

Ten Tales of Positive Change

Aaron Sanders
Rally Software Development Corp.
3333 Walnut Street
Boulder, Colorado 80301, USA
aaron@rallydev.com

Abstract—Is it possible to make a difference? Or at least enjoy the interactions more? These tales relate successful attempts at making improvements. These are my experiences working in software shops and using Agile methods.

Software engineering; Process planning; Best practices; Knowledge management; Organizational aspects

I. INTRODUCTION

Suddenly I find myself at home. The last thing I remember doing was driving away from my current struggle at work and wondering, what can I do to clean up this mess? Shutting the door, putting away my keys and walking to the table I think about successful attempts from the past. It's easier for me to remember the challenges and dwell on what could have been. Replaying the drama and mistakes is not helping and so I refocus on the wins, no matter how small. When did I feel like an improvement was made, and why?

I reflect over my past six years of work in different Agile environments. The time starts at StorePerform, where I created software with a team using Extreme Programming (XP) development and Scrum. After that I went to a start-up called *vianet.travel* to work as both an XP developer and ScrumMaster. Then I moved on to Yahoo to be an Agile Coach; run some training, and embed on teams as a ScrumMaster. After that I trained and coached as an independent consultant. Now I am at Rally, working as an Agile Coach.

Ten tales stand out for me and what follows is a recounting of them. They are described mostly in the order in which they occurred, from my time working in Agile cultures. The tales are about:

- Creating the Motivation to Pair Program
- Staying Focused at Stand-up
- Keeping Progress High and Questions Low
- Reassigning Points to Validate Estimation
- Admitting to the Real Date
- Dealing With an Overwhelming Amount of Work
- Gauging the Rate of Progress
- Figuring Out How to Construct Teams
- Finding Predictability in the Velocity
- Allowing for Cross-Functional Teams

II. CREATING THE MOTIVATION TO PAIR PROGRAM

My first immersion in to Scrum occurred at StorePerform. We produced workflow solutions for big box retail stores. The company retained an external coach on contract and we attended training, which eased and accelerated the transition. Throughout the time we practiced unit testing and had a continuous integration server running.

A. *Keeping the Build Running*

Code coverage for us ran at about 30-35% and we had a company policy to keep it over 80%. We used a code review system to ensure, among other things, that unit tests were created.

The team agreed that fixing a broken build was highest priority. The person responsible for breaking the build was awarded a stuffed GIANTmicrobes® “macro virus” toy to display at their desk and wait to award to the next person who broke the build.

A successful build meant the code compiled and ran. Success also meant that all unit, integration and functional tests passed.

B. *Applying Test Driven Development*

Our manager entered work one morning and called us in to a room. He pulled copies of Kent Beck's book “Test Driven Development” (TDD) out of a large box sitting next to him and handed us each a copy. Our assignment for the day, he told us, was to go home and read it. Tomorrow we'd be coming back as test-first developers. It took awhile for it to sink in that he was telling us to go home and read a book, but we did.

We assembled the next day and discussed the book with our manager and each other. The conversation was moving at a pace that made me believe I wasn't alone in having read the book. We agreed to test-drive any code we created and discussed what we would need to do to support that kind of work. This would increase our code coverage, we hoped, and using TDD would help us to hit the 80% company policy.

C. *Getting the Same Result*

After a couple of months, our code coverage was still running in the mid-30% range. At the time we worked mostly solo and after completing a big effort, we would schedule a code review with a lead engineer.

D. *Creating the Motivation to Pair*

Now that we were apparently adopting XP practices, a couple of us felt like trying pair programming. Our manager supported the concept. We asked around if people wanted to pair and most were not interested. A big concern raised from a couple of the leads was that tasks would take double the person hours to complete. Some developers also believed that it would slow them down from cranking out work.

The merits of pairing were discussed as we continued to work in front of small, low-resolution monitors that were not well liked by and of us. Our accounting department saw high-resolution monitors as too costly for everyone to have one of their own. I asked my manager about the feasibility of approving just one. Could we justify it as a low-cost way to experiment with pair programming and manage the risk of work slow-down for the team? I hoped that folks on the fence would try pairing as the larger, higher-resolution monitors were so coveted by every one of us.

