Clicker training & agile software development

The overlapping mindset of my passions

"Software and dog training
is way too important to afford
not having fun while doing it."

by Björn Tikkanen

Clicker training and agile software development

The overlapping mindset of my passions

Björn Tikkanen

This book is for sale at
http://leanpub.com/clickeragilebook

This version was published on 2013-04-07

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Björn Tikkanen by spreading the word about this book on Twitter!

The suggested hashtag for this book is #clickeragilebook.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#clickeragilebook

Contents

Background

“Do not go where the path may lead, go instead where there is no path and leave a trail.” - Ralph Waldo Emerson

Two of my big passions in life are dog training and my work as software developer. This is also what I spend the majority of my time with. Being a contractor in the IT industry gives me the opportunity to work with software, processes and human interactions at various levels and at different clients. For simplicity, lets call this *agile software development* (which I will explain more in datail later). Clicker dog training (or maybe it could be called agile dog training) has a lot in common with agile software development. In this text I will try to share my view of them both, including similarities and differences.

Earlier in my career both as dog trainer and in the work situation I felt that I did things that I didn’t want to do. And I did not like the way I did them. How things were done really didn’t resonate with me as a person and that rarely brings good results. Changing training method and leaving my previous employment were two things that made the situation ways better. Now I often still feel frustration, but mostly about not being able to do better. I would like to do more good things, help others more, progress faster and be happier in what I do. I would also like to see people around me much happier. I call this *positive frustration.*

Before you start to wonder if it's about the dog sport "agility", I have to say No. Agility in dog sports language is not the same as agility in software development terms. They do have some basic core similarities but since I don't practice this sport I won't talk anything about it. I will instead assume that the methods used for obedience training are also suitable for agility as a sport.

I have been working with software for over 15 years and with dog training for about 10 years. Sometimes when you face a situation it feels like you are totally new to it, but sometimes you feel you know the answer to any question by heart. In both areas the learning and self-awareness curve is about the same. At first you learn the basics and you think that this area is not that hard to master. When learning more about the theory and what it means in practice (with many of the grey zones), you start to realize that the things you do know is just a small part of the whole. Another thing is also that learning is really important!

When we got our first dog in year 2002 we started to look for a training method that would be suitable for us. And at the same time suitable for our dog. We didn't start using clicker in the training from the beginning, but pretty soon we discovered this way of training dogs. At this time, I must admit, I did not fully embrace or understand the clicker method, and used the clicker mainly for positive reinforcement. Since then, I have come a long way and I now use the *clicker method* (as described later in the book)

in more and more situations.

When I started to train our second dog, it was pure clicker training already from her puppy days. Today, when coming to something new I want her to perform, I often start out by using the clicker method. But I always evaluate the situation and see if there is some other way to reach the goal, in a better fashion. Sometimes there is, sometimes not. You see, I also try to be quite pragmatic in just about everything I do. It is no coincidence my company name is pragmatiX.

My software development career has been more of a roller coaster. I think I was quite late at getting acquaintanced with agile methods but after that I was quite fast adopting them. The problem with working as an contractor is that you can't always choose your own ways of doing things. Sometimes you find yourself in a position and place (workplace) where things are just not the way you want them to be. At times you can change things for the better and sometimes you just have to tag along for the ride. A ride with the roller coaster that is. It might take you deep into agile land or far away from it. I've seen both and I've seen transitions in both directions.

So, who should read this book and how? Well, anyone interested in either clicker training or software development can read parts of the book and find respective parts interesting. And if you are interested in both, then I'm happy to have found a "friend". The parts on agile software development and clicker training are not going to state any

revolutionary things that you can't find in other books. They are more meant as a way of orientation, to ensure some basic knowledge, before what the book is really about starts – the comparison between the two. So, if you're in for a quick read, feel free to skip to this part right away. Yes, then it will be a really quick read but on the other hand you might miss some of the context and personal view that I try to establish in earlier chapters.

