International Journal of Contemporary Computer Research (IJCCR), Vol.2 Issue.2 (July, 2018)
ISSN: 2600-9048

Developing Roles in Agile Software Development

Mohamed Kamel Atwa

Faculty of Computer and Information Technology, AlMadinah International University Malaysia, Selangor,

MIT163BX770,atwa712002@gmail.com

Received 16 Jun 2018; accepted 10 July 2018

Abstract

One of the consequences of the Agile principles is a drastic redefinition of the roles that exist in a software project. This paper will study the roles in three parts. In the first part we are going to see what managers traditionally in non-Agile methods do. In the second part we are going to focus on Scrum which has been very influential now days. In part number three we are going to broaden the perspective a bit not just working according to strict Scrum principles to see what other roles can exist in Agile projects particularly with other Agile methods. I will use the articles talked about the roles in agile methodologies and will study a use case of applying agile methodology in a critical web project which is a replacement of current desktop project. then I will show what can we extract from what agile present to us in addition to what is existing from very long years of effort in the field of software engineering.
Keywords: (Agile management, roles, managerial roles, technical roles, agile methods.)

1. Introduction

software development organizations have become more interested in agile methodologies, whose focus is client collaboration, individual value and adaptation to change. This interest has grown because these methodologies have shown productivity gains in several different software development project types. The choice of the most adequate software methodology for software development neither is a trivial task nor guarantees the project's success. Nevertheless, agile methodologies have caught the eye of software companies, given the evidence of the productivity increase they provide [1]. One of the most important basics in agile is the redefinition of the roles in the process of development. It is really new vision which make the project manager role less important than it was before while it split to other roles in more efficient way that lead the project to end as the client wish in the determined time. the main idea from my point of view is that with the last delivery of the last sprint of the project the complete project is already tested and working under to approval of the client

2. Roles of the managers in traditionally non-Agile methods:

we are going to review some of the many tasks and roles that managers fulfil in traditional pre-Agile projects. What do managers do? In a traditional setting they do quite a few things. Here is a list of tasks which are typical of what managers do.

1. Define goals
2. Define deadlines
3. Assign tasks
4. Provide interface with higher management
5. Provide interface with customer
6. Validate requirements
7. Decide whether goals have been met
8. Enforce deadlines
9. Coach, mentor
10. Enforce rules and methodology

One of the primary tasks of the manager is to define the goals of the project and of each iteration of the project. So that is about what a manager does. It's not good enough to promise some results. so we have to define deadlines. And usually customers are very impatient. So that's also one of the manager's responsibilities, to define the deadlines. Then on a day to day basis, a traditional manager will assign tasks. You do this today, you do that next week. The manager is also going to be responsible for providing the interface to the upper management.

This also includes often a role of umbrella, of protecting the team from undue interference with upper management. This umbrella role is of course one of the important roles of the manager. Not only do we need to interface with management in our own company, we need to provide an interface with the customer. Now the customer being total. That is to say, another division of the same company. Or we might even be doing something for ourselves as a team. But most of the time, the customer is well-defined and separate from the team. And it's part of the role of the manager to ensure that the interface with the customer's organization is smooth, and in particular, that the team is aware of what the customer wants and also the customer is aware of what the team is doing, and comfortable with it. at the end the results are up to the customer's expectations. In order for this to work one must validate requirements. Because of course the customer is going to help define requirements. But at some point, it may be some of

