Done List Creation Exercise

You will need the following items to perform this exercise:

1. Your team
2. Many pads of Post-It™ notes in multiple colors (stickies)
3. Pens (I find that sharp, permanent markers work best)
4. A room that the team can utilize for two to four hours
5. An open mind
6. No interruptions

The *Done List Exercise* is constructed of these components:

1. Brainstorming Session
2. Categorization Session
3. Sorting Session
4. Done List Creation and Publishing

This exercise will help you create your initial done list. Notice I said "initial." A done list is not created once, buried in a drawer, and never seen again. Teams should schedule time during retrospectives to periodically review their done list and determine whether there are opportunities for improvement or modification. Feel free to revise the list to meet the needs of the team and its stakeholders, but be careful. Too many revisions, too frequently, may create doubt about the validity of the done list in the eyes of the stakeholders. Invest the time to create a solid baseline and modify the list based on the experience and findings of the team.

Remember that levels of done come in various forms, the most common being technical and functional. The samples presented here illustrate a level of technical doneness. However, functional doneness is also key and can include items such as,

- Automated acceptance tests passing (story level)
- Product Backlog estimated and prioritized (sprint level, product owner deliverable)
- Team retrospective complete (sprint level)

Introduction

People often ask me who from the team should participate in this exercise. This question always catches me off guard.

We have been conditioned to work in our functional silos for so long that people often look perplexed when I answer this question. I believe that *everyone* on the team, no matter what their expertise or background is, can add value and should be involved.

If the team's project is a backend database system or a three-tier web application, there will be people on the team that have less experience and knowledge of the technology than others. That is just a fact. Keeping team members out of this exercise, though, only deprives the team of the valuable contributions of other team members. It also keeps them from having a valuable *team building* exercise.

I recommend having the entire team do this exercise, regardless of skills, background or role on the team.

1. Brainstorming Session

Alex F. Osborn is widely known as the father of brainstorming. In his books titled *Your Creative Power* [Osborn01] and *Applied Imagination* [Osborn02], Osborn outlines the brainstorming technique, a system that uses the brain to "storm" creative solutions to a problem. This creative approach is designed to generate a free flow of ideas. In the brainstorming session there are no right or wrong answers. There is no criticism. There is just the free flow of ideas. The rules of brainstorming, outlined by Osborn, are simple. They are:

- No criticism of ideas
- Go for large quantities of ideas
- Build on each other's ideas
- Encourage wild and exaggerated ideas

At the beginning of the brainstorming session, set the proper tone. The reason the team is together in the room is to identify *everything* that it needs to do to ship software, the truest measure of project progress I have encountered to date.

It is important to set the direction of the brainstorming session in the beginning. Start by writing the question the team will answer with its done list on a whiteboard. What question is that? It depends. The most common question is this:

"What do we need to do, as a team, to ship software to our customers/stakeholders?"

Your question may vary, depending on your own team's unique circumstances; however, the purpose is the same. Make sure the question is on a whiteboard, or any other viewable place in the room, before you continue.

Hand everyone a pen and a sticky pad. I have found that individuals like having different colors; this will ultimately depend on your team members' preferences.

You are now ready to begin brainstorming. Kick it off by repeating the question you are answering (see above for an example). Have each team member write an answer on a sticky, call out the answer, put that sticky in a pile in the middle of the table, and repeat. Team members can write only one answer per sticky and must call out after each answer is written. As team members start to share their answers, inevitably some will say, "I've already written that." Do not worry about creating duplicate items at the moment. This will be addressed later.

Why the call out? I have found that the call out puts the *storm* in *brainstorm.* I have personally identified multiple additional items when fellow team members have called out a done list item. I have also witnessed team members building on each other's ideas when coaching teams through this exercise. I have run this exercise without the team members calling out what they wrote on the stickies, but it resulted in a stale, unproductive meeting. Calling out will feel weird and uncomfortable for most; however, the feeling will subside very quickly once the team becomes engaged and the meeting begins to flow.

Run the brainstorming part of the session until there are "enough" sticky notes to get started with the next phase. It is generally easy to identify when people are done because you will see the flow of sticky notes begin to dwindle. People will begin looking around the room and at each other, searching for something new.

2. Categorization Session

There should now be a large stack of sticky notes on the table, waiting to be categorized.

Start by querying the team members on how they each think the sticky notes should be categorized.

The most common responses I see look like this:

- Development
- Test
- Project Management
- Other

The responses above are decomposed by functional work area.

This response makes sense. After all, we have been conditioned to work in functional silos, not in cross-functional multi-discipline teams. Unfortunately, categorizing this way reinforces our functional roles within a team and does not promote cross-functional interaction. It also tends to create a mini-waterfall for each iteration, missing the point of the done list principles and the "potentially shippable" aspect of Scrum. Additionally, customers and stakeholders generally do not care what it means to be done with development or test; they care what it means to be done with a release to production or a release to manufacturing.

If you get those responses, challenge the team to focus on customer value. Delivering stories at the end of an iteration and releasing to an integration environment where customers can use and interact with the software provides value. Even more value is provided if a system is released to a production environment and officially done.

