The RITUAL of Retrospectives

How to maximize group learning by understanding past projects

by Norm Kerth

You've just finished your software release. You have signed off, and it's been shipped. You're done, right? No! The moment a project ends is the perfect time to reflect on the entire project to see what there is to learn—it's the unique moment when the project can be seen in its entirety. It's also a perfect moment because the end of your project forecasts the beginning of a new project in the not-too-distant future, which you can improve by applying what you've learned from this project. You can look at completing the project as having "paid your tuition." So now what are you going to learn from it?

Can you, on your own, reflect on your past project experience and find significant insight? Not really. In real-life projects, there are many pieces to the whole story of the project, and each individual on the project knows his piece of this story. Until everyone on the team joins together and they collectively tell the story, the learning is likely to be minimal, and the probability of significant group learning is slight. To improve that probability, we need a gathering of the team.

One developer, for example, had a personal style for allocating pieces of his software to particular files. It was an odd style, but one with which he had become comfortable. He saw no reason to change, despite tester after tester asking him to adhere to the team's standard.

QUICK LOOK

- Benefits of a retrospective
- How to structure the meeting
- 3 precautions to consider

During a retrospective, this developer learned that 60 percent of the defects in the build process were due to custom-built scripts—scripts that integrators were writing to incorporate his software into the rest of the system. Until this moment, he'd never known the impact his personal style had on other departments, so he'd had no motivation to change. Confronted with the bigger picture, he now agreed to follow the team's standard in future projects.

This gathering of the team to tell the story is a ritual we call a *retrospective*. The goals of the retrospective are for the team to review the project and answer the following questions:

- What is it we did well, that we want to make sure we don't forget?
- What did we learn?
- What should we do differently next time?
- What still puzzles us?

The *collective* telling of the story illuminates pieces of the project that no one person could see by themselves. The whole community begins to see the whole picture. Connections are made, learning occurs, and the team collectively understands what changes need to be made.

Why You Should Care

Knowing you need to change your organization's process and *doing* something about it are two different issues. Most likely, you know a great deal about implementing changes in your department. How about across your whole organization? Here are three reasons why being serious about retrospectives can help you cross that gap between thinking and doing.

1) Jumpstarts the learning environment

A retrospective, composed of developers, testers, and managers (and anyone else with an important piece of the story), can help the whole community see where changes need to be made.

A retrospective is your best first tool toward establishing a learning culture. Through it, you can identify the most important issues to the whole community, and people can see the importance of embracing these improvements. It's no longer a QA versus Development versus Management issue. Instead, it transforms into a whole-team issue with everyone working toward a common goal. The whole community selects what's important to change, and thereby owns the change.

2) Captures effort data

One of the benefits of a retrospective can be the integration and analysis of important project data, such as person-power expended, size of effort, cost, time, and estimates verses actual. (While "effort" data should be collected throughout the project, my experience shows that this discipline is the first to go when times get difficult.)

An end-of-project analysis identifies what important data might be missing. This will be the last moment at which you have any hope of collecting accurate, critical data, and it still might be possible to determine the data that wasn't captured earlier.

Discussing effort data during the retrospective adds a dimension of fact to the telling of the story. More of the big picture is understood.

A year ago, I was working with Bob, a systems integrator with a major software office products company, in an effort to streamline his organization's build automation. I asked him about the kind of defects he most commonly encountered during a build. Bob didn't have a quick answer off the top of his head, but pointed to an eighteen-inch-thick stack of pink sheets stacked on the edge of his desk. Each sheet listed one defect he had experienced during the recently completed project.

At that very moment (unbeknownst to Bob), the software developers upstairs were discussing how to change their check-in process. Frankly, from what they could see, there didn't seem to be a need to change much. They would, of course, have had a completely different viewpoint had they been able to access the wealth of data in Bob's stack of papers.

Had the entire project community held a retrospective, that stack of pink sheets would have been discovered.

In the absence of such a retrospective, it took an eagle-eyed consultant to put Bob together with the developers and get them talking. Upon discovery of Bob's data, the developers were shocked by the number of build problems. Bob was added to the committee, and they spent the next six weeks reviewing the pink sheets and revising their tools and procedures.

I talked with Bob a few days ago, and he was happy to report a 50 percent reduction in build breaks in the last year.

3) Serves as vital corporate memory

While working at a Fortune 100 company, my colleagues and I led retrospectives on eighty-six projects. Our documentation—the wisdom captured and the data recorded—went into an archive that project managers (especially the new ones) could review and learn from.

