acumen, scientific depth and ability to untangle the organizational knots to remove the barriers to change. Since 2005, Whittier Consulting Group, Inc. has helped some of the world's leading companies get their products to market faster.

She has a global reach with clients in Europe, North and South America, Asia, and Australia/New Zealand. She has worked with companies in pharma, biotech, medical device, high tech, consumer electronics, food and beverage, and consumer packaged goods, among others. She currently supports more than 150 implementations of the Rapid Learning Cycles framework through the Rapid Learning Cycles Certified™ Professionals Community.

Katherine is the author of two books. Her first book, *The Mastery of Innovation: A Field Guide to Lean Product Development* won the Shingo Research Award in 2014. This book contains 19 case studies of companies, including Steelcase, Ford, Novo Nordisk and Philips Electronics, who have used lean ideas in product development to get their ideas to market faster.

Katherine's second book is *The Shortest Distance Between You and Your New Product: How Innovators Use Rapid Learning Cycles to Get Their Best Ideas to Market Faster.* This book summarizes Katherine's ground-breaking work to integrate Agile Development with her work on Knowledge Capitalization into a proven method for accelerating innovation.

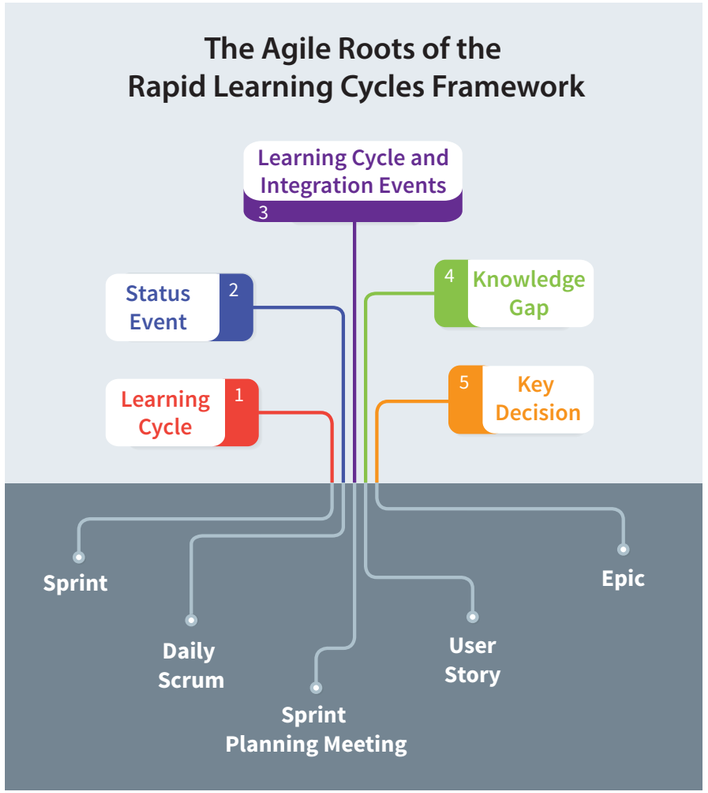
Katherine has climbed seven of the tallest peaks in the Cascade Mountains and spent ten days alone on the Pacific Crest Trail until an encounter with a bear convinced her that she needed a change in strategic direction.

RAPID LEARNING CYCLES INSTITUTE℠

Planning For Uncertainty and Discovery

The Agile Roots of the Rapid Learning Cycles Framework

By Katherine Radeka



Planning for Uncertainty and Discovery

The Agile Roots of the Rapid Learning Cycles Framework

Key Takeaways

- **Agile is not sufficient for programs with high cost-of-change and complex dependency chains, yet Agile's tools become necessary in environments with high uncertainty.**
- **Rapid Learning Cycles adapts Agile's tools to help teams make better decisions when those decisions must stick — but the work inside a Learning Cycle is much different.**
- **Agile Software Development integrates well with the Rapid Learning Cycles framework because the foundations are well-aligned.**

I live in a home built in 1910 with original basement walls that were poured using a form of concrete that is known to have a lifespan of about 120 years before crumbling into dust. The whole weight of the house sits on these walls, which are already starting to show signs of deterioration. We need to deal with this before too much settling harms the rest of the structure. The simplest repair would be to pour new basement walls on the inside of the old ones, and transfer the weight of the house to these new walls.

A major project like this has a lot of uncertainty that can't be cleared up until the work starts, and we know exactly what we're dealing with: How much will it cost? How long will it take? The contractors will be working off of plans that have a lot of assumptions built into them about the existing conditions that may or may not be true. Who knows what they'll discover when they start digging?

Agile is a method for dealing with situations with high uncertainty, and we are going to be dealing with a lot of that. Yet this type of project isn't suitable for Agile, either—not on its own. We can't just rip out the new walls and re-pour them if we learn later that they aren't going to work as expected. The project will have a lot of moving parts that need to stay coordinated and in sequence. This is a project that needs Rapid Learning Cycles so that we make good decisions that stick.

Agile Is Not Sufficient for All Programs

The methods that we call Agile today developed in an environment with low cost-of-change and low dependencies. You can see this with the way that Scrum, the best-known Agile method, handles planning and backlog management. Aside from the order of the user stories (defined chunks of work) in the backlog, there is no other way to track dependencies. The Planning Game assumes that the product owner can re-prioritize these user stories at will. Even in software that's never 100% true, but once the basics of the architecture are in place, the user stories can be developed in the order that will deliver the most immediate customer value.

Agile emphasizes driving these user stories to "done-done" status, so that a real user can test them and give feedback. If the developer missed the mark, he just changes the software to fix it. If the UI design is leading people down the wrong path, the designer just rearranges the screen for the next round. It takes time—but not nearly as much time as replacing a strategic supplier. In fact, since the cost-of-change is so low, the Agile community emphasizes getting something—even something terrible—in front of a user to give her something to react to, so that the team gets better feedback. This concept reaches its peak in the Lean Startup's goal of releasing a "minimum viable product" as quickly as possible, then improving it based on real customer experiences.

