How to Run Your Own 3-Sprint Agile Workshop

-- Robert L. Read

Purpose

You can't learn Agile software development from a book, anymore than you can learn to perform a one-handed jump shot without repeatedly tossing a basketball in the hoop. You can read a book about the basic idea, you can read a book to get started, and you can read a book about refining your technique, but in the end you have to practice.

A basketball player performs drills in order to play better in the game. In technology development, we can perform workshops to get better at our main business---delivering software.

This is true whether you are a developer, designer, project manager, or, perhaps most importantly, a customer or an executive. It is particularly important for executives to appreciate the advantage of Agile methodologies, and the best way to come to that appreciation is to see it in action. The lowest-risk way to do this is to perform an exercise that can be done in a single afternoon with any group of willing participants.

A successful workshop will do two things:

1. It will surprise the participants with how much can be accomplished within a short time.
2. It will convince people that evaluating early iterations is invaluable for making adjustments for the next iterations to achieve the best results overall.

A workshop may be run with an existing team, or with people who don't know each other meeting for the first time for the purpose of playing with Agile development.

Structure

The shortest reasonable workshop is comprised of 3 simulated "sprints". Each sprint has the same structure, which represents the actual structure of a software sprint. In order to fit the workshop into a three to five hour period, each sprint must be very short---30 to 60 minutes. Nevertheless, your participants may be surprised what can be accomplished in this short time, if you, as the Master of Ceremonies, can effectively enforce a disciplined structure.

Although one can always adjust the Agile process to suit your needs, I recommend being strictly dictatorial in your workshop around timekeeping and the structure of the sprints. The overall structure is:

- An introduction and explanation of the process,

- Sprint One,
- Sprint Two,
- Sprint Three, and
- A Retrospective and Reflection.

This means that there must be a rigid agenda, with Sprints One, Two, and Three beginning precisely at the planned time.

Each Sprint likewise will consist of:

1. A period of writing stories on note cards,
2. A development period that focuses on interaction between the customer and developers,
3. A demo,
4. A sprint retrospective which focuses on the process, and
5. A break before the next spring begins.

The time allotted to each of these activities will depend on the overall timeframe for your workshop. In general the demo and the retrospective for each sprint can be done in 5 minutes each and the story writing period can be 5 to 10 minutes.

Teams

You can have one, two, or three teams---but if you are running a workshop for the first time, I recommend starting small. No matter how many teams you have, you must enforce a precise agenda for each team. Every workshop participant should participate in all demos, and you should organize your agenda around consecutive demos.

I prefer not to have the teams compete, but to give each team a different goal. It is even better if these goals are complementary.

The Teams consist of two clearly distinguished roles. The *Customer* gets value from using what is built, but does not care how it is built. It is critical that to remember that the Customer is end-user, not the person funding for the project. For example, a CIO of an agency is the executive in charge, but is rarely the Customer, except for dashboard and reporting projects. It is imperative, especially in government work, to always seek to interact with the actual Customer as early in the process as possible. In the case of a learning workshop performed as an exercise, the Customer may be an executive who simply wants to see how Agile can work for them, or could be a true end-user of an existing team. In many cases, the Customer is not a technologist, and in all cases they should have no opinion about how to accomplish something.

The second role is named *Developer*---by which we mean Designer, Coder, Writer, or more generally, Maker. The ideal team size is 1-2 Customers and 2-4 Developers. If you have more than five Developers, there will be natural pressure for it to devolve into two Teams, particularly in such a short workshop.

Roles

Each participant performs one and only one of the following three roles:

1. Customer
2. Developer (Designer, Coder, Writer, Maker)
3. Master of Ceremonies and Coach

The Master of Ceremonies/Coach

The Master of Ceremonies/Coach enables the rest of the participants to learn by performing key responsibilities.

Prepare the Participants

The Master of Ceremonies/Coach must coach the Customers ahead of time so that they are prepared to drive their teams. The Customer must have a clear goal in mind, and sufficient willpower to direct the team appropriately. If the Customer does not have one ahead of time, the Master of Ceremonies/Coach must work with them to develop a goal.

Provide an Effective Physical Workspace

They must also enable fun by making sure that the physical space is appropriate, there are appropriate refreshments available, and the office supplies are readily at hand. Notecards and pens are absolutely required. A whiteboard or flipchart for each team is valuable for “scribing” the sprint retrospectives and the final workshop retrospective. Finally, they must make sure that the computer and audio/visual facilities are properly prepared. Not all workshops will require computers or audio/video space, but larger ones may.