This archive proved its value in several ways. If managers feared they were at the beginning of a seriously under-resourced project, for example, they could use the past history of similar projects to effectively make their case for more resources.

Starting projects without adequate time or resources was a common occurrence at this company. I realized *why* while working with this retrospective archive: the firm's natural memory of projects was *severely* optimistic. This led to the funding of chronically unrealistic project plans, which precluded realistic project alternatives. But faced with the facts from eighty-six historical events and the wisdom from both successful *and* unsuccessful projects, the corporate leaders learned to better recognize the difference between realistic and unrealistic project plans.

Running the Retrospective Ritual

Over the years, I've experimented with many ways to run retrospectives, and I've tried a number of different exercises to help focus discussion. I have found it useful to

divide a retrospective into three parts: The Readying, The Past, and The Future.

1) The Readying (Prime Directive)

In the first part of the retrospective, I focus the group's attention on what needs to happen for the retrospective to be successful. This means explaining how the retrospective will proceed, establishing ground rules, and generally reducing the anxiety people might have about participating in such a ritual. Fear is a common emotion for people to have as they go through their first retrospective. Concerns vary, but include:

- the fear that they might be attacked
- the fear that during discussions they might look like a fool
- the fear that events will be reflected in their performance review
- the fear of hurting someone's feelings

These emotions need to be dealt with before a meaningful dialog can occur. To address these fears, I introduce the "Prime Directive of Retrospectives," and charge the team to develop ground rules that will reduce the fear.

The Prime Directive of Retrospectives says:

> *Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.*

This directive is the basis by which I run the retrospective, and all discussion is based on this belief. If the discussion shifts toward blame or criticism, I stop the discussion and we rework the message until it is congruent with the Prime Directive. Defending the Directive is often as simple as stating, "Okay, that is the view from *your* piece of the story; who *else* knows a piece of this story?"

I charge the group to develop their own ground rules in order for them to accept the responsibility of establishing safety—rules which usually include:

- Don't interrupt
- Listen until the speaker is done talking before thinking of a response
- What is said in this retrospective is confidential unless we all agree otherwise
- Everyone is entitled to have his or her opinion, even if it's different from yours

There are two rules I always add:

1. Involvement in this retrospective is optional
2. There will be no humor at the expense of anyone in this room

With participation in the retrospective being optional, those afraid of being coerced into something start to relax. As for the second rule, humor that targets participants doesn't work to anyone's benefit, because it's hard for the facilitator to know if a joke was a form of endearment or an insult. It's possible for a jest to be intended affectionately, but to be taken as a personal slight because the person on the receiving end is feeling a bit vulnerable. Because of this, that kind of humor is best left out of a retrospective.

I have found that laying out some ground rules moves a group into a stage where they are ready to review their project.

2) The Past (Key Events)

In the second part of a retrospective, the team works on the four framing questions we listed earlier in this article: What worked well? What did we learn? What should we do differently next time? What still puzzles us?

But simply *asking* these questions will yield little learning. Unstructured discussion will just bring out opinions and debate based on underlying pieces of the story that have not been completely told or explored.

Instead, we identify *key events* or *happenings* that occurred throughout the project, and place them on a timeline. Testers are likely to find different key events than what the developers will identify, and the project managers are likely to select yet another set of happenings as being important to discuss. (And that's to be expected; each person and each group will have seen the project from their own limited viewpoint.)

When laid on a timeline, these events and happenings focus the community's attention on pieces of the whole story. I divide the timeline into about six equal time periods and we begin to review the project period by period. To review, I read the events for a particular period, and ask the group which events are important to talk about. When an event is mentioned, I ask to hear the story that makes this event significant. After hearing one person's story, I ask if others have another piece to add. As the storytelling proceeds, I lead the group toward developing an answer to one of the framing questions. Once we have an answer, we move on to other events.

For significant projects lasting six months or more, this storytelling might take four to six hours, or occasionally even longer. A group can't remain focused for this long, so I intermix breaks and other activities that might lead to deeper learning and team building. Playing together, sharing a meal, or watching a movie together are valuable uses of rest time. In the middle of one retrospective, we took a break and went bowling! While the group bowled, they also talked about the project—and returned to the timeline analysis with several new answers to add to the stories already told.