When reading the rest of this book keep in mind that some things you read is my own opinion and not facts. Most of the things I write about in the chapters on software and clicker training, I have taken from what I have learned from others and from research during the years. Even though I do not point out every time I use someone else's thoughts or my own opinion, I hope you can sort this out by yourself.

At some point I started to think cross the borders of these two domains and to get some of the thoughts down "on paper" I tweeted them. I think I started during one of my many evening walks with my dog, where a totally different state of mind than the usual often appears. I used the hashtag #clickeragilebook, mostly to keep track of the ideas, without really imagining that I would actually do anything about it. Some of the tweets will also appear in the text where they "belong", so the connection is still there. Most of the tweets are also collected in an own chapter. Part of the rationale for writing this "booklet" (short book) is therefore to collect my thoughts and also to challenge myself in thinking a little bit harder about

all of these things. But also to maybe spread some of the visions I have about the workplace situation and the hobby of quite a few dog training teams (meaning the trainer and the dog as a pair). My vision is that we should find both work and hobbies challenging but also fun and meaningful. What I can see almost every day at work or during dog training is people that struggle with both themselves and their environment, making the situation far from fun and rewarding. What you will find in this book is by far no prescription on how to solve this, but maybe some thoughts that can help on the way.

In both work and dog training I try to think holistic, in that Happy dog trainers in Harmony with their dogs are also Happier in their working situation and the rest of their lives. And the opposite of course. So, by giving both parties tools to work with, I hope that change in the one end can lead to change in the other. Give people the power to solve problems and they will show amazing results. Lovingly train your dog and you will earn trust and loyalty, together with hopefully a successful partnership in the training. Be present with and care for your team and they will give everything and more back in return. It is by no means a zero sum game. Played correctly, we are all winners.

"Caring for eachother is not a zero sum game. Played correctly, we are all winners" (Tweet)

When I write "we" in the chapter about clicker, I often mean me and my dog. Sometimes I refer to her as "her" and even sometimes "the dog". But most often "the dog" is any dog

and not precisely her.

Similarities

The reason to write this book was not to teach you about agile software development. And it was not to teach anyone about clicker training. It was about my feeling that these two have a lot in common and that they with respect to this both fit me well. This chapter will focus on looking at different parts of them and finding where they share *principles*, *values* and *methods*. For me this is what this booklet is mainly about.

"Clicker method and agile are alike in two ways. They are often both counterintuitive and "common sense" at the same time #clickeragilebook"(Tweet)

Inspect and adapt

Doing things in small steps or small batches creates a need to often look at what is happening. Then take action to be able to do better or totally different in the future. In agile we call this *inspect and adapt* and the very same thinking is also found in clicker method training.

In software we look at both what we have created (working software) and also at the state of our process (how work works). It is often easy to check the software part, but maybe even more important is the way we look at and improve our process. In many cases it really does not matter

what you start with, just that you get better at getting better all the time. Then you will soon be good (enough).

In clicker training we have our exercises that is the equivalent to the software. These are the ones that bring us value, at for example competitions or in our normal daily life. But we also need to improve our ability to train and have fun, to be able to learn new things easier. Otherwise it is easy to get to a certain level and don't have the power to lift from there. So, we *inspect and adapt* both with respect to the exercises we want to perform and the way our training sessions go.

Short feedback loops

Working in short steps with short plan create a need for feedback loops often. We want all this to happen in both areas. Most agile methods or way of working have a rhythm where one of the main objectives is to look at the current situation and feed the feedback loop with valuable information. As mentioned earlier, the feedback loop should be active all the time, but at the very least at the heartbeats of the rhythm.

One of the important parts in software development is when we at regular (or irregular) intervals show the current working software to the customer. This is a good point for checking if what has been done is what was wished for by the customer. If not, let's go back and do it in a better

fashion.