Preparation of the non-physcial workspace may include establishing ahead of time a shared file storage, ideally with version control and easy web viewing, open to all participants, such as those offered by a shared code repository on github. This makes it easy for people to collaborate, leaves a history, and produces a durable record of the workshop product to be used later.

Enforce a rigid schedule

Enthusiastic teams will want a little more time to improve their demo. The Master of Ceremonies/Coach must not succumb to pressure to deviate from the pre-planned schedule. If a demo is less than impressive because a story is not done, this is a learning experience for the team.

Coaching During the Workshop

Coaches should make make sure that stories are written down on notecards during the story writing phase. However, even more important is that the progress of the development team during the development phase be immediately and constantly shown to the Customer, who may make adjustments in priorities at any time.

Coaches may also provide guidance on Agile techniques during the workshop. However, this is probably the least important of his or her responsibilities. If they can enforce the schedule, he or she need not be a terribly experienced Agile practitioner. The Teams will learn by doing, and if they have properly prepared the Customers and created an enabling space, the Teams will learn from their failures and/or enjoy their successes based on their own merits.

Enforce Responding to the Customer

The paramount responsibility of the Master of Ceremonies/Coach is to insist that ALL work be shown AS SOON AS POSSIBLE to the Customer and that Customer changes to direction and priorities be IMMEDIATELY respected by the developers.

The Customer

The Customer is the most important role, and demands preparation. The Customer is considered part of the team. The Customer must bring to the workshop a clear idea of something that they would like to have produced within the workshop and an ability to flexibly set priorities and give feedback based on progress.

The Customer must be prepared to participate in a story-writing process that is motivated by a clear goal. This goal need not be precise or detailed. However, the Customer must have in mind what they want and the ability to express delight when progress toward this is made. He or she must be prepared to help the group articulate very small steps towards the goal and compromisingly choose priorities for steps that team can accomplish. The Customer must be prepared to converse with the developers and make a quick prioritization based on a quick estimate.

A Tester or Quality Assurance person is considered a Customer for the purpose of such a workshop. An Executive who want to experience Agile development should probably play the role of Customer, because the most valuable thing that can happen in the workshop is for them to come to appreciate the power of being able to make iterative changes to development based on what has been learned from rapid, early development.

The Developer

The Developer may be a computer programmer, a writer, a designer, or anybody else who is making things. The reason the Developer role must be clearly delineated from the Customer is that the Developer MUST NOT be a bully. That is, Developers always have a sort of magic power---since they have a monopoly on making things, they can easily slip into asserting their own ideas about what should be made. This is the one thing that Developers should not do. They must cede all control over what is made to the Customer.

It is, however, the Developers' job to teach, (in this case very quickly) the Customer how much time it will take to make any given story that they ask for. The Developers must provide quick estimates on the spot. They do not have to be accurate, or confident---they need only be honest.

Example Team Goals

You can run 3-Sprint Agile Workshop around a goal of producing a valuable end product. However, you may prefer to simply treat it as an exercise.

When I did this many years ago in Mumbai with three teams, I used the task of building an online calculator. This is a good exercise, because one can easily imagine a very weak calculator with features being smoothly added to create a powerful calculator.

Looking at great teaching books can provide good ideas for projects. For example, a currency conversion calculator sounds simple, but has some interesting challenges. This was an effective example from Kent Beck's book Test Driven Development by Example introduced shortly after his seminal work Test Driven Development. I have not run a workshop around it, but I believe one could run an effective 5-hour workshop around this exercise as well.

For a group of participants that did not include strong coders, I might plan goals easier to program or which involved no programming at all, such as creating a simple blog with GitHub Pages or WordPress.

If I had a group of program managers, I might have the Developers essentially be writers, and suggest goals such as "Build a website that informs people about the benefits of regular exercises." In all cases I would work with the Customers ahead of time to make sure they had a clear idea of what they wanted to accomplish, but were open to change based on what they got from the Developers in the First and Second Sprint.

The Surprise Pivot

In order to demonstrate the value of Agile development in a fun and challenging way, you should prepare ahead of time a little trick to play on our teams. This is a "pivot" away from what they were originally thinking to a new mandate driven by the Customers. Such changes are major risk to government software development. Sometimes these are actually driven by regulatory changes, sometimes by a changing technical environment, sometimes by discovery of a new opportunity.

The pivot should occur in the 3rd sprint. If possible, it should relate the multiple teams that you have to each other. It is important that you coach your Customers ahead of time about the pivot, so that they can correctly communicate the new requirement to your teams.

