CONTINUOUS COMMUNICATION

PART 2

Kaspar van Dam, Consultant,
Improve Quality Services, follows up his first article on the importance of communication in agile with practical guidelines.

In this article I intend to set out some ideas and methods, I think helpful to improving communication in agile teams. I consider them to be mere guidelines instead of hard rules and I will certainly not claim this to be a complete list of everything needed. But it's my experience and the experience of many colleagues that these are things that could help to make software development a lot more efficient, and in the process also a lot more fun.

KNOW YOUR DOMAIN

Times when IT only needed to know how to program or test are gone. If we truly want to understand what drives the customer to require certain functionality we need to understand the world in which the customer lives. Thus, we should speak the same language the business speaks, creating a ubiquitous language – one of the most important pillars of domain driven design. Only when one understands both the business needs and the technological challenges can one give an optimal advise on how to effectively create the most value for the business.

SHORT LINE, LESS NOISE

We probably all know the game from elementary school where one kid whispers a story to his neighbour who in turn passes it on to the next kid in line, etc. The last kid in line tells the story out loud and everyone laughs because it's a completely different story than the one the game started with. Always.

The same applies for software development. The longer the line, the more noise you'll get. Whenever possible try to get information directly from the source and use the opportunity to ask any questions you might still have directly to this source. Also make sure you've understood what they said. Summarise what has been said in your own words and ask for confirmation that this really is what the other party wants. Prevent situations where everyone interprets things their own way resulting in miscommunication and developed software that turns out to be

something completely different than what the business thought they were asking for.

BE IN THE SAME ROOM

Continuing on the short line principle: face-to-face communication always works best. Preferably including a white board or piece of paper and some markers. There's a hype around working location independently. However, quite often this is more something to cut costs than something born out of necessity. No matter what, it's simply not working! Real teamwork is only possible when you're together as a team. At least, most of the time.

PAIRING

Besides sitting in the same room it could also be a good idea to sit behind the same desk every once in a while. Most programmers are familiar with the term pair programming. Coding together means two pairs of eyes and two brains. Thus less chance of making mistakes and less chance of continuously overlooking that one, not so obvious, error.

Within an agile team the principle of pair programming can be extended to not only apply for programming and programmers, but also for testers and other team members. For instance, as a tester, join forces with the programmer when he's writing his unit tests. And in return, let the programmer sit next to you to think along when writing automated tests. It's a case where one plus one equals three.

DON'T ASSUME

It's a well known saying in the business: don't assume unless you want to make an ASS out of U and ME. It is very tempting to assume things when someone's not present at a certain important meeting. Especially when your under stress because of an upcoming deadline. But even when you're almost certain what that certain stakeholder would say, never forget to get this confirmed. Also get it confirmed before you start working on it, and when it's not possible to get it confirmed because the stakeholder in question has no time? Then there's only one assumption you can make: apparently it's not that important, so no need for the team to put any effort into it, yet. Don't forget to get this assumption confirmed or refuted as soon as possible.

DARE TO SAY 'NO'

Within software development we have grown to treat important stakeholders with kid gloves. However, keep in mind that we as developers and testers are the specialists. The business really appreciates it when we dare to say 'no' whenever we know something will just not work technically or when we know there are other ways to create the same or even more business value with less effort. At least, they should appreciate this. There's no shame in pointing out that something simply isn't possible, even when this is the result of a lack of time and/or the lack of certain knowledge. Try to create an open atmosphere where there's room to be honest and also room to criticise. However, 'no' should never be the final word. There should be argumentation why something can't be done or shouldn't be done. Whenever possible offer some alternatives as well.

#NOESTIMATE

During a recent Cuke Up! Conference agile specialist, Dan North asked the audience two questions about estimation: "How many of you ever made an estimate on how long a project would take?" Everyone in the room raised a finger. However it was quite shocking to see all fingers disappear after the next question: "Did any of you ever get one of these estimations right?"

It's a simple fact that it's mere impossible to give accurate estimations for software development. Even though we know this, we still often tend to give very precise estimations on how long something will take. Sometimes even as detailed as man hours or FTEs. Everything to get it fitted in that one planning spreadsheet. But what's the value of an estimation when we know for a fact that's it's near 100% certain that this estimation is wrong! We already know we will run into certain 'Unpredictable Bad things' as Dan North calls it. Bad, because they will cost us time and/or money. We can't put an estimation on how much time or money they will take simply because we don't know in advance what these bad things will be: they are after all unpredictable!