After introducing this line of thinking, the responses I typically see change dramatically:

- Done with a story
- Done with an iteration
- Release to integration
- Release to production

Once the categories are identified, clear, and understood by the team, create areas on the walls in the room to post sticky notes under each category.

This is a team exercise. Each person will grab some sticky notes (preferably others' notes) from the table and begin placing each note under the appropriate section. Team members should not spend a great deal of time wrestling with whether categories are right or wrong—at this point, they should just get the notes on the wall. Sorting the notes should take between ten and twenty minutes, depending on the quantity of sticky notes generated.

Once all sticky notes are on the wall, have everyone take a few steps back and look at the wall.

It's going to look like a lot of stuff.

3. Sorting and Consolidation Session

Once everything is on the wall and categorized, it's time to begin consolidation, using an approach similar to that advocated by Mike Cohn in his *User Stories Applied* book.

This is a team exercise. Each person will review the sticky notes on the wall and look for duplicate items. Types of duplicates include exact matches, similar matches and "we don't know what the heck these are."

For exact matches, simply take the two matching sticky notes and stick them to each other so that the top one fully covers the one underneath (see Figure 2). Do this only when it is clear that the items are exact matches and that the decision to marry them will require little discussion later in the session.

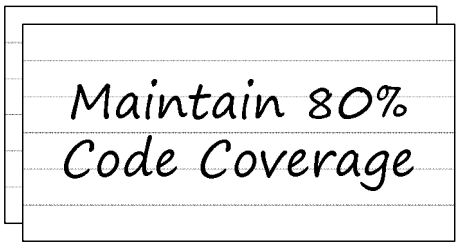

Figure 1. Exact Matches

For similar matches, have the top sticky note offset the one underneath. This helps everyone to easily identify items that are similar, but not exact, matches. These items will require further team discussion to determine a clear meaning of the done list candidate item.

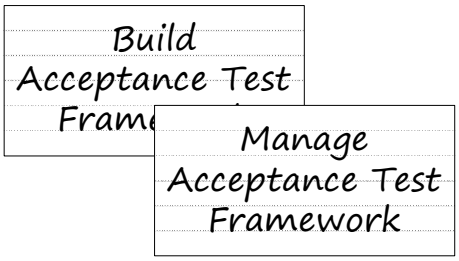

Figure 2. Similar Matches

For matches that make no sense at all, create a new category off to the side and name it "other" or something similar. The team will revisit these items last.

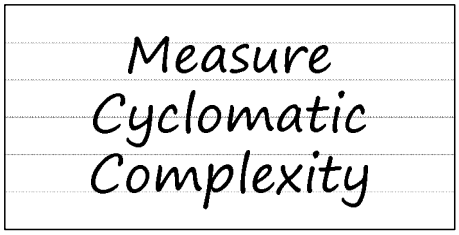

Figure 3. Makes No Sense

Again, have the team step back and look at the wall. They will probably still be overwhelmed, but they'll be starting to feel that things are a bit more manageable.

Once all notes are consolidated, the team is ready to discuss what each sticky means.

The process I use for this is relatively simple. As facilitator, start with a set of stickies (I like to start with the exact matches, as they are the easiest), take them off the wall, read them to the team and ask, "What does this mean?" Once the team answers, move on. It is important to time box this activity. If the team cannot come to consensus, or even agreement, shelve the item for later or take a mental break by going for a walk.

If there are team members that do not answer, probe them directly and ask for confirmation that he or she understands what a specific sticky set means before moving on.

After the exact matches are finished, move on to the similar match sets. These are managed slightly differently. Take the set off the wall, read them to the team and ask, "What does this mean?" When the team answers, pick the card from the set that most closely matches the answer they gave or create a new sticky to reflect what was discussed.

After a set of similar items has been discussed, the team should be left with only one card that represents what will be committed to and communicated by the team. Throw out duplicates or consolidate them like their "exact match" counterparts. Throughout the process, new stickies may be created and old ones discarded. This is not only expected, it's key.

Lastly, move on to the stickies that are anomalies or make no sense. Use the same process of pulling a sticky off the wall, reading it and asking what it means. Follow this by asking, "Does the team need this item on its done list to be done?" If the team responds with a resounding "yes," move the sticky to its proper category. If the team is so-so about it, or just responds with a plain no, facilitate a discussion that results in a resounding yes or the item being tossed.

Once the team has reviewed all sticky notes, understands what each one means and agrees that all stickies are in the right category, this portion of the exercise is complete.

Question: What percentage of the list your team created is concerned with about writing code? In my experience, writing code[1] *encompasses about 5 percent of the total items listed to be done.*

4. Done List Creation

Teams are generally tired at this point. The entire exercise is a draining experience but very rewarding. Creating and publishing the done list is the easiest part.

Take a digital picture of what is on the wall and put it in electronic format. Publish the list on the internal team website *and* create a poster that can be hung for all to see. It serves as a reminder of what it means for the team to be done and acts as a powerful communication tool. When stakeholders ask, "Are you done yet," the team can point to the done list and answer, "We are done with our stories and working on our release" — and the person asking will understand where the team is at.

[1] By writing code, I mean the actual code itself. I do not consider unit tests, acceptance tests or any other code written to support the actual mainline production code that delivers customer value.