Accounting approved the purchase request swiftly. Upon the monitor's arrival, we set it up in a small room where nobody currently worked. The room we chose had a wall of windows that looked in to the building across the street from us. The room was large enough to accommodate two, or three, work areas. Setting up the workstation drew curious on-lookers who were told that anyone could work in front of the new monitor. The privilege required finding someone to pair with.

E. *Practicing Pair Programming*

Those of us who were interested created a schedule, with three two-hour pairing sessions each day. The demand for the sessions grew and pretty soon our manager was able to get a second monitor approved. We switched up the pairs each session and among four of us we had six pairs. We could all pair with each other every day.

Within another quarter we added two more large monitors. Some days our manager would also rotate through the pairs. We switched rooms so that most people could pair in the large room. The smaller room now became the space where people worked solo when desired.

F. *The Results of Pairing*

We found that code coverage increased every Sprint. We agreed to increase coverage by 3% a Sprint, instead of failing to obtain 80% coverage, and usually exceeded that amount. Pair programmers were not awarding each other the "macro virus" toy.

The time spent on task did not double as speculated and our velocity increased. We noticed a drop in bugs along with an increase in code coverage. Other code measures improved for our team, and it was getting quiet around the foosball table.

Work was becoming fun for us and we didn't need to play games for distraction. We did have to start enforcing breaks at the end of a two-hour session. Time was flying by as we worked intensely together and the breaks helped us keep energy levels high during the day.

Code coverage passed 80% and that level was added to the definition of a successful build. Less than one bug per week was being produced, and we fixed most bugs within a Sprint. We changed our working agreements again. We had fun building a system together that became easier to create and maintain.

III. STAYING FOCUSED AT STAND-UP

Having learned what Scrum and XP could do, I convinced the next organization I joined to try Scrum, with me as the ScrumMaster. The company built a reservation booking and payment engine called *vianet.travel*. Small proprietors with manual bookkeeping used it to advertise and rent out their places to people on holiday.

A. *Stand-ups Take Too Long*

Splitting my time as a developer and ScrumMaster, I was as likely to join in the conversation during stand-up. Stand-ups took too long, where we did more talking than we should. Sometimes it took us over an hour for everyone to answer the three questions.

B. *Noting Who Speaks*

Instead of employing a draconian measure like using a stopwatch and forcing time limits for each person, I decided to listen for someone responding to another person answering the questions. I paid attention to people who spoke for a long time. I jotted down in a notebook the names of those who spoke and what seemed to be the heart of the issue.

Nobody at stand-up said anything about my note taking. As I wrote, sometimes help was offered to the person speaking. Sometimes the team had moved on to the next person before I had finished writing it down. Otherwise, I waited for a pause and would remark that I had made note of who was conversing, and wondered if we could move on to the next person. At the end of stand-up I went through who was having conversations, in case people needed to follow up.

C. *Finding the Purpose of Stand-Ups*

Stand-up time decreased to less than 15 minutes and most people would leave to do whatever was next on their list for the day. Occasionally some people would find themselves in a follow-on session. These optional sessions allowed us to end on time, with the opportunity for some people to get back to work, while others could continue discussing the plan if needed.

Before long, I added a blocker board next to our task board. Sticky notes replaced my notebook for recording the impediments that came up in the discussion. At the end of stand-up I answered the three questions as a developer. As ScrumMaster I went through the new impediments raised and gave people a chance to verify or add others. I then reported on what impediments were removed yesterday, which ones were being worked today and what was in the way of removing impediments.

D. *Clearing Impediments*

This visibility allowed others to help me when they could, since I was splitting my time between ScrumMaster and developer. Seeing the impediments on a board allowed management to volunteer help with impediments that we as a team were unable to remove. Highlighting what was in our way made us want to remove them faster. Agreeing on what was in our way and how to remove it let us have the courage to attack the problem while respecting each other the entire time.