3) The Future (Improvement Proposal)

One danger of holding retrospectives is that the sponsor will fail to follow through on the retrospective's recommendations. After significant work is spent reviewing a past project and identifying what had been learned and what to do differently, if nothing changes, then the time spent on a retrospective has been wasted. If the only outcome of a retrospective is a report that gets typed, distributed, and filed, then the retrospective will fail. The report will never be read again. Not only that, participants will lose confidence—in

the retrospective's value, and in the sponsor as well.

To make sure that the wisdom and lessons learned can live beyond the retrospective, the third part of a retrospective is designed to focus the community on what they are going to *change* in the next project. I break the community up into cross-departmental teams and ask each team to build a proposal (a plan that's both specific and realistic) to Management on how to change the software development process in ways that incorporate the lessons learned. In this process, managers, developers, and testers work together to recommend an approach they believe will succeed.

In some cases, groups don't know what a proposal to Management looks like, so I often spend some time covering the basics of selling an idea to Management.

This is work—hard work—and it takes time. But something interesting happens during this exercise. People discover they can be the masters of their own software development process: they *can* change it, and they *want* to change it when they realize they can improve how their group operates. The give and take between the various departments becomes cooperative rather than adversarial. Magic truly happens in the late stages of a retrospective!

I often finish the retrospective by asking the community to find ways to "change the wallpaper" back at work. Invent posters to keep the lessons learned alive, devise a screensaver that reminds everyone of the lessons learned, create T-shirts…"Who knows," I tell them. "Be creative." I hope for the day when a team will post Burma Shave poetry on signs along the path from the parking lot to their desks.

Precautions

A retrospective is a high-powered tool, and as with other powerful tools, you need to exercise some care to assure it is used properly.

1) Import a facilitator

Perhaps one of the reasons retrospectives are not more popular is the risk that the ritual will turn ugly as the Prime Directive is violated and people blame and accuse each other of all sorts of faults. An experienced facilitator can prevent this from happening. When selecting a facilitator, choose someone who is disinterested—someone who wasn't part of the project. You can either facilitate or participate; you can't do both. Trying to do so robs the community of a full-time facilitator *and* a full-time participant—a participant with an important piece of the story to tell and to learn from.

Where do you find an experienced facilitator? Large firms usually have people who can play that role upon occasion; check with your Human Resources department. It's important that the facilitator be familiar with the software development process and the vocabulary.

Seasoned software engineering process groups and mature corporate-level quality assurance groups might have someone who has mastered the job of leading retrospectives. If this is the case, take advantage of their services. If such a group exists but doesn't yet offer retrospective facilitation, you might show them this article, because providing such a service is a terrific way to further corporate improvement initiatives. There are also independent consultants who lead retrospectives. Contact me if you need help finding one.

2) Allocate adequate time

A process such as a retrospective takes a certain amount of time, and it can't be shortened. The team needs time to get ready, then they need time to review the past project, and then to plan for the next project. Over the past twenty years, I've experimented with lengths from two hours to four-and-a-half days. My experience has shown me that a review shorter than two-and-a-half days does not address the important issues from the big-picture perspective, and without that there's no lasting change. I've learned that most teams and projects need about three days.

For projects where there were difficulties (project failure, project cancellation, and the like), a team needs more time than three days. But four days is too tiring, so I've settled on recommending a three-day residential event: check into a hotel or beach house, and live, sleep, and eat together for a few focused days.

Your first reaction might be, "This is going to take a tremendous amount of our time!" Actually, when you think of this review as one part of your whole project—a project that might have run for six months or more—you will discover this important retrospective is less than two percent of the overall cost of the project.

There's no way to cheat: If you short-en this lessons-learned process to a time period that will not allow for careful analysis, then you will get no effective change in your software development process.

One vice president I know decided to limit retrospectives to half a day. Several retrospectives were held in the abbreviated format. The facilitator led the team in identifying recommended changes; but in such a short timeframe the real issues were not explored, the stories were never told, and the big picture was never understood. Because participants didn't appreciate the need for making changes, retrospective after retrospective made the same recommendations…but nothing ever changed.

Eventually, the vice president decided that since nothing ever resulted from the retrospectives, retrospectives *must* be a waste of time. "This just confirms my initial reaction to that retrospective ritual thing," he explained, "and why I was smart to limit them to just a few hours." Armed with this upside-down logic, he then banned any further use of retrospectives.