the requirements are unrealistic. And because the manager is responsible for defining goals and defining deadlines, it's his or her responsibility if needed to step in and say no we can't do this. It's unrealistic. Or we could do it but not within the time limits specified. , So it is important to validate the requirements. Now the goals have been defined. And at various intervals these goals will have been realized, or the team Claims. So part of these goals and so it is part of their manager to decide whether these goals have been met, or how much of the goals have been met at a particular point in the development of the project. It's not only goals that count, it's the time at which they are met. it's part of the manager's role not only to define the deadlines that is kind of the easy part but to enforce the deadlines. It's easy to say this is going to be ready in December, especially if we are in January. It's much harder to make sure that when December comes the goals have indeed been met. And of course, I'm talking about January and December, but typically in an Agile development process, and generally in a modern development process, many of the goals of more short term. And the deadlines loom on the horizon, so it's important to make sure that these deadlines are met. Now these are all management tasks in the traditional sense. In the strict sense of the term managing people, and foreseeing goals and deadlines. But managers in practice do more. Often manager is a senior member of the software profession. That's not always the case. You have managers who are more of the MBA type, who are more pure managers. But much of the time, in software projects, the manager himself or herself is a software professional. More experienced, hopefully, than the members of the team. And so part of the role of the manager is to serve as a coach and to mentor the rest of the team. A good manager is someone to whom more junior people come when they hit a snag, when they don't know what approach to use for a particular part of the problem. And as a seasoned professional, the manager is here to coach people and to mentor them. That is part of the role of many good managers. Finally, any well-organized team, and in general any well-organized company that does software development, will have defined some rules and some methodology for developing software. House rules, style rules, rules regarding the software process, rules regarding testing, and software quality assurance. In general, of course, rules are only as good as their enforcement, as their application. So it is part of the manager's role to enforce these rules. To make sure that they don't just remain ink on paper or pixels on a screen. But that the team actually applies them and abides by them. This was a brief overview of the role of the managers in traditional management before the emergence and adoption of Agile management [2], [3], [4]

3. Roles of Agile methods (Scrum):

Figure 1

Scrum is very particular about roles in a project, and in strict Scrum development there's actually three roles that have to be fulfilled, not two and not four, but exactly three, and here we are going to see what these three Scrum roles are. Scrum's major contributions are in the management area, so it's not surprising that Scrum takes a shot at completely redefining the traditional manager responsibilities that we saw in the previous part. In fact, in strict Scrum, there is no such role as a manager. The traditional tasks of the manager are split between three different roles, the self-organizing team, the product owner, and the Scrum Master. The last two are people. The first one is a group of people, which collectively takes a certain number of critical decisions. let's review those three roles in turn.

Self-organizing team:

The team is a group of people. it has responsibility for many of the fundamental managerial tasks. A Scrum team and in general an agile team, this is a theme that runs through other agile methods as well, is cross-functional. What does that mean? Well, in the agile world, they don't want to have narrow specialties. And don't want in particular to have people who feel they own a particular expertise in the project and as a consequence a particular piece of the code or the product. But here what's important is that the team is cross-functional, meaning the project don't have anyone who is in charge solely of a certain area. but may have experts, for example. may have people who know more about object-oriented methodology, about databases, about networks, or any particular specialty that is needed by the project. But no one owns any

particular area. And this means in particular in the Scrum view that we have a list of tasks to be fulfilled for a certain iteration. And whenever someone becomes available, he or she takes one of these tasks. essentially anyone be able to take on any task. That's what it means by cross-functional. No narrow specialization, no personal ownership of an area. Scrum also specifies an ideal team size based on a famous paper in psychology about magic number seven, [3] plus or minus two. the ideal size of a team is seven, plus or minus two members. Does this mean that Scrum is limited to small or relatively small projects, medium-scale projects? Not necessarily. There's a notion of Scrum of Scrums, which is intended to tackle big projects. So as the name suggests, a Scrum of Scrums is a group of individual projects that are coordinated at a higher level, but each one of them remains a standard Scrum project, with this kind of number of team members. Part of the responsibility of the team, which of course is very important, is for a particular iteration, not for the project as a whole, but for a particular iteration of the project, to select the major goals for the iteration and the results that are going to be released by this iteration, known in Scrum as a sprint. So that's of course a key role, especially since it goes with the next two. The team organizes itself and its work, so you don't have or need someone which tells the team where to go and what to do on a step by step basis. It's really the team that decides to assign the tasks to its members and organize the work. The team can really do everything it likes as long as it thinks that is going to help reaching these goals and as long, of course, as what it does fits within the rules, within the guidelines that have been defined for the project. And of course, at the end of an iteration and maybe even before, in some cases, the team will have to demonstrate its results to the product owner, who, as we are going to see, is the one who decides in the end what passes and what doesn't pass as a suitable result. So these are very important responsibilities. They correspond to much of what traditional managers do. And they are farmed out in Scrum to the collective actor, the collective role of the team. In meetings of a sprint or in meetings of a Scrum project, they are going to have to decide who is permitted to talk freely and who is just on the side-lines, because meetings may be of interest not just to the team itself, but to some other participants, for example representatives of other projects with which the team is collaborating. And there's a risk that the meetings get out of hand. In order to keep the meetings focused and short, not everyone can speak at any particular time, so there's a distinction between core participants, the team proper, and fellow travellers. The core participants are those who are permitted to participate actively in the meeting. And the fellow travellers are there to listen. And they will participate actively, but only if asked. This is also known as committed people and involved people.