IV. KEEPING PROGRESS HIGH AND QUESTIONS LOW

A. *Everyone Wants to Know Status*

Vianet established a cadence around Scrum and soon our customer, stakeholders, executives and others would ask how the current Sprint was going. The people asking questions numbered more than we had on the team. The questioning increased as a Sprint neared its end.

I asked if people could hold off with the questions until the end of the Sprint. It had become too much of a distraction for the team.

B. *Designing Good Retrospectives*

A friend suggested that I read “Agile Retrospectives” by Esther Derby and Diana Larsen. The book was recommended to me to help weave activities together to get the most out of a retrospective. In reading it I noticed that the effect could be amplified by tying retrospectives together for another level of build-up, from one retrospective to the next.

It’s suggested by most Agile Coaches to only put in a couple of hours to prep for the Sprint review. I would invest up to three hours preparing for a good Sprint retrospective and another hour and a half to facilitate. I spent some more time writing up summary notes to help us remember decisions, and to help me design the next Sprint’s retrospective.

C. *Communicating Results*

At the end of the Sprint we would have our review and take feedback from our customer and others. When asked how we thought the Sprint went we could usually point out that we had once again met our commitment and if not, a quick explanation why not.

Further comments from the team, we insisted, would have to wait until we could have our retrospective. We decided in our retrospective what information we wanted to share out, which was mostly conclusions and next steps.

D. *Better Interactions*

Our team constantly improved from how we practiced retrospectives and we made a habit of relating afterwards how we felt about the Sprint. While we included feedback from Sprint review into our planning, we would also explain our plan for improvement that came out of the retrospective.

The questions dissipated and we could work without as much distraction. This allowed us to meet our commitment and celebrate it in the Sprint review.

V. REASSIGNING POINTS TO VALIDATE ESTIMATION

A. *Keeping a Good Agile Estimation Practice*

As part of our Scrum practice at vianet, we spent time as a team tending to the Product Backlog. We played planning poker and kept all our stories sized. We knew the entire size of the Product Backlog. Every Sprint we looked ahead and refined some User Stories.

Our velocity was stable and we could predict what User Stories would fit in a Sprint. The total size of the Product Backlog and the number of Sprints left projected that we would not hit the date promised to the customer.

B. *Verifying the Estimates*

The Product Owner started to question our estimates. He neither questioned the technique itself nor the merits of relatively sizing items, just the numbers we produced from this estimation technique.

We decided to spend a day with the Product Owner looking over the backlog. We looked at some of the work we’d finished to help calibrate sizes, and triangulated it with the items not yet started. Some of our User Story estimates changed. We sliced some of our User Stories thinner and even removed a couple. The Product Owner was convinced that our estimates were sound.

When we finished and totaled up the points, it was nearly the same as the beginning number. We learned that keeping the backlog sized is priceless and that reassigning points is useless. The total size of the Product Backlog had not moved a significant amount from all of our work, although we now were intimately familiar with it.

C. *Time for a Talk*

This didn’t help the fact that we all recognized that we needed to have a tough discussion with the customer. We couldn’t deliver on the date we promised with the features they wanted.

VI. ADMITTING TO THE REAL DATE

We established a trusting relationship at vianet with our customer. We invited them to each Sprint review and planning ceremonies. They checked out and deployed our code from our servers in to their environment.

A. *Negotiating the Date*

It was the end of fall and our customer said they needed the work finished by the start of spring. Everyone wanted some buffer in case something delayed deployment and we agreed to finish development by the middle of winter. This date was written in to the contract drawn up by lawyers and signed by both sides. The rate that we were now building the

product at had our team estimating we would have something out after the date in the contract.

B. Inspecting Progress

Our customer attended reviews, inspected progress by checking out and building the latest code at any time, and saw our responsiveness from requests in planning. It was because of this interaction that when we told them, they understood that we had more work to do than could be delivered by the contract date. Everyone believed the data they were seeing.

C. Working Closer with Our Customer

The customer then upped our trust in them. They told us that the intention was to go live in the early summer. Our velocity and Product Backlog size showed that we would finish most of the work by the end of spring. The customer worked closer with us, providing frequent feedback on the direction.