The solution seems simple: #NoEstimate. However, it won't work to just start work and see where we'll get and when this will be. So #NoEstimate does not mean you have to abandon estimation and planning all together. It's about making it more valuable. For example, by working with ranges instead of man hours. A range I personally find ▶

> Prevent situations where everyone interprets things their own way resulting in miscommunication and developed software that turns out to be something completely different then what the business thought they were asking for

KASPAR VAN DAM
CONSULTANT
IMPROVE QUALITY SERVICES

With over 10 years of experience in IT, Kaspar advises colleagues and clients on matters concerning testing and/or collaboration and communication within (agile) teams. He has published a number of articles on test automation, agile ways of work and continuous communication and is a speaker on these matters at events.

We tend to forget that communication, effective communication, is really quite hard! You have to work for it

Editor's note: Part one of Kaspar's article was published in the November 2016 issue of TEST Magazine.

very effective is the range between 'it might be possible to have it ready, but only if everything goes well' and 'we would be quite ashamed if we haven't finished it by then!' In other words: a good case scenario and a bad case scenario. If management still requires an estimate in man hours then ask them if they really prefer an estimation in hours that'll almost certain be wrong or an estimation in the form of a range in time that'll almost certain be right.

BRING YOUR OWN TEAM

We all know the 'bring your own device' (BYOD) principle. The idea behind 'bring your own team' is basically the same. Every new team needs time to get up to speed. You need to get to know the people, you need to create a safe environment in which you can trust each other and where you have room to ask for help without being afraid of being laughed at.

Given that a team needs time to get started, why not bring a complete team that's already passed this stage? Or at least let the team itself decide who should join the team. This does mean a lot of responsibility for the team, which could be a risk if the team isn't up for it though. For instance, if they value personal friendship over creating a well-oiled team, and the best result for the customer, then this principle might backfire. Therefore the next point:

TEAM RESPONSIBILITY

If a team feels responsible for what they do (or don't do!) then the team members will likely feel more involved in the project and tend to work more efficiently while having more focus on quality. They don't build and test good software because the manager wants them to. They don't fall back to 'mortgage driven development', or working just because it pays the bills. They actually want to make something good for themselves. Because they'll feel proud when they deliver something good and are ashamed when they don't. This is all about intrinsic motivation. However, a team can only feel responsible if they actually get responsibility. This means that management should give the team a lot of freedom on how they want to do their work. However, do keep in mind, giving a team responsibility is no guarantee that they will also feel responsible. This is dependent on the team and its members. It's a matter of giving and taking.

HAVE FUN

Possibly one of the most important but least understood things within an agile world, is how valuable it can be to have fun while working. Everyone knows that people will work more efficiently and actually harder when they are having fun. Everyone would like to have fun at work (right?). Especially when working in a team it's really important to not just work from 9 to 5, but also to just have a good time as a team. So, invest in a football table, a dartboard or a game console on the working floor. Take an hour off and go to the pub together or some other fun 'teambuilding' activity. A company that understands this principle well is Google. I strongly advise you to Google the Google offices and see how they brought the principle of having fun at work to the next level! You want people to want to go to work.

CONCLUSION

I very much doubt anyone reading this article is in awe about hearing loads of new things. I actually hope most people reading this article are wondering if they really did see anything new here. Communication within any project is something so obvious, so logical. It shouldn't be something to even take the trouble talking about. Right?

That's just where things might go wrong. We tend to forget that communication, effective communication, is really quite hard! You have to work for it. Not just during the stand-up or in that one specific meeting. But all the time. Thus, continuous communication.

It's good to take a step back and look at our 'agile processes'. The communication structured in some predetermined meetings and the lack of communication often visible outside these meetings. Have we forgotten what the Agile Manifesto told us in the first place? Has agile actually failed?

I think it hasn't, but I do think we should take that step back and look from a distance at what we're doing in our work. Then conclude that we often neglect the importance of communication. It is however something worth investing time (and money) in. Because continuous communication can make software development (or basically any type of work) a lot more efficient, as well as a lot more fun. So, let's do it! Let's talk.