Product owner:

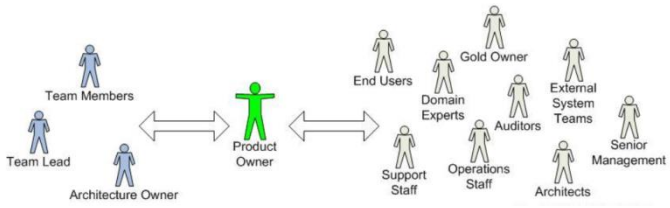

Figure 2

The basic idea of the product owner, who is a person not a group, is that he or she is responsible for ensuring the interface with the customer and making sure more precisely that the product meets the customer's needs and expectations. the product owner is going to define the product features. He or she is going to decide on the release date. the two go together, since product features are great, but only if they are delivered within the lifetime, so time is very important. As per Kent Beck's citation at the combination between timing constraints and functionality constraints [5]. The product owner decides also on the content of any major release, not just the final product, but intermediate releases as well, although he is not in charge of individual small sprints, small iterations. That is the task of the team itself, the self-organizing team. There's a strong emphasis in Scrum on business aspects and in particular on profitability. A term that recurs in Scrum discussions is ROI, Return On Investment. And the product owner is responsible for that view of the project, which is not necessarily the first one of a typical developer, making sure that a project returns and delivers value to the customer. The product owner prioritizes features. in a project typically, we want lots of features, even if, as we've seen, agile has a strong minimalistic view, trying to limit the set of features. Well, still, we might define more features than we can implement. And the need here is for prioritization. We need to know ahead of time what features are essential and what features are nice to have but less essential, so that when the going gets tough, we know what to sacrifice. And this is one of the responsibilities of the product owner, prioritization. There's a rule in sprint that a product owner can

change features and priority, but only over a limited period, typically 30 days. And the major responsibility in the end of the product owner is to decide whether a particular release or the final product has resulted in something that can be presented, can be handed over to the customer as satisfactory or not. the product owner is the one who says, yes, this is good enough or no, the team need to get back to work. We cannot give this to the customer. And of course, it's a major responsibility.

Scrum Master:

The Scrum Master is a person. Again, it's not a group, but it's a person in this case, who is responsible for enforcing the methodology, in this case, the Scrum methodology. So you can think of him or her as a kind of political commissar who makes sure that the team properly applies the defined philosophy, not only Scrum as a whole, but particular guidelines and rules that may have been defined at the level of the company and at the level of the particular project. But a Scrum Master actually does more. So here are some of the tasks. To ensure that a team is functional and productive, to enable cooperation across all roles and functions of the team, to shield the team from external interferences. , there is sometimes a natural trend for upper management and other parts of the company to meddle into the daily work of the team. And that can be quite disruptive. It's part of the role of the Scrum Master to serve as what called an umbrella to protect the team from such undue interferences. The Scrum Master, as I mentioned, is the political commissar who is there to enforce the Scrum process, in particular to organize the daily meetings, also known as daily Scrums. The planning meetings, the review meetings, he's in charge of that. The Scrum Master quite importantly is in charge of helping remove impediments. Impediment is an important notion in Scrum. It's an obstacle. It's anything that affects the progress of the team, known as the velocity, in Scrum. In Lean terms, we had this rejection of waste, this injunction to avoid waste. we can also define impediments as anything that might produce waste, which of course we want to avoid. examples of impediments are quite diverse, hardware limitations, assume a situation in which compilations and tests are taking too long because programmers don't have enough memory, so the Scrum Master may be the one who says, we need to give everyone an extra eight gigs of memory, because we're wasting time for no good reason. Another completely different example is missing requirements. in the study of waste. You stop because you don't know what to do in a particular area. And the customer expert who was supposed to give you this information is not available or is not giving it to you. So that's the task of the Scrum Master, to make sure that those kinds of blocks do not occur or are resolved if they do occur. You might be waiting for some software from some other group or from within the group, some library element that you need for your work and it's not ready yet. That's an impediment. You might have management interference or bureaucratic delays and it's going to be part of the task of the Scrum Master to fight these things and make sure the impediments go away. In strict Scrum methodology, the Scrum Master is only a Scrum Master and only does not develop himself or herself. And if there is not enough work to occupy a Scrum Master full-time for your project, the idea is that a Scrum Master shares his time between several projects for which he is only Scrum Master. To go a bit more into the assessment mode here, it's a pretty dangerous idea, because it's not so good to have in a project people who are only talkers and not doers. And we may want everyone to roll up their sleeves and participate effectively. So there's no absolute rule here. But the general observation is to be wary of people who are just advice-givers and who don't have to live up to the results of their own advice and apply it as developers. It's probably better if the Scrum Master has extra time to make him or her develop along the rest of the team.

In strict Scrum, there is no manager, no role of manager. The tasks, as we've seen, are split between the team, the product owner and the Scrum Master. On the other hand, Scrum authors realize that for some companies this is going to be too much and that they will still want to have a project manager. And so they have some advice for managers if there are any. The managers should do certain things and not do some other things. So they should support the team in its use of Scrum, like the Scrum Master does. They should contribute wisdom, expertise, and assistance, so that's kind of the coach and mentoring role. They should not assign tasks, get status reports. Such micro-management is part of the team's responsibilities. Instead they should play guru, mental coach, and so on. So we can see here that we're kind of looking at a manager who is also probably the Scrum Master, at least has some of the responsibilities of the Scrum Master. There's also the advice that managers may need to evolve their management style, for example use Socratic-style questioning to help the team discover solutions rather than imposing a solution, as a traditional Steve-Jobs-like manager or Henry-Ford-like manager might do. What we've seen in this part the three Scrum roles. Scrum, and in general agile methods, strongly curtail the traditional role of the manager. In the next part, we'll see some

other roles which replace part of the tasks in other methods. But in Scrum, we have three well defined, precisely defined separate roles, the team, the product owner, and the Scrum Master.

4. Roles of Agile methods (Non-Scrum):

Not all Agile methods are limited to the three Scrum rules that we saw in the previous part. we are going to study a few other roles that applied in Agile methods. Scrum is not the only Agile method, and other methods, in particular methods that preceded Scrum, have come up with their own definitions of roles, replacing some of the traditional roles of developers and managers and complementing some of them

Developers:

Main job: turn customer stories into working code. Developer obligations:

1- Know and understand technical issues
2- Create and maintain the system as it evolves
3- Answer: “How will we implement it?”, “How long will it take?” & “What are the risks?”
4- Work with customer to understand his stories
5- From a story, decide implementation
6- Estimate work for each story, based on implementation decisions & experience
7- Identify features that depend on other features
8- Identify risky features and report them to customer
9- Follow team guidelines
10- Implement only what is necessary
11- Communicate constantly with customers. [6]

expert user:

An interesting idea from Crystal, which of course is not entirely new since many projects have had in the past something like this, an expert user is, as the name suggests [4]. a person with expert knowledge of the project area, of the project domain, who can answer questions and even suggest solutions to problems. It's important here to find the right person. There's always a risk of getting someone who claims to be an expert and maybe is not. So, for example, if someone has too much time on his or her hands, you may be a little bit suspicious because real users typically are very busy, real expert users are very busy with doing work. You also should not just limit yourself to testers from the development team. And so what Crystal advises here is to have a minimum of once a week meetings two hour with expert users and the ability, to make phone calls any time during the week. But this is one of the ways to obtain accurate information about the needs, the actual needs, of the actual users and avoid developing something that will be recognized too late as not fitting those needs. [7]

Embedded customers:

In XP, in extreme programming, there is a strong emphasis, on the role of customers. We have seen the notion of embedded customers [6]. And we have noted that it's not such a practical idea. But still, the notion of customer is very important. So for example, customers should not meddle too much into technical decisions. Of course, these days almost everyone thinks they know something about programming and about technology. That may or may not be the case. So it's the developers who are responsible for the technology, not the customers. For all the added roles that we have in Agile methods for the customers, it doesn't mean that customers should be responsible for everything, and they should also know their place. This is basically what this text tells us a bit more politely. The customers are also responsible for helping the team analyse the risks. In particular, weighing the various user stories, the requirements elements against each other. It's difficult for the developers to know what is more critical, what is more risk prone, and what is less critical. And here the customers provide a fundamental help. The customers are responsible for providing precise user stories so that developers know exactly and, more importantly, accurately what users need. They should be the ones who say what are the stories with the maximum value so that features can be properly prioritized. And they should collaborate with the team, not treat the team as an external body, but really attempt to work closely with them and help them at every step. So in the same style, this same article which I recommend gives a number of elements of advice as to what developers should be responsible for. I'm not going to go through the entire list. I just refer to that article. But I'll pick a few examples.

Implement only what is necessary: This, of course, is part of the minimalistic spirit of the Agile approach. And the idea here is that team should refrain from own impulses, which are natural for a developer to want to do, always more to add more features, well, make sure to implement only the features that are fundamentally useful for the customer, and not just all of those that please you for intellectual elegance and value. The responsibility of the developers also include *working closely with customers to understand their stories*, not just to get started on the basis of a vague description and make interpretations that may or may not be justified, but try to understand in depth what a customer really means when he has specified a certain user story. And in general, developers should constantly communicate with customers involved in the project. It's of course part of the developer's role in this self-organizing team to estimate the work for each story. Those are some of the examples of the

important responsibilities that developers have in an Agile project. There are also a number of rights. There are notions like personal safety and responsibility. So a developer has a right to give his own estimate for his own work, to work sensible and predictable schedules. Schedule only work that can be done, not have unrealistic expectations which cannot be met and which as a result affect everyone and lower the morale of the team. Produce code that meets the customer needs. And it's also important for the developer to be shielded from having to make decisions that are really not developers' decisions, but business decisions. Of course, the boundary's sometimes fine between strictly technical and strictly business decisions, but we should be aware of these problems and not unfairly force onto developers decisions that are really management or customer decisions or business decisions.

Customer responsibilities in XP:

1- Trust developers' technical decisions, because developers understand technology

2- Analyse risk correctly, weighing stories against each other

3- Provide precise stories, enabling developers to produce comprehensive task cards and accurate estimates

4- Choose stories with maximum value, scheduling the most valuable stories that could possibly fit in to next iteration

5- Work within team, providing guidance and receiving feedback as quickly and accurately as possible

Tracker:

In some sources, you find a special role of tracker, who is a person responsible for keeping track of the schedule[6]. which is the ratio of what the team is actually doing to what it should be doing in an ideal scheme. And so this is something that we're going to track in an Agile project, in particular in a Scrum project. And we may have a special role assigned for this. And it's particularly important then to look for signs of potential problems, like changes in velocity, like too much overtime work, or too many tests that fail rather than pass before we move on. So these are numbers that measure progress, and it's important for someone to be responsible for measuring that progress.