We worked together deciding on what to build next. People from both companies traveled between the two sites. Our Product Owner seemed calmer. Everyone understood clearly what to do and we launched within the window predicted. The customer eventually acquired vianet.

VII. DEALING WITH AN OVERWHELMING AMOUNT OF WORK

I located a job as an Agile Coach working at Yahoo! headquarters in Sunnyvale, CA. I worked with a group of internal coaches, offering both training and coaching for the nearly 500 teams that were experimenting with the Scrum framework inside the organization. My directive there was to observe teams in action and offer help when asked.

A. Observing Team Behavior

Pretty soon I was invited by a Product Owner to observe his team's stand-up. This team sensed that something wasn't right with their Scrum implementation and he was seeking advice. They had also recently lost their ScrumMaster and people were wondering what might happen to replace him.

Walking through the buildings from where I worked, I arrived a little early to the meeting and soon about 20 people filled the conference room. When everyone got there we quickly went around the room and everyone said something about what they were doing.

The tone most people used sounded like a status report. I couldn't detect a common thread woven through what people talked about. I looked around and did not see a Sprint task board anywhere. The meeting lasted for only 15 minutes and then everyone immediately exited the room.

B. Investigating the Situation

Asking around afterwards I found that the information stored electronically was out of date, having been maintained by the ScrumMaster who had left. Nobody was really thrilled with the electronic tool, something swiftly put together internally to support one of the first Scrum teams. Although

many teams used the free tool it was no longer supported nor maintained by anyone.

The Product Owner, a couple of managers, two leads and myself discussed an approach. We concluded that it would be best for me to embed as ScrumMaster for a while, perhaps a couple Sprints or more. This would allow me close inspection of what was going on. It would give the team some time to find a replacement and for me to hopefully work with that person.

Observing a couple more stand-ups I found it very difficult to discern what the expected outcome was of all this work that the team was doing. I spoke with people about how many tasks they currently worked on, and what those tasks were.

Most people worked on more than one Product Backlog item at a time. Some people worked on several items concurrently. Nothing had been deployed to production for over a quarter now. There were around 600 open bugs, and technical debt mounted with more being opened than closed.

C. Exposing the Work

After discovering all of this, I arrived at a stand-up with a pile of sticky notes. I passed them out and asked people to write down what they were working on, one task per sticky note, and to put how many hours remained. I then asked people to initial their sticky notes and place them on the white board. We clustered tasks together and wrote down what Product Backlog Items they belonged to. The resulting sticky notes covered two 4'x8' whiteboards on one wall and one more board on the opposite side of the room.

There was a shared concern that the team worked on too many items. Seeing it hanging on the wall all around them convinced everyone. The team decided not to take on any more work until the current work in progress diminished significantly. The Product Owner prioritized the list and provided focus for the team as most people had tasks for more than one item.

D. The Effect of Work Limits

We calculated how many items finished weekly. Looking at this number along with team size helped us establish a limit for how many items could be in flight at any time. Before long, the team finished one Product Backlog Item every two to three days and deployed two to three features per week. Hours were no longer tracked. The bug count went from close to 600 to several dozen and eventually down to zero bugs open by the end of a Sprint.

The highly visible task board continued to be modified by us and now fit on a wheeled 3'x5' whiteboard and stood where people worked. The Product Owner was happy with what he saw and I caught him once hugging the board. Having the work visible resulted in better preparation and understanding by the team of the flow of features desired.

VIII. GAUGING THE RATE OF PROGRESS

An interesting pattern was revealed to me. I noticed it in my embedded team and with other teams who asked for help. Product Backlogs were not being estimated.

Teams were not estimating in points, days, nor even time. It is unclear what led to this lack of discipline. A consequence of teams not being able to say when something might be done resulted in other people setting deadlines for those teams.

A. *Playing the Team Estimation Game*

Demand for coaching was high and we were always being called in to situations. In looking around if I found a backlog with hundreds of items and no estimates, I would try to schedule a session for playing the Team Estimation Game. To play the game I would print out the Product Backlog first and then spend time cutting out each item. I would then place 8.5 by 11 paper around a table with the planning poker numbers on them.

