Retrospectives – Learning Not Just Repeating

Tim Mackinnon, Agile Coach
Iterex Ltd.

Is your team making the same mistakes over and over again? Is your team uncomfortable talking about the underlying problems on their project, or even worse, failing to celebrate its successes? You are not alone, and you may be encouraged to learn that many software teams are using a technique called 'Project Retrospectives' to greatly enhance their performance.

Retrospectives Help People

Surprisingly, I find that even the most successful projects with the best technical staff still encounter problems that seem to 'eat away at the morale fibre of the team'[1]. We often see this on teams dealing with stressful situations, like Fire and Police departments, who may even publish a 'Post Mortem Review', which considers the events that led up to a particular situation and discusses how they could have acted differently.

While software projects are not normally life and death situations requiring split second decisions, to members of a team grappling with conflicting technical requirements – similar levels of stress are quite common. This is where the work of Norm Kerth, author of 'Project Retrospectives'[2], is particularly helpful. Originally described as:

> *retrospective (rèt´re-spèk-tîv) – a ritual held at the end of a project to learn from the experience and to plan changes for the next effort*

Kerth and the 'Retrospective Facilitators Group' have applied retrospectives not only at the end of projects, but also in smaller monthly and weekly increments. In each case, these teams took the time to discuss what happened, what worked well and what they should do differently. They also focused on the people involved, and also found their meetings helped them adjust their processes or redistribute their resources more effectively. Furthermore, during each retrospective, they built up a list of actions which they prioritized from to implement in the following weeks.

Planning a Menu of Activities

To achieve these results, the teams above took the time to plan a set of relevant activities, something which Kerth compares to planning a menu. He suggests identifying a 'starter', then moving to a 'main course' and finally finishing off with a 'dessert'. He classifies the exercises as shown in Table 1.

Starters	Main Courses	Deserts
I'm too busy Define Success Create Safety	Artefacts Contest Develop a Timeline Emotions Seismograph Offer Appreciations Passive Analogy Session Without Managers Repair Damage Through Play	Making the Magic Happen Change the Paper Hopes and Wishes Closing the Retrospective

Table 1 - Retrospective courses

By understanding what the team needs, and selecting the appropriate 'dishes' for the three stages of the retrospective, the result is a well balanced session that helps everyone learn what is most important.

Avoiding a Blame Culture

While planning a retrospective is very important, even more important is making sure that everyone understands that this is NOT an opportunity to place blame. Many people initially shy away from a retrospective because of a bad experience which simply degenerated into a 'blame fest'. A great retrospective is an opportunity to learn what things are going well (so they can be repeated) as well as discovering what things need to be improved. For this reason I find it important to read out Kerth's 'prime directive':

> ***"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."***

These words should be significant to any team, however in the case of Agile development, they are particularly important. The tight relationship between the practices, how they support each other, and the principles and values from which they are derived leaves lots of room for mismatches to occur. If everyone is focusing on improvements in one area, then there is a high probability that something else is suffering and it's likely to be causing issues somewhere else. Being able to identify these issues and actually talk about them in an open and honest manner is fundamentally important.

Learning Isn't Easy

While there are many interesting and varied exercises that can be used in project retrospectives, there is one that I think stands out, the 'Create Safety Exercise'. Ironically this exercise is probably the easiest to perform, but it's often skipped. It consists of the following steps:

1. Make everything optional – and stress that the process is not about finding fault
2. Use a private ballot to find out how people feel about saying what needs to be said
3. Gather and tally the ballots for the group to see how safe they feel

For the second step, ask participants to think of a number between 1 and 5 that indicates how safe they feel, as follows:

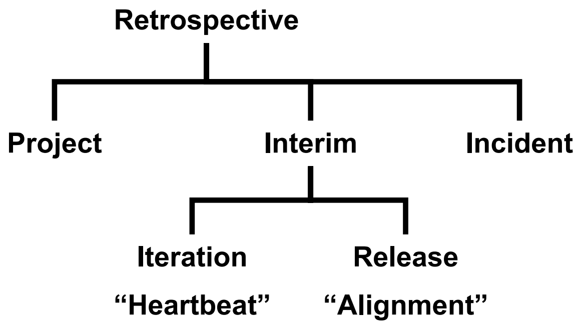
5 – No Problem, I'll talk about anything
4 – I'll talk about almost anything; a few things might be hard
3 – I'll talk about some things, but others will be hard to say
2 – I'm not going to say much, I'll let others bring up issues
1 – I'll smile, claim everything is great and agree with managers

The last item is particularly important, as while it's a serious ranking, it usually breaks any tension in the room. I also find that it's important to stress that the emphasis should be on 'safe enough to say what needs to be said in the context of this team'. This is particularly true if you have new people in the team who might not feel that they can contribute to the content of the discussion. Furthermore, before you ask people to rate their safety make sure they understand the agenda for the day, the optionality of the exercises, and that everyone can usefully contribute to the event even if they haven't been involved in the project from day one.

