codemanship

Case Study

Peer-led Learning and Software Craftmanship At The BBC

Peer Group Learning and Assessment is a new approach to helping teams of software developers to gain new skills - say Jason Gorman and Kerry Jones

It's an experimental coaching method that has had some early successes in a small number of pilot programmes at progressive organisations. Here they present two personal experiences of an initiative taking place in the TV Platforms section of the BBC, in White City, London.

Introduction

Peer Group Learning and Assessment is a new approach to helping teams of software developers to gain new skills and to learn to apply those skills on their projects. It's an experimental coaching method that has had some early successes in a small number of pilot programmes at progressive organisations including the BBC. Here we present two personal experiences of the initiative in TV Platforms at the BBC in White City, London.

Why Peer-Group Learning & Assessment?

(The External Coach's View - Jason Gorman of Codemanship)

I've been helping software developers to write "better code" since I went freelance in 1997. My experiences working with hundreds of programmers across a wide range of application domains and levels of ability, as well as my own personal experiences of learning disciplines like OO design, test-driven development and refactoring, have led me to some useful insights which I'm now applying with clients like the BBC.

The programmers who learned and improved the fastest were invariably the ones who genuinely wanted to learn. The ones who gained little from coaching were the ones who weren't interested or motivated to learn. The key factor then was not how good I was at teaching programmers to do things the way I thought they should be done, but how motivated they were to learn in the first place.

I resolved to try a different approach. By focusing on trying to inspire programmers so that they might want to learn, and on fostering an environment in which anyone who wants to learn and improve doesn't just feel, but in every practical sense, is supported in their efforts, I'm finding that the results are very encouraging indeed.

Teams and individual programmers need to take ownership. They must want to learn. They must want to write more maintainable and more reliable code. Programming computers is a practical skill, just like playing the guitar or riding a bicycle. For sure, there's knowledge that can be imparted by coaches like me. But ultimately knowledge isn't enough. Understanding how to ride a bicycle is not the same as being able to ride a bicycle. Knowing where the chords are on a guitar's fretboard is not the same as being able to play a song using those chords.

I've worked with some very knowledgeable programmers and been shocked and dismayed upon reading their code. There's a world of difference between knowing what we should be doing, and actually doing it. The sad reality is that many programmers who say they do TDD and refactoring, for example, really don't. Not habitually, at least. From my own experience I know that programmers who aren't in the habit of doing things like writing a failing test before they write any new code, or always looking for duplication and refactoring it away, will almost certainly end up with poorer quality software. And how knowledgeable they are makes little difference if that knowledge doesn't translate into good habits.

Peer-Group Learning and Assessment (PGLA) is a non-traditional approach to helping programmers and teams learn and improve at core disciplines. It's built on some basic tenets:

> People learn because they want to, and people learn best in groups where everyone else is keen to learn
>
> Building habits over longer periods is more effective than imparting knowledge in the short-term
>
> The best test of whether a juggler can juggle is to ask to see him juggle
>
> Organisations that offer practical support for learning, as opposed to just lipservice, will attract and foster better programmers

The software craftsmanship movement

Key Points

* The programmers who learned and improved the fastest were invariably the ones who genuinely wanted to learn
* Peer-Group Learning and Assessment (PGLA) is a non-traditional approach to helping programmers and teams learn and improve at core disciplines
* People learn because they want to, and people learn best in groups where everyone else is keen to learn
* Building habits over longer periods is more effective than imparting knowledge in the short-term
* The best test of whether a juggler can juggle is to ask to see him juggle
* Organisations that offer practical support for learning, as opposed to just lipservice, will attract and foster better programmers
* The programmers who have successfully passed a peer assessment in "apprentice-level" TDD really can do TDD
* Showing how we develop software, providing constructive feedback and learning from our mistakes is essential for the development of software engineering as a discipline

has many schools of thought about how best to instill good habits and nurture professionalism and responsibility among programmers. Peer-Group Learning and Assessment is very much from the "learning by doing" school of software craftsmanship. There's no deep theoretical foundation or philosophy behind it. It simply groups together programmers who want to learn, helps them build a concensus about how they think something should be done, gives them a framework for focused and mindful practice over an extended period of time, and helps them to slowly but surely build up good habits.