Handing out portions of the backlog to each team member, we would start the session by silently placing items into the corresponding point bucket. The bigger the bucket, the higher the number on a piece of paper taped to the table. Team members were free to move items from one bucket to another. The team kept moving some items around and we noted these for estimating with planning poker later.

B. *Consequences of the Game*

The intent of playing team estimation is to quickly size an entire backlog. These teams went through over one hundred items in less than an hour. The technique is not as thorough as planning poker. Done silently, team estimation doesn't invite the conversation found in planning poker. It does allow for a baseline sizing of the backlog. Team estimation can establish an historical base line velocity for a team if they estimate finished work as well.

Consensus is established quickly on the really big items and on the volatile items that need refining before they are ready for Sprint planning. It helps a team gauge the rate of progress and helps everyone conclude when to expect work to be done instead of forcing dates on people.

IX. FIGURING OUT HOW TO CONSTRUCT TEAMS

After the Agile coaching group shut down at Yahoo!, I decided to take a chance at being an independent coach. I quickly found work with a network management group. The business unit consisted of over 70 people, mostly working in San Jose and Bangalore. The Vice President of the technology group wanted to "burn all the ships", and hoped this no retreat attitude would help the team convert to Scrum within a quarter. I spent over nine months with the business unit.

A. *Creating the Initial Teams*

To begin Scrum, the managers created three large teams along architectural boundaries and HR reporting lines. Each team was around 15-20 people. Immediately, integration issues in the system and between the teams showed themselves. It was the norm that for one feature, at least one person from each of the teams would need to work on it. Stand-ups were taking a long time and I would see the same person in multiples.

B. *Using Story Themes for Organizing Teams*

I noticed that the Product Backlog was organized into themes and this gave me an idea. At the next planning meeting I tried an experiment. I wrote down each theme on a separate piece of paper. At Sprint planning I held one up and exclaimed, "If you work primarily in the area of the sign I'm holding, come stand over here." As people gathered, I handed them the sign, moved a few paces and held up the next piece of paper with another theme on it, repeating the process.

People collected in groups of two or three and as I got to the end of the list of themes I asked them to look around. Did they see any other groups and/or themes they worked with closely? People shuffled around a little and before long, we had six teams of around eight to ten people each.

There was one holdout group and once that was settled, we had a few people that still hadn't decided where they belonged. Finding them a place required a little management intervention. I then asked each team to think of a name, so that we could continue Sprint planning using the new names for our new teams.

I asked the teams if we could try this experiment for a Sprint and if necessary, people could switch teams in the next Sprint. A balance was reached between steady, persistent teams who felt comfortable in the code, and the capability to move people around to share knowledge.

C. *Benefits of the Team Restructuring*

Within a quarter continuous integration was in place. Another quarter after that and each check-in produced a disk image. The image included the operating system, all necessary third party applications, and packaged code the team built, installed on an appliance, which was then rebooted. Within another quarter all tests passed with each build and unit tests covered over 90% of the code base.

X. FINDING PREDICTABILITY IN THE VELOCITY

Being an independent contractor is exciting yet somewhat lonely, having to support myself and find every opportunity. I look to constantly improve my craft by finding others to collaborate with and have found those needs met by joining Rally. I now concentrate on improving my coaching and training practice with other coaches. We have a support system around us and I get to work with people who have different strengths and skills.

One of the clients I work with is a networking company. One of the teams I worked with had a difficult time establishing a consistent velocity. Forecasting when a release might be ready for system testing was difficult for this group. This holds back other groups who have to integrate with them at the system level.

A. *Leaving Defects Unsized*

The Product Backlog and historical velocity information showed me that the team was not sizing defects. When inquiring why that decision was put in to practice the answer indicated that story points were seen as a measure of value. After all, the team is only credited with the point value of a

User Story when the Product Owner has accepted the work as done.

The group sees defects as having no value and show that the original story was not really completed. We discussed if User Stories can be in an accepted state when defects exist for them. We talked about how to apply a formula to recalculate velocity and accommodate for defects. It seemed to the team to be a lot of work with little reward.