The equivalent in clicker training is when we try out a whole exercise in an environment that is not the usual training one. For example, this will include more disturbances in form of other people, dogs or the environment around us. This will give us a better understanding of (information) how well we know the exercise. If it is really simple to do something in training setup but not with more things happening around, we know that we need to focus more on this. Just doing more of the same thing probably won't solve the problem, even though "overtraining" partly helps.

"If it hurts, do it more often" (Martin Fowler)

All the short plans do not have to be plans for success. Some can be plans for learning something new about your team (software or trainer/dog). If we do this in safe and short steps, we don't build negative stress during the testing or when it fails. It is ok not to succeed all the time and we "celebrate" when we learn something new. Some parts of software development call this "pivot". This means trying something completely new and unknown to learn things that might have a magnitude of positive effects.

Clicker training (and evolutionary change?) is more about finding the bright spots than identifying the weak ones (Tweet)

Both are about planning and then testing what you planned. Make the plans short and the time between planning and

validated learning short. After learning, make a new plan and start the new cycle. When we keep our plans / experiments small, the risk is often quite small as well. When we try larger things we have to acknowledge that the risk is higher. We probably won't do this if we don't have a large problem we need to solve urgently or if we can afford to take a loss or reduction in functionality.

Focus on quality

It is easy to cheat oneself and think just about delivering new behaviours (in clicker) or new functionality (in software) as fast as possible with no regard to the quality. But both in agile and clicker we really care about the short term as well as the long term quality in what we do.

Almost every piece of software will be connected and/or used by some other part. Then having created this one part fast and sloppy will lead to other parts, or the whole system to work at a lower capacity or even with worse functionality. Therefore we "build quality in" and secure everything we are building as soon as possible. Having a sound foundation will most likely create more smiles and less curses among the people that work on buildning the system. The same probably holds for the people that are going to use the system.

In clicker training we work hard to lay a good foundation in learning all of the basic skills with good quality. If we

cheat here we will have a harder time to work with these in chains later. Then the risk is high that we will have to go back and re-train the parts over and over again to be able to use them. If we compete with a dog and advance from one class to the next one, many of the exersices will be kind of the same but with more and tougher parts. To have lower quality in these exercises then is a ticket to Failureville. Or at least a bigger challenge.

Learning to solve problems, not just hiding them

Often when we see a problem we try to solve it. But almost as often the problem we see is just an symptom of the real (root) problem. Solving the "symptom" will most likely be just compensating for the real problem, that will still exist and the symptom can reoccur at any time. Instead we should try to find the root problem and solve that, even if it takes more time than solving the "problem" (symptom) we see.

When things don't go as expected with our dog training it is easy to look at just what we see. But thinking a step further might help us quite a lot. For example if something disturbed our dog during an exercise we can go on when the disturbance is gone. Or we can try to get better at working under strange circumstances and be able to perform even when things are not as usual. Or when

the dog does not fully engage in the exercise, we can either try to "force energy into it" or we can think about what led up to the dog loosing part of its energy. Solving the root problem will probably make the symptom occur more seldom while treating the symptom might even make it appear more often. Yes, we often amplify the problems when we really are trying to remove them by working directly with the symptoms.

When we have a problem in software we also often just see the symptom. It might be that we create bad software with many bugs. The easy conclusion is that we just are bad programmers that write bad code. But looking under the surface we might find things like bad requirements (what the customer wants and how it should be done), bad tools, non working process for early testing or some other process shortcoming that leads to our failure in getting high quality software to our customer. Instead of focusing only on the place where the problem manifests itself, we can greatly benefit from solving the cause of the problem instead. To really find the root cause is a challenge in both dog training and software development, but it is something that probably will pay off.

The power of self organizing

"Every time I tell someone what to do I have failed. Keep the failures small and unexpensive" (Tweet)

Having problems solved on your behalf can sometimes be very nice. But when served with solutions all the time, the team won't think for themselves when facing a new situation. They will rely on some other part to help them out. Showing how or greatly helping the dog with exactly what to do all them time, will have as effect that it won't try for itself. It will instead wait for orders on what to do. Even though it is tempting for a manager to point out a solution to the team when something needs to be done in i rush, it is better to stand back and let the team solve the problem. Doing one or maybe a few quick fixes will quickly turn into a habit that is soon the way to work. This will lead to *help dependence* in both areas and that is the opposite of what we want to achieve.