It also helps them to test their habits in a more meaningful and practical way, leading to a kind of peer-to-peer certification that addresses some of the shortcomings of traditional skill assessment techniques.

It's still early days, but I've been very impressed with the results we've been getting. The programmers who have successfully passed a peer assessment in "apprentice-level" TDD really can do TDD. They're in the habit of doing the things they agreed in their groups that they should be doing. They write the assertion first. They always run the tests after even the smallest refactoring. They don't tolerate duplicate code and refactor mercilessly. They always run the test to make sure it fails for the right reasons. And so on. Each group has their own habits, and therefore each group has mastered their own school of test-driven development. It's theirs. They own it. And that's what really seems to make the difference.

The pioneers who went through Peer-Group Learning and Assessment in TDD are now tackling the much more challenging discipline of refactoring. They have selected a matrix of common code smells and commonly-applied refactorings and will spend many hours over several months seeking out those smells in their own code and practicing their refactorings to eliminate them, all the while applying a set of good habits they have agreed to in their group like "use automated refactorings wherever possible" and "make sure there's adequate test assurance before refactoring any part of the code".

Meanwhile, those same pioneers have become TDD coaches, helping new peer groups to define their own school of TDD and to build up the good habits they believe will lead to better code. This seems to increase the buy-in and ownership of the coaches to their own school of TDD. It also gives them ongoing practice at TDD to help maintain these habits, and from a new perspective, which is a great learning experience in itself.

Peer-Group Learning and Assessment is more challenging, no doubt. And some organisations will certainly lack the will and the means to make it work for them. But because it is challenging, and because it really stretches programmers (and coaches), it is more rewarding. Peer groups feel like they're really achieving something. And they really are.

How we run Peer-Group Learning & Assessment

(Peer-Group & Internal Coach Kerry Jones)

Over the past nine years I've constantly been on a steep learning curve as a Software Engineer and as a Technical Architect and, as a result, I've become interested in how I learn to become an expert in specific skills and more generally how people learn. I'm also responsible for helping to find effective ways for our team of 40 software engineers to learn how to become expert software engineers. We've tried many different approaches to teaching software engineering skills, including sending all 40 engineers on the same training course and employing experts on a contract basis to pair with them on projects. The problem I have with week-long training courses and training performed in short bursts is that, as soon as you get back to work, you start forgetting what it is that you've learned. If you're not using that skill every day it's all forgotten far too quickly. I believe that deliberate and repetitive practice are much more effective ways to learn. If you're not forming habits by practicing a specific skill you will never get to the point where you start to gain a better insight. And that is essential to becoming an expert.

There is research to validate these insights into learning. Mary Poppendiek has recently presented on the topic of Deliberate Practice in Software Engineering which references publications and summaries of the research field by Dr. K. Anders Ericsson.

Becoming an expert in any specific skill relies on:

1. The motivation of an individual to want to improve
2. Having a mentor available who can design exercises based on the apprentices current skill level that will instill good habits and stretch the learner
3. Deliberate practice and repetition of training exercises, and
4. Immediate feedback so the learner can see the improvement in their performance.

Here I explain the Software Craftsmanship Programme Jason has introduced to us based around Peer-Group Learning and Assessment. We also reflect on what we've learned in the process of implementing it and show how we are creating an environment where everyone can learn to become better software engineers.

Software Craftsmanship Programme

We started our Software Craftsmanship Programme with an initial trial group of eight Software Engineers. Most of us were senior, so there was a fair amount of development experience within the group. But, as we quickly found out, we had a lot to learn, and still have. The programme consists of a number of groups of about eight engineers, including myself, progressing through various stages of practice, starting with the more fundamental skills and then moving on to the more challenging disciplines. Once one group has

successfully passed their assessment - which means they have been found by their peers to consistently apply the good habits their group has agreed on - they are then qualified to become coaches at that level of skill and work with a new group of apprentices to help them build a similar set of good habits. At the same time as coaching new peer groups, the people from the first group move on to learning a different skillset or a more advanced level of practicing an existing discipline.