When prompted, the team came to the conclusion that defects looked a lot like User Stories and that they could see the relative size of them in comparison to those stories. Someone noted that the expected/actual language of a defect matched up well to acceptance criteria.

We discussed what makes up the size of a story, like how complex it is, and how much uncertainty we have about it. User Story size seemed to us to be correlated to the amount of effort the team needed to invest to finish a story. I admitted to having thought about story points as a value metric and had changed my mind, I now thought that they related closer to the cost of an item.

The team talked about what they believed and concluded that it would be worthwhile to try estimating the size of defects and see what happened. They already met weekly to triage them. Defects were stack ranked with the rest of the Product Backlog. It seemed to the team that it would require minimal effort to size them as well.

B. The Effect of Sizing Defects

Velocity stabilized with the team keeping all the items including defects in the Product Backlog relatively sized. The total point increase of the Product Backlog confirmed what most people thought; the projected release date was further out than first promised.

The predictable velocity allowed this team to now reserve a certain amount of point capacity to invest in automation and other infrastructure. Preventing defects instead of merely fixing them reduced their cost of feature deployment. This could be seen by them in different ways, such as the reduction of defects backlogged and less time spent in triage.

Understanding velocity showed data to help the team decide if more features had to be cut to meet the date, if the date would have to move, or if more people needed to be hired. In the end the organization acted on all of these options.

XI. ALLOWING FOR CROSS-FUNCTIONAL TEAMS

A home entertainment provider called me in to facilitate User Story writing followed by release planning. The business unit had not tried either activity before. Leadership had been replaced and the new General Manager and Vice President of Technology decided to invest in the group's burgeoning Scrum effort.

The organization followed a structured work breakdown approach with tasks functionally derived. People were willing to approach product definition in an Agile way and wanted to experiment with User Stories. We discussed the vertical nature of Stories and that teams were formed

orthogonally, based on the architectural stack. A few people pointed out that teams were split by functional area.

The group understood that co-located and dedicated teams fit the Scrum model. Yet voices grew stronger that teams were not set up to autonomously deliver end-to-end features. We talked about the merits of the cross-functional team and folks insisted that the group could not afford another reorganization. One of the Vice Presidents had declared that no "dotted line" relationships could be formed to create virtual teams. She found that such an approach diluted the commitment of individuals and made the process opaque.

We got in too some passionate discussions over the once taboo subject. I continued on with Story writing. We later revisited the team structure with executives prior to release planning. The decision to not allow virtual teams stayed in place. Release planning was put on hold for about two months and then they asked me back to finish. The VP had relented, and kept her job.

When I got back, the business unit could work in a matrixed organizational structure. Virtual Scrum teams had been created without going through another Human Resources led reorganization. We were able to proceed with release planning. The teams were organized where they could implement the features independent from each other yet stay coordinated. This helped increase their production rate while increasing the quality of features as well.

XII. CONCLUSION

What do all these tales have in common? They started out as an idea that led to a small change. These changes could be implemented easily, cheaply and quickly when there was complete buy-in from everyone that the change was necessary and for the better.

Of course, I didn't always find myself in a better place. One result had us building a product that became attractive for an acquisition, while our related labor cost did not. Many of us lost our jobs to promote the sale. Another consequence of success was having a senior executive in our organization determine that formally sponsoring Agile was no longer needed. We were invited to look for work in other parts of the organization for awhile, or had to leave the company entirely.

Yet there are times when I did help make things better. The best improvements I've been a part of happened gradually, required patience, and added up over time. This helps remind me that I can improve my current situation. I need to help the team see the problem clearly and suggest ways to validate proposed solutions.

If we have a shared perspective of the current situation, then we can agree on how to make it better. We will need to check in frequently, and constantly re-validate the understanding and approach.

ACKNOWLEDGMENTS

I would like to thank Erica Young, Ken Clyne, Ronica Roth, Ann Konkler, Bob Gower, Jeff Patton and Ben Carey for their comments, advice, and encouragement.