Dealing With the Long Haul

We find that software projects rarely just stop once they are delivered, and given this longer outlook, it becomes particularly important to consider the health of the team as it is developing and supporting any new features.

Having tried lots of different styles of retrospective over the last few years, I have noticed in the software development world, there seem to be 4 styles of retrospective that each have their own particular use:



Project retrospectives are the largest undertaking of the four styles, and as described by Kerth, you should plan for one to three days to cover the breadth of information that your team has accumulated (which may even be over a period of several years). For this kind of event, you will work with a facilitator who will plan and organise the retrospective and lead the group through the chosen exercises. At the other end of the spectrum are Incident retrospectives, which cater for unexpected events where the team needs to quickly review what happened, and form a plan of action to overcome their problem. In these situations there is very little time to explore how people might improve their relationships, however if you can achieve a high level of safety, it is possible to have honest process level discussions to understand the immediate problem and fix it.

In between these two different styles of retrospective are two other 'workhorses' that give a team a better chance to discuss and learn from what they are doing, as well as practicing expressing themselves to each other.

Iteration "Heartbeats" – Quick Process Wins

Many teams work in weekly or bi-weekly iterations, and the end of each one is the ideal time to explore what happened. Typically these kinds of retrospectives are quite informal and last anywhere from 15 to 60 minutes. They can be self-facilitated and the team ultimately derives a list of actions that they will implement in the next iteration.

This quick retrospective does come with a health warning, as many teams experience frustration in identifying so many actions that they aren't able to deal with them before the next heartbeat arrives. My suggestion (and now current usage of these kinds of retrospectives) is to get everyone to share one thing that they thought went well and would want to repeat, and one thing they would do differently. We record these items on a flip chart and if appropriate quickly vote to determine which item they would like to see improved for the next iteration. You sometimes find teams members playing a game of "Snap", where a common item is vocally snapped by every member of the team.

While very useful, heartbeat retrospectives typically only reveal process related improvements, which aren't necessarily profound or controversial. People are happy to make tweaks to their ongoing process but they need more time to consider larger issues.

Release "Alignment" – Exploring Profound Changes

To explore these larger issues, and to reflect on product direction, team dynamics and goal alignment, I like to arrange for a quarterly team review. This is also the perfect time to hold an "Alignment" retrospective which I organise for a duration of three to four hours (I've often been asked to reduce this time, but teams just aren't able to cover the material and propose recommendations in less than three hours). These retrospectives are more in line with the menu of exercises described earlier by Kerth, however they are run on a slightly smaller scale. A typical agenda might look as follows:

- Create Safety Exercise
- Project Health through Pictures
- Project Timeline
- Review "what went well", "what didn't go so well"
- Actions
- Top Tips for Future Projects

From this list, the Project Timeline is high energy activity which allows everyone to create an overview picture of the project and the items that went well (on green 'post it' notes), and the items that didn't go so well (on red 'post it' notes). The instructions are simple: hang a large sheet of packing paper on the wall, identify some key dates or events with the team, and then ask everyone to write their items on the relevant post-it notes using big black markers.