For each group, the process is the same:

1. Choose the skill that you'd like to improve
2. Attend an orientation day with an expert in the chosen skill and a number of coaches to develop the group's Skill Worksheet
3. Pair with others in your peer group, the coaches and the expert until your Skill Worksheet is completed
4. Complete an assessment day where all the peers of the group mark each other and decide on who should pass or fail
5. Rinse and repeat

That's it in a nutshell, but there is a lot more detail in each of these phases that needs to be explained and understood, so let's start.

Choosing the skill

For groups new to this process, choosing the skill they'd like to improve can be challenging. However, there is one skill that we believe to be a fundamental basis for the rest of the software engineering practices. That is Test Driven Development (TDD). As a result, all three of our groups have started with "Apprentice-Level" TDD. Our first group after passing the Apprentice Level TDD have moved on to Apprentice Level Refactoring and are also coaching the two new groups in TDD. Each group has eight software engineers with varying skills and experience.

It's important to highlight here that noone in our 40-person software engineering team is exempt from this programme. It doesn't matter how expert you think you are at a specific skill one of the most important features of this way of working is Peer Learning. Everyone in the group is learning from each other. It's important to have an expert who isn't part of the group pair with all the members of the group at least once, however it's up to the group to organise pairing sessions with the peers in their group and to help each other complete their individual Skill Worksheet. If you are lucky enough to have people who really are experts in the skill participating in the group then it gives them a great opportunity to help teach everyone else in the group. I also firmly believe that an expert will also learn a lot from the peers in the group.

Orientation Day

The orientation day is facilitated by an expert in the skill who spends the first hour or so demonstrating the skill, explaining what they do and the habits they have developed and what they believe to be important.

The goal of the orientation day is for the members of the peer group, excluding coaches and the expert, to choose and agree on a set of 10-12 good habits that they will have adopted in relation to the specific skill. These habits will form the basis of the Skill Worksheet for this group and they are what each of the peers will assess each other on during the assessment day to determine if they have passed or failed.

Our peer group choosing and agreeing on the 11 habits that we wanted to adopt was an important aspect of our peer learning. It created a sense of ownership amongst the group. We had in fact created our own "brand" of