Case Study: The 18F July 1st Workshop

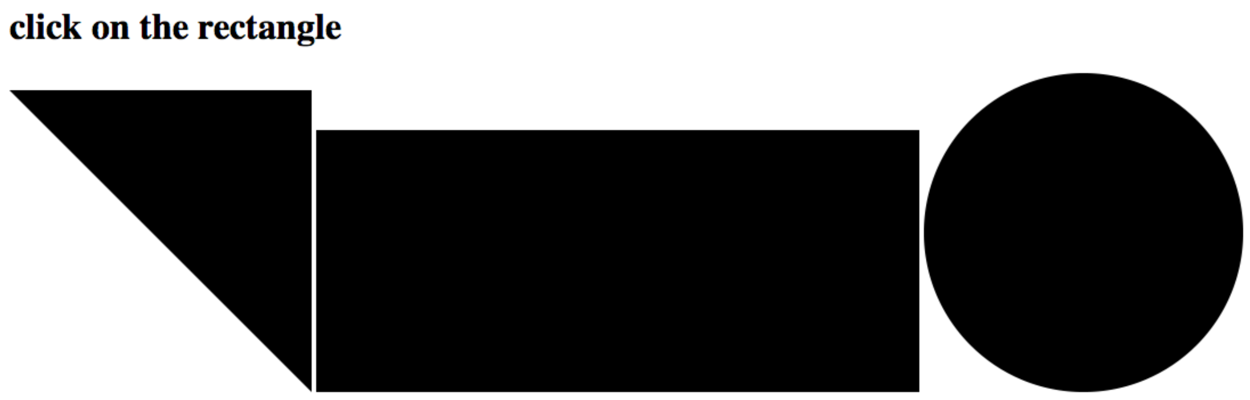
When I recently ran such a workshop at 18F as an internal team-building exercise, we had two teams. (This was a managerial mistake on my part, I should have had three, since one team was too big. If you run your own workshop, expect to have a few things go wrong.) I had previously recruited two volunteers to be Customers. The goal I suggested to them were these:

1. Create an educational game for children involving rectangles.
2. Create a trading game based on the *Game of Thrones* drama series.

These were ambitious goals for a 5-hour workshop, but 18F has strong developers.

The Customers and I intentionally kept the goals of the workshop teams completely secret, so that we would have a strict time-box and a cold start.

The Customer for the educational game took her team in a direction I had not planned: she aimed it at very young children, who did not know what a rectangle was. That's the way Agile works: the Customers usually surprise you. By the Second Sprint they had a playable, if tiny, game developed that would actually hold a young childs attention for a minute or two:



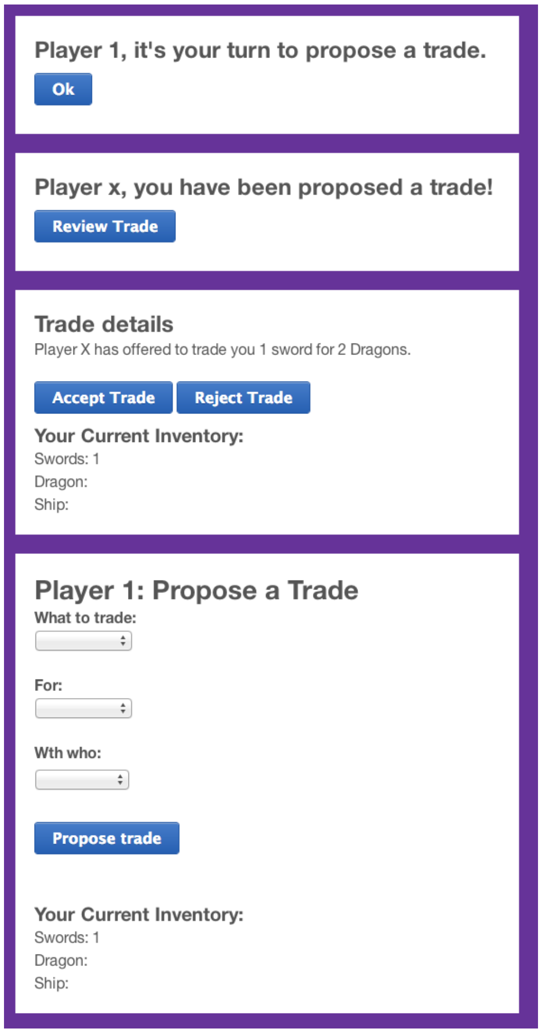
The Surprise Pivot had been planned in advance, and interrupted the progress toward an educational game. This was unfortunate, but it is not uncommon in software development and government work in particular where mandates are made over which you have no control.

The Surprise Pivot Mandate made to both teams: Integrate the *Game of Thrones* with something about Rectangles.