Leaving the control to the team or the dog can help us avoid them relying on someone else to hand them solutions. In the long run it is far better for the individuals if they learn how to solve the problems by themselves. This applies to both software development and clicker training with dogs.

We say that software teams are self-organizing around their work, which means they are given an assignment, trust and the freedom to solve the problem in the way they find most suitable. This is letting go of control and trusting the team. Self-organizing as a team term is mostly about having the freedom to select how to organize around solving a problem. There is no authority that controls the way the team works and that tells the team on what and how they should improve. This hopefully leads to teams

that take pride in their work and responsibility in all that they to.

When the dog is to do something we can either show it how to do it (much direct help) or let it solve the problem by itself. In the name of clicker training we select the latter and by this we hand over much of the control to the dog. The dog will probably take the chance to exert this "sense of control" more often and that is also our goal.

The dog should feel it is not only following order, but also have "a saying" in what is happening. By letting the dog learn by trying things itself, this is also possible. A positive side effect is also that the understanding and acceptance of the exercise is better when done at free will.

"Clicker method replaces command & control, leadership and threaths with trust, curiousity and empowerment (to try)" (Tweet)

A parallel between old workplace and dog training thinking is of course the Theory X[1] where the belief is that people don't want to do a good job unless bribed or threatened. Here we also see the strict hierarchy that is built, managers that master the workers, as a means to get the job done. If we use this in the workplace, why should we not use the same method for training our dogs? Threats, punishments and at occasion a bribe or treat. Self organization and distrubuted leadership is one way of turning this around.

[1] http://en.wikipedia.org/wiki/Theory_X_and_Theory_Y

Progressing with baby steps

We want to use the same sort of very short feedback loops in both agile software development and in clicker training. In both we want to make a minimal effort to get to the end goal, but at the same time we want to get all the parts good. This means that we look at quality of the parts as well as quality of the whole all the time. We often refer to this as "building quality in".

To make this happen we do things in small steps, baby steps one might say. Every unit (small problem) we test in our software is like a basic skill in our clicker training. We need to make sure that both are in good shape, fit for building on and do not regress to a bad state. We do this by exercising them often and making sure they maintain the standard or the criteria we have on them. The same thing happens during our shaping exercises. Then the criteria is set for each stage and we work in really small steps. Before we actually know exactly how it will look, we ask the dog to take the first step. If it is right, we build upon that. Pretty much like a ping pong game.

When we put the parts together in dog training we do this by chaining. Either forward or backward. When we have successfully chained a few parts together we add more and more until we have the whole drill. Every part is "tested" with the criteria we set out from start. We only continue the chain if every part is good enough and by that also securing the parts of the chain at the same time as we secure

the whole exercise. And guess, we do the same in software but there we call it integration testing. In this part we test "chunks" of software that may or may not have been built by the same team or people. This linking things together is one of the hardest parts in both software and dog training (if you ask me).

In software we also talk about Behaviour Driven Development (BDD), a technique where we instead of testing the small parts, test the whole. That is, the whole behaviour of a specific service or part of product. By talking about what we want to see at the end we have something to aim for, not just a set of parts to build separately.

To be able to test your own and your dog's status you can either compete or participate in an competition like training event. Training is training and competition is competition. You might think that the difference is not that large but when nerves, emotions and whims come to play, strange things can start to happen. Therefore many of us need to train under conditions that are competition like to lessen the chock when at a real competition. This is testing for "production".