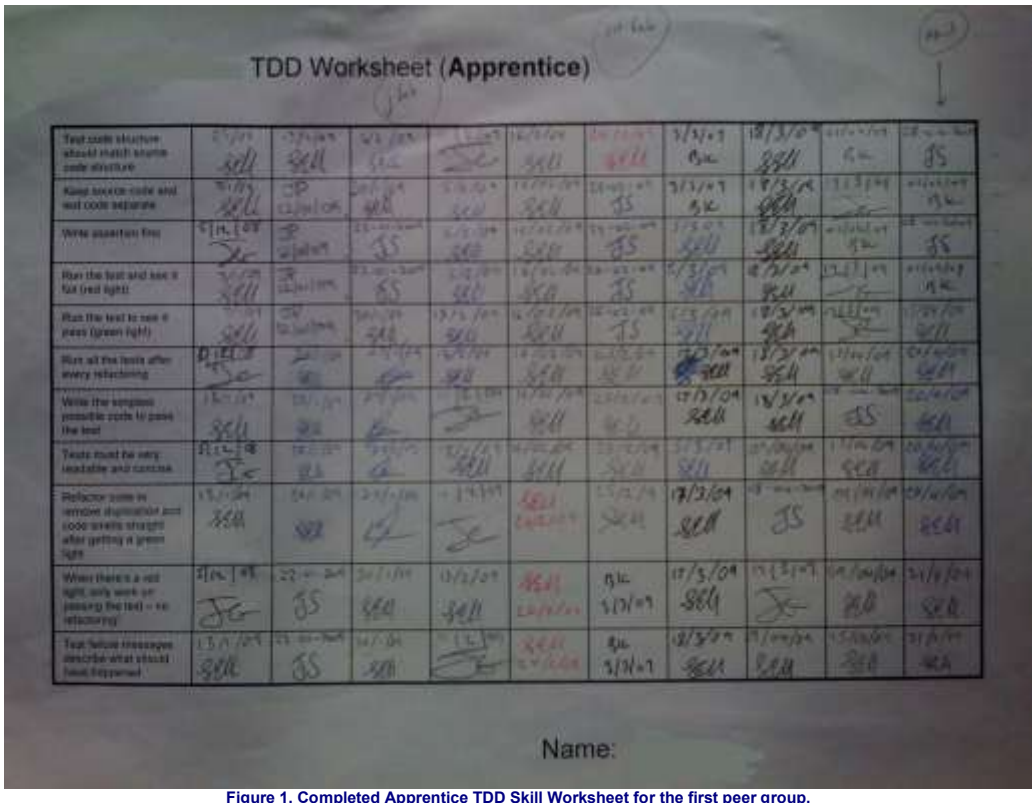

Figure 1. Completed Apprentice TDD Skill Worksheet for the first peer group.

TDD and for the first time all eight of us agreed on what TDD is, or at least what it is to us. We had an established set of good habits that we wanted to learn and we had all bought into them. We were influenced by the expert in the room but the expert needs to facilitate

the discussion and to explain the reasoning behind why they feel a specific habit is important. The expert can also explain why other habits are bad and provide insight into why they are not worth adopting. Ultimately the decision on which habits to choose is entirely up to the peer group. As this was apprentice level TDD we decided to leave more complicated areas of TDD like mock objects out of this round and to introduce them during the next journeyman level of TDD. We also decided to leave anything that was a bit controversial out of apprentice level TDD and chose to discuss and possibly adopt them again once we start the journeyman level TDD.

The Skill Worksheet

Each individual in the peer group receives a Skill Worksheet that will look something like Figure 1.

Our worksheets were designed to give every member of the group at least 55 hours of pairing with their peers and/or a coach while strictly following the habits on the worksheet. The worksheets are completed when all of the boxes on the sheet have been filled in by someone from your peer group. To fill in a box or set of boxes a peer must have worked with you for at least an hour and confirmed that you followed the habits listed on the sheet. Pairing sessions were two hours long and would involve ping-pong pairing or passing control of the keyboard after the first hour. During the pairing session the person being signed off chooses a maximum of four habits to concentrate on for this pairing session. At the end of the session, if your pair agrees that you followed those habits for the entire hour, they put their initials in the relevant boxes. If you're not the person at the keyboard during a pairing session then your job is to point out where your pairing partner is not following the agreed good habits.

The pairing sessions force every software engineer to perform the skills over and over again, reinforcing the learning until the good habits we had chosen became second nature. By the time we had reached the end of our worksheet we were doing TDD with a good bunch of habits automatically. We didn't have to think about the process any more. Some of us actually started to feel uncomfortable if we saw someone do a refactoring and leave it too long before they ran the tests again.

Completing the worksheets in this way meant that we were setting aside time to deliberately practice TDD. Sometimes we were working on real code bases and sometimes we were working on a set of TDD practice examples but throughout it all we were deliberately practising TDD and repeating training exercises.

Assessment Day

The assessment day was hard, even after setting aside time over the previous four months to practice. One of the reasons it was hard was that one of the major goals of the day was to prove that we could TDD all day and maintain the good habits that we had been practising without making too many mistakes. The day was split into coding sessions of an hour. Each session was recorded using a screen capture tool, like Camtasia Studio. After each session, we would swap desks and watch back the recording of one of our peer's sessions, keeping an eye out for any instances where they failed to apply one of the good habits we had agreed on in the orientation. If we spotted them failing to apply a habit (e.g., refactoring when the test was failing), then we would put a big "X" on an empty worksheet against the habit we caught them breaking. If you got three or more X's during any 1-hour coding session, you failed the assessment. If one of the sessions was done in a pair, then both members of that pair either passed or failed together, so it didn't matter who was typing and who was navigating when the mistake was made.