Meanwhile, traditional project management does not handle uncertainty very well. If we prepared a detailed project schedule for the basement based on what we know now, it's almost certain to be wrong, probably vastly underestimating the amount of time and money that this will take. As we learn more about the work to be done, the plan will be changing constantly. Agile can help us plan for this uncertainty, which is why Agile project management is one of the roots of the Rapid Learning Cycles framework.

Scrum in the Rapid Learning Cycles Framework

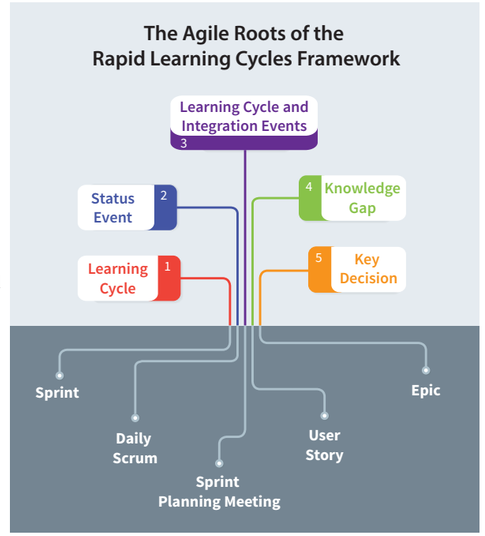
If you're familiar with Scrum, you may have recognized elements of it in the Rapid Learning Cycles framework. The Status Event is a scrum. A Learning Cycle looks like a Sprint. The Learning Cycle and Integration Events include a planning session adapted from the Sprint Planning Meeting. You can think of a Knowledge Gap as a "user story" and a Key Decision as an "epic" if that's helpful to you. We changed the language because we did not want to get engaged in any Scrum purity wars, and we wanted to emphasize the ways that hardware development is different from software development.

A team running Rapid Learning Cycles looks, on the surface, like a team running Scrum. But the work that happens inside a Learning Cycle looks quite different from the work that's happening inside a Sprint for a software team.

For example, hardware teams don't want—or need— to meet every day on every project. If the team does not have enough happening on a day-to-day basis, the daily stand-ups get tedious and lose their power to generate pull. It's better to meet every other day or twice a week, if that allows for more work to be completed between events. The Learning Cycle and Integration Events are much, much more than just demo/planning events. This is where the heart of the work gets done: capturing and sharing knowledge, making the team's most important decisions, building the team's understanding of the product that will be the result of their work.

An epic is a large chunk of functionality that allows a user to do something significant. In an online banking system, the "Transfer Money" feature would be an epic. It would be broken down into user stories like "Transfer between accounts at the same bank," "Transfer money out to a different bank in the United States" and "Transfer money internationally."

You could do something like this for hardware, and if you make a product that has areas of high modularity—like a tractor—then this may make sense. You can organize meaningful development work around a new attachment to "mow a flat pasture" or "mow a sloped lawn." But even then, that assumes that the tractor itself is in place, with a standard interface for mower attachments, and a well-understood production process.



The early versions of the framework treated Knowledge Gaps like User Stories: bundles of meaningful work that could be delivered within one Learning Cycle. The Rapid Learning Cycles framework really came together when we recognized the central role that Key Decisions play in determining the final outcome of a product development program. They tell us what our most important Knowledge Gaps are likely to be, and where to focus the limited time we have before Key Decisions need to be made.

In Agile Software Development, teams aim to write shippable code that is ready to be released to a real customer and the culmination of a sprint is the product demonstration showing the new features added during the Sprint. In the hardware space, we can often find ways to close Knowledge Gaps without building the product itself. The culmination of a Learning Cycle is the Learning Cycle Event where the team shares the knowledge that has been built in the cycle.

Agile Software Development within the Rapid Learning Cycles Framework

The good news is that if your product consists of both hardware and software components, Scrum and other flavors of Agile work well within Rapid Learning Cycles as long as the cadences are coordinated between Learning Cycles and Sprints. Ideally, they'd use the same cadence but some software teams like to run twice as fast, with two Sprints for every Learning Cycle. That also makes it easier for software teams to recognize when they need some Rapid Learning Cycles, too.

This is another reason to separate the terminology: so that teams running both don't confuse user stories with Knowledge Gaps or Learning Cycle Events with Sprint Planning Meetings, when the teams are doing different types of work.

If the software team has to make major decisions that have high cost of change, such as architecture, deployment strategies or field update processes, then they can use Rapid Learning Cycles to understand what they need to know and then build the knowledge to make those decisions with high confidence.

Rapid Learning Cycles is Agile for Hardware Development

The "agile organization" has become a recent buzzword as some companies try to leverage the benefits they've seen in Agile Software across the organization. Agile has a lot of strengths to leverage that I've not covered, like the focus on people instead of procedures, and on delivering demonstrable customer value instead of promises. At the same time, Agile is not a good fit for all of the work that happens inside a company, especially those programs that build new hardware products that must be produced at industrial scale. There is even a home for traditional project management for things like major construction projects.

For our basement project, we'll use Rapid Learning Cycles to reduce the uncertainty, and then manage the project with traditional project management tools (Gantt charts and the like). The project will have a complex chain of dependencies and a defined sequence that must be followed (empty the basement, demo the built-in cabinets on the walls, build the forms, etc) so it will be in the sweet spot for traditional project management. But we'll still need to run some Rapid Learning Cycles alongside the project so that we're prepared to deal with the things we can't predict in advance.