"Clicker training is all about info discovery to find out what to do (do the right thing) & then a touch of doing it right" (Tweet)

Don’t call us, we’ll call you

In a restaurant you have a few different options of how to get your food. One options is to have the waiter come to the table with the food. Another option is to have a buffet where you can select what to eat and when to get it. The first option is a *push system* and the other one is a *pull system*. When you pull something you do it when you have the time and capacity and most of the time voluntarily.

To push things, or force them, on your team or dog rarely brings a good result. To use pull is to hand over the control to the other part of when (and what) to do something. In software we say that teams pull their “working orders” from a list of work to do, whenever they are ready to take on more work, not when someone else says they should do it. We want to create high motivation and just going through the motions is not an option. I believe that we can benefit from using a pull approach in both software development and clicker dog training.

So, pull can be used for both dog training and software development. If the dog wants to do something and does it voluntarily and when ready for it (pull), it will probably have a better result than if forced to do it. Taking the exercise away will here be negative for the dog and doing the exercise a reward in itself.

Working software / behaviour

Delivering value for someone is important in both practices. That is why we focus on working software and competition / situation ready drills. Even when we start out with something new we want it to be meaningful or helping us learn new things already from the start.

Sometimes this makes us start working from the end. An example of this is in backward chaining where we sometimes start with the very last thing of a chain. That can for example be sitting at heel, doing nothing else then that. And since it is the end of a chain and where the dog has to be when finished with the whole drill, it is valuable. To get it all to work of course we have to add what led up to this position. This could be sitting 25 meters from the trainer and run to him at signal, ending up with sitting at heel as trained earlier. Getting to the end state could be said to be the most important, so we focus on that first.

Another example from software development is if we are building for example an online shop. When done, it will consist of a web frontend and a backend server. The web frontend is what the shopper will see when browsing the store. The backend server will hold all the data such as shopper information and the items available in the shop. Here, instead of starting with building a full backend with all functionality we will instead start with the customer facing part, the web frontend. Now we can test assumptions like if customers will be interested in purchasing from our

shop on a few customers and see if they were valid. If nobody of the test customers understand how to navigate in the shop we can change the design and try it out again. This is easier with a frontend connected to a minimal backend than if we had built all the functions in the backend already.

Coaching, not managing

An agile team does not need managing as written in the old books. They need a purpose, support with solving problems outside their reach and maybe a coach for further growing.

"By punishing someone for failing they will likely fail less often. But also try less often" (Tweet)

The same goes for the dog. Controlling the dog too hard will be counter productive if we want it to try out new things and feel good about it. But coaching gentle is in my mind a bit different in the meaning that it looks more on the "what" than the "how". I guess the "why" does not apply that much in the dog-training situation. So, it is more about getting the dog to try doing (anything) than forcing it into a certain thing. At least when talking about the coaching part.

"I don't want to teach my dog a single thing. But the list of things I'd like her to learn is long. And prioritized" (Tweet)

If you see a problem with your team or dog, I think it is a good thing not to make a big deal about it. Maybe even not saying or doing anything at all about it. If they who have a problem see it by themselves they will be much more committed to solving it. And it will probably be solved in a better way. If we tell someone they have a problem and also pushes a solution, the chances that it will backfire or at least not work smoothly is pretty high. When having a dog related problem we will try to focus on what is the right thing to do in that situation, instead of focus on what is going wrong. Then by rewarding the correct behaviour and ignoring the incorrect one, we will put the correct one higher on the reportoiare of the dog and extinguish (let become forgotten) the other one.

Communication

Both are very much about using the right level of communication. Finding a high bandwidth communication that suits all parts is a winning factor in an agile environment as well as in dog training. Communication is also about how we should treat eachother. Treat others the way you would like to be treated. And it starts with how we communicate. When a team can communicate their feelings and are honest about them, they will probably also have a higher bandwidth when talking about how to create value. One way of speeding this up in the team is when team members dare to show themselved vulnerable in front of each other.

Clicker training is also about fast communication and fast feedback. We want to create a quick flow in our exercises to get the *speedy learning* that BF Skinner talks about. And it is not about a one way communication just from trainer to dog that needs to take place. The trainer also need to see or catch the "questions" or "statements" in shape of for example body language or energy level coming from the dog.