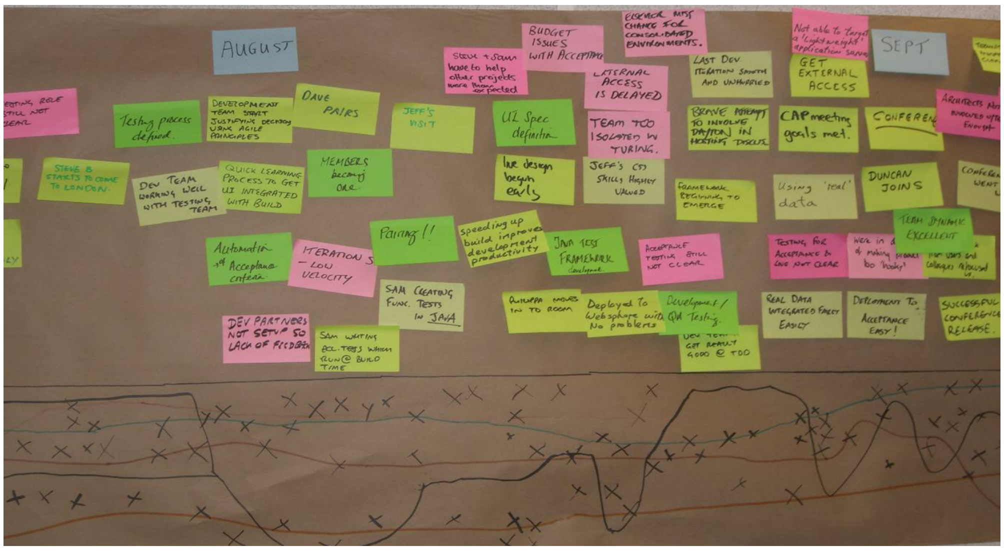

AUGUST
SEPT
Testing process defined
Dave Pairs
Jeff's visit
UI Spec definition
Team too isolated in Turing
CAP meeting goals met
Conference
Budget issues with Acceptance
External access is delayed
Get external access
Last Dev iteration smooth and unhurried
Members become ill
Dev team working well with testing team
Quick learning process to get UI integrated with build
Using 'real' data
Duncan joins
Team dynamic excellent
Automated acceptance criteria
Iteration 5 - low velocity
Pairing!
Java test framework
Sam creating func. tests in Java
Deployed to Websphere with QA testing. No problems
Real data integrated fairly easy
Deployment to acceptance easy!
Successful conference release
Dev partners not set up so lack of feedback
Sam writing acc. tests which run @ build time

Figure 1 - Retrospective Timeline with Seismograph

Once complete (Figure 1), the timeline forms a basis for discussions about 'what went well', 'what didn't go so well - with recommendations', and 'puzzles'. These conversations should be facilitated, with a particular emphasis on providing recommendations. I find that it works best if you ask each person to find their top 5 items (both well, and not so well) and then to rewrite them on a new 'post-it' with 'recommendations' stuck underneath. We then go around the group, one at a time, each presenting the most burning issue, and 'snapping' common items where appropriate.

Building on from this is the 'Top Tips' exercise which helps share information between different project teams in larger organisations. Rather than a facilitator gathering information from different teams, I pose the question to each team as the challenge: "You have just won the lottery and are catching the plane to Hawaii in the morning. What tips would you like to leave for your replacements so that they can continue your existing work, as well as be successful in any new projects?"

From these descriptions, you can see that an 'Alignment' retrospective has more varied exercises than its 'Heartbeat' counterpart, however what really differentiates it is that team members have the time and opportunity to comment on more than just process related improvements. As teams become more accustomed to working with each other (and this may take time, requiring several retrospectives) they start to take this as opportunity to talk about issues of team dynamics, personalities or failed approaches. For example one team raised the thorny issue of whether certain people were confusing refactoring with redesign, and using it as a mechanism to make un-agreed technical changes. These are much harder issues to confront however they open the door for real productivity improvements over and above the obvious process related issues that may be discussed weekly.

Common things that went well

Over the course of working with many different kinds of teams, there are many little gems that stand out. The following are some of the items that commonly crop up in successful Agile teams.

Good Team Dynamics

- Everyone worked well together, and 'gelled'
- Even when new team to the team - they picked it up very quickly.
- Not like other projects, always know who to talk to (and get good answers)
- Have a team that takes pride in its work

Handled Change Well/Good Agile attitude

- Even with unexpected changes in the project, handled it well and 'got on with it'
- When users changed priorities there were no complaints (patience)
- It encouraged users to focus on things in more detail

Standup Meetings

- Shared good information efficiently
- Having users involved in stand-ups helped better understood the issues
- A positive way to start the day

Pairing

- The transfer of knowledge through sharing works well
- It results in better code, you get the best of two people's designs
- The act of explaining helps you solve things more quickly

Good user Relationship

- Responded to users needs and still worked well together
- We could step in and support users (they loved it)
- Users were often pleasantly surprised with the results
- Good conversation around different options & flexible solutions