The Rectangle team responded by producing a beginning of a game which did fulfill the mandate.
A problem arose on the second *Game of Thrones* team which I had not anticipated: too many people (eight!) wanted to be on it. It is very hard for that many people to work closely in such a short, intense, sprint. The team naturally split itself into two distinct but cooperating teams. One

began working on a paper-based game which was to guide the software team. The software team got a rudimentary trading game done under the guidance of the Customer.

The *Game of Thrones (Software)* team produced code that implemented the beginning of a trading game that looked like this:


Player 1, it's your turn to propose a trade.

Player x, you have been proposed a trade!

Trade details

Player X has offered to trade you 1 sword for 2 Dragons.

Your Current Inventory:

Swords: 1

Dragon:

Ship:

Player 1: Propose a Trade

What to trade:

For:

Wth who:

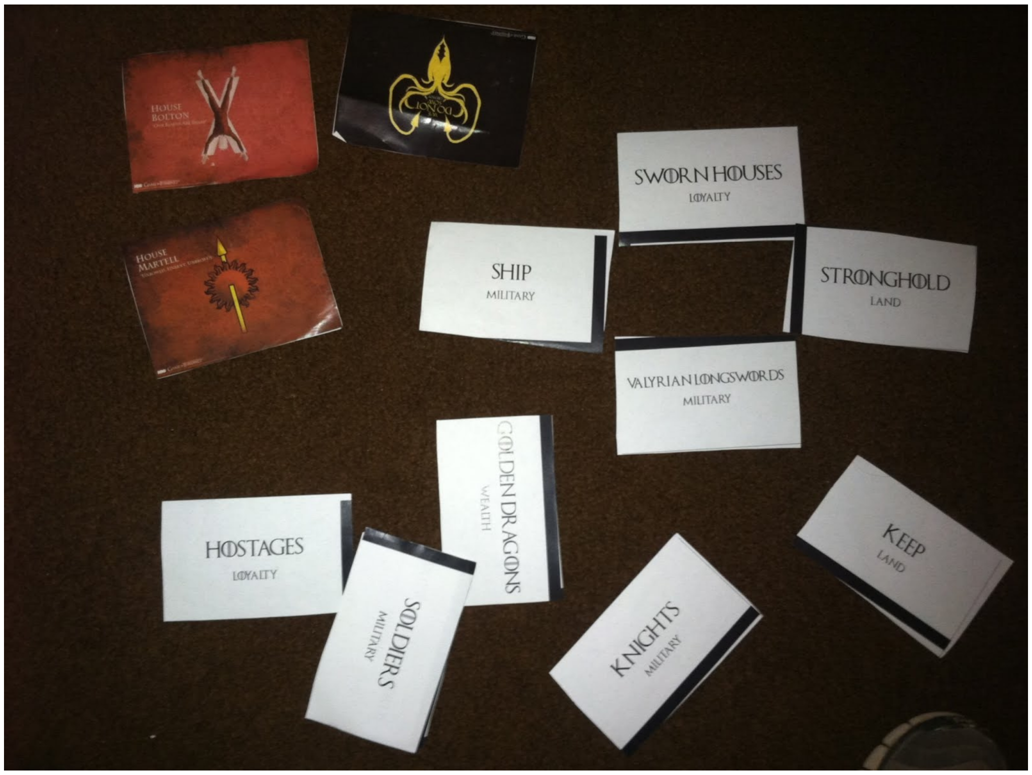
Your Current Inventory:

Swords: 1

Dragon:

Ship:

The *Game of Thrones (Paper)* team produced a paper trading game that consisted of objects and and concepts from the drama. When faced with the Surprise Pivot, they met the mandate by adding black bars to the concept cards and creating a game goal of forming rectangles of the black bars, as exemplified:



Since this was a trading game, the goal was to obtain the cards that fit together to create a rectangle.

Things That Can Go Wrong

To some extent, things going wrong is helpful, if the team can learn from it. Here are some things that can go wrong:

- The Team can fail to break the goal into small steps that can be executed within the limited time of one sprint, so that they have nothing to demo.
- Developers can be bullies and impose their own desires on what gets built.
- Customers can fail to assert priorities.
- The Team can fail to simplify their goal enough to get anything done within the workshop.
- A Team can be so large that it is difficult for them to coordinate work within a sprint. If you know you have that many participants, you should increase the number of Teams to keep the Team size smaller.
- The Team may devote too much energy to thinking about the final product in the Third Sprint and thus fail to do good a good job presenting functionality in the First Demo. A Team that does this will also be stung by the Surprise Pivot, which will likely cause some of their work to be wasted.

You Can Bring On 18F as a Coach

18F has recently created a new line of business called Agile Assisted Acquisitions which offers Agile coaching and other services and cost-recovery prices under the Economy Act.