To the masses

"The clicker can be (and is) as misused in "clicker training" as scrum in agile. Training with clicker != training according to clicker method" (Tweet)

Both techniques have now reached a broader user base. At the same time many of the practitioners use the methods in a way that was not really supposed. In many cases even not for much good. For example, many people say they "clicker train" their dog just by using the clicker for reinforcement. The clicker method is about so much more and the mix-up is a bit unfortunate. The same thing happens for agile methods where companies say they are doing for example Scrum[2], just by going through the motions of the process, but not embracing a agile mindset.

"Clicker training and sw dev can both be run by the

[2] http://en.wikipedia.org/wiki/Scrum_(development)

book and still suck. Rules seldom triumphs passion and motivation #clickeragilebook" (Tweet)

Mindset

"Take the time and effort to reward attitude / mindset and not just the success that was the result of the behaviour (Tweet)

Passion and creativity replace obedience and compliance. Earlier methods in both areas where much about command and control, forcing things through and trusting that building the parts right would make a good whole.

Now instead we look at ways to tap into the vast potential of indivudals that by free will take the lead to deliver value in the best possible way. This best will be different from time to time and found by agile teams during their voyage. Therefore curiosity, trust and continuous learning are pillars of mindset common to both areas. Another important factor both when thinking about ethics and effectiveness is respect for others. People that get respect from their peers will act more genuine in return, creating a positive spiral. The trainer wants the dog to respect him and this starts with him respecting the dog. This is done via creating a safe environment and good communication channels. Some would argue that respect is fear, but I say it is the opposite.

Not tools

Neither folks in a software team nor dogs are just tools used to reach a certain goal. At least not in my opinion. I see them as our training companions and workplace peers.

If the boss does not treat the people in their teams as equals and make sure they have a meaningful job and work environment, I think a lot of human potential is wasted. Companies that are profitable are good, but even better with profitable companies where people prosper and are treated with respect.

The same things hold for dog training. At least I don't see my dog as a tool that I use for having fun or winning prices and glory. I see her as my training partner that together with me have a meaningful, fun and giving leisure activity. If we win prices it only means we are working well together and can get on with even more challenging tasks. That's why I don't want to use a training method that does not mean fun for both of us. I rather move more slowly or even get less far than using methods that I find unethical or not fun for the pair of us. Or even quit before putting success over friendship…

"**Software and dog training is way too important to afford not having fun while doing it**."

Focus on the end result

Even though we split things up in to pieces in both areas we look at the whole pretty much all the time. In software we have the end goals from the user and in clicker training for competition we have an image of how we want it to look at a competition. It is really easy to loose the whole at times but disconnecting from it can cost us a few surprises. That's why we in software have acceptance tests and in dog training we try to chain whole exercises and sometimes the whole competition program.

The dog trainer should set up criteria for the dog and then hold the dog accountable for these. This does not mean any kind of blame or physical punishment, only that the correct repetitions are the only ones to be reinforced. The trainer should also hold himself accountable for everything around the training and to take into account all the circumstances that can effect the dog at the given moment.

Marine corps and Ninjas

To be able to do different things and to do them good you need to be good at a range of different things. Often many of these things are really simple in isolation. On the other hand you need to be really good at performing them so that you can do several of them in combination when it really matters. Two groups that practice (or practiced) this are

Marine Corps (picked this up from conversation with Jabe Bloom) and ancient *Ninjas*. These two groups are known for very special training to be ready for their very special assignments.

I don't say that we in agile and clicker training should prepare for war, but learning the basics so that it can be performed backwards in the middle of the night won't hurt. Therefore people dedicated to good software (which we generally are in agile) practice programming in so called *Code Katas*. Yes, the name even reminds us that this relates to what the Ninjas did back in the days. By fluently be able to solve different kind of problems we will create an opportunity to do other improvements instead of spending extra time trying to come up with a solution. Having trained over and over again will keep us from forgetting about the problem solving skills even when the heat is on. And trust me, the heat will be on!