In this case, the problem was not that retrospectives were a waste of time, but rather that not enough time had been allocated to them to make them significant.

3) Hire an expert if the project had great difficulties

A retrospective on a successful project is *easy* to lead. This is most likely because a healthy project's team already knows how to work together; that contributes to both a successful project and a successful retrospective.

On the other hand, retrospectives on projects that *failed* are more difficult to lead—as are retrospectives on projects that struggled toward completion and where conflict was high. When considering whether you are the right person to lead a retrospective, determine how difficult the ritual will be and compare it to your skills. Don't take on a ritual if you aren't confident that you can handle it.

If you are inexperienced in leading retrospectives, find an easy project first. For the more difficult retrospectives, if you haven't honed your facilitation skills to deal with the strong, disruptive emotions which sometimes surface, you might want to partner with an expert. Hire a retrospective facilitator who is experienced in handling controversy to both help lead the retrospective *and* coach you.

A Retrospective Is an Old Idea

Holding a retrospective ritual is a very old idea. It has served the human species well, as the stories of hunts were retold around campfires and the stories of child rearing were shared as baskets were woven. It has survived this long because it works. It's a fundamental vehicle to discover, share, and pass along the learning from experience—something we also call "wisdom."

I leave you with one question: *Is your organization good at acquiring and using its wisdom in creating software?*

Maybe it's time to try a retrospective. It's something that every true learning organization has as part of its culture, and it's one of the best ways to grow—project by project—into a smarter and increasingly successful organization. STQE

Norm Kerth (nkerth@teleport.com) *has been helping clients improve their software development process for over fifteen years, emphasizing object-oriented methodologies, patterns, and retrospectives. Norm is the author of the recently published* Project Retrospectives: A Handbook for Team Reviews.

PERSPECTIVE

No Pain, No Gain

Bob Marsilio's team members' greatest challenge may have been their religious differences.

We're not talking theological debates here—the problem was of a much more earthly nature. "Our project started out with an incredibly talented staff," Marsilio remembers, "but it was a group of people with dramatically different opinions about our recently adopted object-oriented (OO) engineering processes."

In one camp Marsilio had people with an aerospace background, for which formal documentation, traceability, and approvals at each step were orthodoxy. In the other camp were devotees of extreme programming models—people who believed strongly in improving the system through constant refactoring, total ownership (sometimes called "no ownership"), pair programming, few formalities, and no documentation other than the code.

The tension took its toll. "Here we had a great company, with great products, but we were seeing way too high a rate of staff turnover," recalls Marsilio. Attempts at resolving these religious debates led to "process" team meetings, during which everyone would leave the room in a spirit of agreement—and then return to their old patterns. Some members, worn down by what they saw as personal criticism and arguing, withdrew from the process or—increasingly—sought employment elsewhere. "Our expertise in OO was shrinking, not expanding," says Marsilio, "and we knew we had to create improvement right away."

Marsilio was familiar with the technique of post-project retrospectives—but that was the stuff of post-mortems, not in-process adjustment. "Honestly, what I thought we needed," he recalls, "was an *intervention*."

After some brainstorming, however, Marsilio and his team came to see the retrospective as something that could put the project back on the right track. "Even though we weren't at the end of the project," he says, "we felt a strong need for immediate improvement and decided to squeeze the retrospect in between incremental releases."

So immediately after the alpha release, the company brought in a retrospectives facilitator, who developed exercises to put the team at ease and create a feeling of openness. "By the end of the first day," Marsilio says, "people were eager to tell their part of the story." The third and final day the team discussed improvements to their processes and ways to change their culture, setting parameters for acceptable confrontation styles and agreeing on escalation procedures—all ways to help resolve gaps between OO orthodoxies and concentrate on the bigger picture. "The retrospective helped them express what was happening, and how they felt about it—and they saw that other members felt that way too," Marsilio explains. "What we ended up doing was assigning process to our culture, saying 'this is how we agree to act, this is how we agree to argue.'"

Was it worth it? "Absolutely," says Marsilio. They ran mini-retrospectives after both the beta and general releases, and continue to employ the process on projects as needed. Despite the inherent risks to new projects after any organization's implementation of OO processes, his team ultimately managed to beat the odds by releasing a product (albeit a year late) with a high degree of quality and functionality. "It really changed how our people respond to problems in a positive way," he notes. "And the people who experienced that first, intense, three-day session will carry that awareness with them for a long time." ***—A.W.***