Great Continuous build environment

- Removed manual intervention – identical from dev to live
- Continuous deployment testing gave confidence
- Allowed rapid changes even at the last minute

Test Driven Development

- Writing automated acceptance tests for the website
- Separating concerns via Mock Object approach
- Made tests more readable (good names, clear intent)

Common Things That Could Be Improved

There are also many common 'gotchas' that you need to look out for. The following are some of the items that teams have also highlighted in their retrospectives:

Time pressure

- Acute pressure, always full on
- Learning new things is tiring too
- Not being able to complete full stories is stressful

Recommendations

- Ensure you do release planning, and visibly track progress of releases and iterations

Test Environments

- Too few environments available, shared servers caused grief
- Not enough access rights to servers (to automate things like data migration)

Recommendations

- Make sure other departments are part of the team, and they attend standup meetings

Business Involvement is more than you think

- Agile needs business involvement all the way through (it doesn't tail off)
- Keep getting trivial feedback until users really have to accept the full system
- Getting a formal commitment for user time (e.g. one or two days per week)

Recommendations

- Invite users to the standup / hold it at a time convenient for them
- Have a spare desk so that users can work in your development area
- Have more than one nominated user that can give feedback

Refactoring

- Is there dead code still left? Have we really done enough?

Recommendations

- Rotate in new staff, fresh blood helps identify opportunities
- Adopt a can do / won't put up with it attitude

Top Tips For Future Projects

While tips are often quite project specific, there are some useful items that re-occur between projects, for example:

Nominate more than one user

- So you can ask questions from more than one person as they will invariably be too busy when you need them

Talk to other Projects

- Ask for Retrospective Results from similar domains
- Speak to staff who have experienced real projects

Automate From Day One

Use source control for Operating Systems and Application configurations

Have a single deployment script

Do QA pairing

Futurespectives – Influencing Your Future

While retrospectives help you learn from the past, there is also an opportunity to use some of their results to influence your future. I came up with the idea of a 'Futurespective' at the 'Retrospective Facilitators Gathering – 2005', and was helped to develop this exercise by Debra Schratz.

I ask the team to imagine that they have stepped into a time machine and have teleported to a time just after the completion of their most successful project (which in reality is just starting). As the project was a success, the sponsors are keen to do a project retrospective, and so we are examining the future past.

I encourage the team to think about the many successful events which should be recorded on the project timeline (as green 'post-its'). Sometimes there are things that had the potential to not go well (recorded as red 'post-its'), however the team always managed to overcome those difficulties (i.e. any red 'post-its' should be followed by amazing green ones).

It's important to have fun with this description and use metaphors like 'Dr. Who and the Tardis', as this un-inhibits everyone's thinking which can lead to unexpected innovations.

Once the future timeline has been created, I then ask everyone to step back and mine it in a similar way to a normal project timeline, 'what went well', 'what can we do differently' and finally what actions should we take for the upcoming project. It can also be useful to get groups to tell a story that walks everyone through the events they consider significant on the timeline.

Conclusion

While technical problems are rarely to blame for project failure, many approaches to software development choose to overlook the important people aspects of delivery.

Learning cannot be rushed, and far from just running a quick team meeting every week, you need to periodically and systematically take the time to select the appropriate exercises that will allow your team to safely reflect on the true problems that it might be harbouring. The process of discovering these little gems can prove extremely rewarding both in terms of increased productivity as well as long term sustainability for the teams you have assembled. A learning environment is a happy and productive environment which is why innovative organisations are investing increasingly more resources into making sure that they learn from and incrementally improve the projects they partake in.

References

[1] T. Mackinnon, "XP - call in the social workers", in *Extreme Programming and Agile Processes in Software Engineering*, Lecture Notes in Computer Science 2675, M. Marchesi and G. Succi, Eds. Berlin: Springer, 2003, pp. 288 - 297.

[2] N. Kerth, "Project Retrospectives, a handbook for team reviews", Dorset House, 2001

About The Author

Tim Mackinnon is the director of Iterex Ltd. He facilitates future/retrospectives as well as coaching teams learning Agile Development, and writing and presenting papers for many international conferences. Tim also co-invented the 'Mock-Objects' testing technique and is a founder of XtC, a group which organises XpDay and meets weekly in London to discuss Agile development.