This is what we did on the assessment day:

1. Individually record yourself solving a pre-defined problem using TDD in 2 x 25min sessions with a 5 minute break in between.
2. Swap desks with someone from your peer group and mark your peer's session using a blank skill worksheet
3. 30min break
4. Pair with someone from your peer group to solve a pre-defined problem using TDD in 2x25min sessions
5. Swap desks with a different pair from your peer group and mark their session
6. 1hr lunch break
7. Individually record yourself solving a pre-defined problem using TDD in 2x25min sessions
8. The group expert marks this final recording and because he has eight hours of sessions to get through these are not marked on the day

What we found and what we changed as a result

One of the most important ideas behind peer learning and the worksheet is that you need to set aside lots of time to practice. A few of the engineers in the first group found it hard to set aside the time required to complete their worksheet. In both cases this was because of major project deadlines where one person was working on projects that didn't involve writing any code and another was working in a language that doesn't have good enough tool support for unit testing or refactoring. They found it hard to take time outside of their projects to do the practice they required. This highlighted the fact that you absolutely need support from senior stakeholders in implementing a programme like this. If the support doesn't come from high enough in the company it is difficult to find the time to complete the worksheet.. To allow these two engineers to complete the worksheet we pushed the date of the assessment back one month and I worked with their project manager and line manager to help them find the time they needed to complete the worksheet.

An engineer who is aiming to complete a worksheet really needs to set aside a minimum of two hours a week to pair with someone in their peer group. Anyone who didn't do this found it hard the closer we got to the assessment day because they had too many pairing sessions to complete and were struggling to fit them in around their other work. If you left most of your pairing sessions towards the end, you were effectively cramming for an exam. A result of the cramming is that you don't pick up the habits as well as other engineers and have to concentrate more on the assessment day and don't do as well on the assessments. To get around this issue and make sure peers are pairing every week, we've introduced a weekly stand-up so that peers can check each others progress on their worksheet and organise some time that week to pair.

Two engineers in my group found it easy to complete the worksheet within a couple of months because they were working on the same project together and were already using TDD on the project. As they were developing new features at work for one of our products they were also able to sign each others worksheet, and it was possible for them to spend the two hours required for a pairing session almost every day. This is the most effective way to complete a worksheet with minimal disruption to your daily work.

Towards the end of our worksheet, a mock assessment was introduced where we set aside a day to go through an assessment as if it was a real assessment day. When we did this most of us failed. It was a valuable experience because we had still not developed the habits we were hoping to by this point. This left us with some time to practice more and to work on the areas which hadn't become habits. It would have been better if these were corrected early on, so in future groups we will be performing a mock assessment when they are about half-way through completing the worksheet.

Once my group had successfully completed apprentice-level TDD, we became coaches for the next two groups starting their apprentice-level TDD. A promising sign is that, the responsibility of leading and coaching new groups through the same process, the peers from my group are showing greater enthusiasm for the programme. They are now seen as coaches of a specific skill within the department and feel empowered because they help define and drive the process and are starting to teach our other software engineers.

Gathering Empirical Evidence

One of the areas that Jason and I are interested in is how we can use software metrics to gather empirical data and evidence of the improvement in code quality as a result of the Peer-Group Learning & Assessment (PGLA). What we will be looking for in the future is a way to demonstrate more objectively that software is better tested, simpler and easier to change. We don't yet have enough data to provide conclusive proof that the software craftsmanship programme improves the quality of our code. However, the initial results are encouraging, as indicated in Table 1.

In Table 1 - LOC is the Lines of Code, Line/Branch Coverage refers to the unit test code coverage and Complexity is measured using (Cyclometric Complexity / the number of methods).

Each of the rows in Table 1 is a Java code base developed by software teams that were made up of a few people who were participating in the TDD apprentice-level training, and some who weren't. The software in the top section was developed before the TDD apprentice-level coaching started. The software in the bottom section existed before we started our first peer group but has been developed further since the introduction of PGLA. The software in both sections has been developed by teams involving engineers in the first TDD peer group before and after their TDD apprentice-level assessments.