In clicker training the *basic skills* are where we spend our *kata* time. Being really good at the basic skill of sit in one situation will also make it easier to get it to work well in another situation. This could be for example a quick sit while in motion. Having done the homework well makes the rest much easier.

Finding the balance

To succeed in respective area we need to find the delicate balance between different "values". We want to be able to work close to the border to chaos without falling over the edge, at the same time as we want to have enough structure to be in control. We need to be really creative at the same time as staying within the borders of what is allowed. We want to try to do the impossible at the same time as we need to be realistic.

Sometimes things get too intense. In some implementations of agile frameworks (for example eXtreme programming) a technique called *pair programming* is used a lot. It basically means that people solve problems together, working very closely and intensively together. When doing it right it can bring great results, but on the other hand, it can sometimes wear people out from being too intense. As most other powerful things, use with care!

In clicker training we want to have different levels of stress at the dog to be able to perform different things. In for example a situation where the dog is to run fast, we aim for a higher stress level than in a sitiation where the dog should stay put for several minutes. Trying to get the dog to the right level of stress can sometimes be too stressful for both the trainer and the dog. We want the dog to be both in a good stress level and also concentrated, and finding this balance is not easy.

Positive reinforcement and use of the clicker

"I wonder if merely positive reinforcement clicker style can help change team behaviour at work. Negative punishment, positive reinforcement" (Tweet)

Clicker method is all about positive reinforcement so I don't think I have to make a big case for it here when talking about dog training. We show the dog that it is doing the right thing by some kind of reinforcement, and by adding something it is positive. This is the core of the clicker method.

But how does this apply to software teams. If you thought that I would describe a best practice for clicking people on an software team I'm afraid you will be disappointed. Totally. Sorry.

Positive reinforcement, if done right, will likely give positive effects in form of connection in the team. But when working with humans this is probably not really accepted since it can be called manipulative. What is better suited in a team is genuine respect and honesty with each other. This leading to people expressing their feelings with maybe the same words as an "positive reinforcement conversation", but without any thoughts of manipulation. So, even if we don't use the clicker method or positive reinforcement in the same way we can still have the same mindset but on another level. The key is doing all these things together

with the team and not one person manipulating the rest.

Summary

When we *trust and respect* the individuals we work with we can *unleash the full potential* of our relationship. This is not merely a sign of niceness, it is a cornerstone of success in both work and training.

By stepping away from command and control and towards *self organization*, we can unleash often hidden potential. Leave as much as possible of the "how to do things" to the ones performing the actual work. In software this is the team. In dog training it is the dog. Letting go of control might be hard, but is in retrospect often much rewarded.

Hand in hand with self organization comes the need for *short feedback loops*. The team or dog needs to know if they are heading in the right direction. Without this important feedback, self organization often fails. The *goal* is what the team form around and work towards.

We aim to do the right thing in the right way. That is, doing what matters to someone and also doing it good. By using short feedback cycles to gather information about how well we are performing, we can learn and adapt to better ways of working. We call this *inspect and adapt*.

To succeed with the *inspect and adapt loop* we need to be open and honest with information. This is much easier in

relationships with a high level of trust. A good information flow that works in all directions enables fast and reliable communication, which is at the hearth of both clicker and agile.

When we have the things described above to build upon, we reduce the fear of failing. We are looking for everyone involved to take part in making the result awesome. This might be done by boldly testing something that turns out completely wrong. Doing this with a mindset that wrong is one step on the way to right, helps us learn new things along the way.

I believe we can achieve greatness without forcing, threatening or using short-term solutions with long term negative consequences. Instead we start with small improvements in our work that reinforces learning, and by extension, better performance. This way of reaching better performance while helping people and dogs to grow, is one of my dreams. Will you join me?