Before TDD Apprentice Peer Learning

LOC	% Code Coverage	% Long Methods	Average method Complexity	% Duplicate Code
2013	34.1%	2.53%	2.1	2.9%
6902	54.0%	0.30%	1.9	14.4%
2144	89.7%	1.07%	2.1	6.3%
5171	64.1%	1.26%	2.4	7.7%

During and After TDD Apprentice Peer Learning

LOC	% Code Coverage	% Long Methods	Average method Complexity	% Duplicate Code
1463	96.7%	0.27%	1.5	0.80%
2550	96.3%	1.18%	1.6	2.1%

Table 1: Software Metrics of projects developed by the team before, during and after the TDD apprentice-level assessments

Keith Braithwaite's research on measuring the effect of TDD shows what we expect to be different in code that is developed using TDD. Rather simplistically, we expect the code to be better tested and for classes and methods to be less complex. To give us a simple indication of this at first we've chosen to use code coverage, percentage of long methods, average method complexity and percentage of duplicate code. What's interesting about these results is that the code developed before the TDD apprentice-level peer group started was actually developed using what the engineers believed at the time to be TDD. After completing the TDD apprentice-level assessments, the code coverage seems to be increasing and the overall complexity and length of methods is decreasing along with the amount of duplicate code, which is a promising trend.

Conclusion

The first group of TDD apprentices, including myself, was run as a trial within our department so we could assess the benefits of the peer learning approach. As the Technical Design Authority (TDA) for our code bases, being a part of a peer learning group has been a real eye-opener. Although we only have a small amount of empirical evidence that the software quality has improved, it's obvious when pairing with those of us who have passed the TDD assessment that the quality of the new code we're producing has improved. The code is better tested, the engineers are regularly improving the design of existing code, and, because the new code is developed using TDD, the new code is more modifiable and maintainable than our previous software projects.

So far we've not publicised our software craftsmanship programme to departments outside of our own. However, the buzz created by our software engineers and managers is starting to get noticed in other departments. Managers who are running software teams close to us recognise the value of helping their software engineers improve and work towards becoming experts, and are starting to ask questions about what we're doing. One of our goals in future is to find out if there's enough appetite from other departments, and to work with them to help spread PGLA to help more software engineers become experts.

The feedback from the first group of engineers and coaches, as well as the improved quality of the code we are now producing, has allowed us to get the funding approved to scale up the peer-group learning for TDD apprentice-level to two more groups of eight engineers, and to start the first peer group on the refactoring apprentice-level. Now that 16 new engineers are starting their TDD apprentice-level, and the results of the first group are becoming visible, being a part of the peer-group learning programme has become something that most - if not all - of our software engineers really want. As we've not been able to scale the process any more quickly, there are a number of engineers who were upset at not being included in one of the new peer groups. They've heard stories from the first peer group about just how much they have learned and improved over the past six months. There's a real sense of achievement in passing the peer-group assessments, and that's been noticed by everyone in our team. Passing the TDD apprentice-level assessment is a goal that's also been written into some of our software engineers' yearly appraisals, which shows our commitment to continuing with this programme in the immediate future.

I believe in learning by doing. The Software Craftsmanship movement that has grown in recent years is great because more than ever software engineers are showing each other how they write code. Showing how we develop software, providing constructive feedback and learning from our mistakes is essential for the development of software engineering as a discipline. Peer-Group Learning and Assessment promotes shared learning and creates an environment where people take responsibility for learning and practising. It provides a structure where software engineers can learn from a mentor, allocate time for deliberate practice and get immediate feedback during pairing sessions and assessments on their performance.

Jason Gorman is a director of Codemanship Ltd and has coached hundreds of professionals in the key disciplines of software craftsmanship. His web site parlezuml.com has been visited by more than 1 million people since 2003. He chaired the first international conference on Software Craftsmanship in 2009, and contributes to other conferences including XPDay, Software Practice Advancement and CITCON. He is also executive producer of "Boffoonery!", a comedy benefit in aid of Bletchley Park.

Kerry Jones is a Technical Architect who specialises in broadcast and digital media. He worked in the TV Platforms Group of BBC Future Media & Technology, the team responsible for delivering the BBC Red Button service and interactive applications including BBC iPlayer on internet connected TVs. He was also on the review panel for papers submitted to Software Craftsmanship 2009