Coach:

We have seen the notion of Scrum master, which is a bit specific to Scrum. In general, in Agile methods prior to Scrum, there was a notion such as coach, which is a role with the following responsibilities, some of which are familiar because we have seen them in the Scrum master. The coach is here to guide the team, to mentor the team, to lead by example as a good commander, to teach when necessary, and often to teach just by doing. Sometimes you just watch someone doing something right, and that's the best way in some cases to learn, to offer ideas to solve difficult tricky problems, and to serve sometimes as an intermediary with management. So, in Scrum, this is part of what the Scrum master does. what we've seen in this last part of roles is a variety of roles, not exactly identical to those of Scrum, but replacing or complementing many of the traditional software development roles.

5. Assessment and Discussion:

we have noticed that while we study the roles from traditional methodologies to the agile methodologies there are some really good ideas and some ideas which are perhaps not so good. So, we need to perform an assessment. the user assessment that matters. He is the one who have to decide what among the Agile ideas are the ones that are right for him and which ones are not so good but there are also some objective criteria. After all, Agile development didn't come into an empty world. We have had several decades of software engineering before. So, we have some empirical studies. And we have some analytical criteria to help us judge. Unfortunately, there are a number of Agile ideas which are really bad and which we have to stay away from. not everything is going to improve software development however fashionable it may be to say that we are applying Agile methods. As assessment we will discuss some of the nations around roles in the following points:

1- The idea that the method keeper or the political commissar as we described him (the Scrum Master) should not develop code. We want doers. We don't just want talkers so if he has time and don't work as Scrum Master on other projects he must have some code responsibilities. In our case the Scrum master was in a team develop and lead and guide the developers.

2- Many successful engineering projects are successful because of a strong manager. And to pretend that every team can be self-organized and will have better results if it's self-organized than if it has a strong manager is extremely detrimental.

3- Pair programming is actually a good idea in some cases It can help and it's widely applied. But it's not justifiable as the only way to develop software. In my case I used it when I have a heavy task required in a tight time so I use an expert developer with a normal developer to do it and it successes. another case I used pair programming a few days before one of the

developers is going to have his vacation so I put his replacement with him in recent tasks to make the replacement live the tasks and start to produce directly once he become the responsible developer.
4- Cross-functional teams. It's in general a good idea to have lots of knowledge shared in a team between the team members. But it's inevitable that some people will be experts in some areas. we want to avoid that a project is fundamentally dependent on one person. But we cannot ignore this phenomenon of specific expertise. In our case we success in prevent the idea of a person own a part of code and a problem of his absence. Everyone can work in any task code, but in the software organizations we cannot share the roles of DBA or operation or networks specialists.
5- embedded customer. this is a pleasant idea in principle as introduced by extreme programming. it really worked very well in our case where the customer represent was always with us in meetings or on phone to discuss any non-clear task.
6- Tests as one of the key resources of the project. a major contribution of the Agile approach to rehabilitate the notion of test and convince us that our regression test suite which we run again and again and again is one of the key assets of the project. Essentially as important as the code base. This is a major advance in software engineering.

6. Conclusion:

There has been a tendency in the Agile literature to promote Agile methods as a panacea, as a marvellous recipe. And some of the productivity improvements that are announced. It's a major improvement. But it doesn't displace the previous ideas. It complements them. Software development is hard and what really counts in the end is quality. Anything that helps produce software of better quality is going to be helpful. And Agile methods have that potential. Lots of good ideas. Agile in that can help. There is no reason to reject those from any particular style of software engineering. Particularly since we should be quite humble there since there is not much empirical data from impeccable, unimpeachable studies that show that technique A is better than technique B. We have very little of that in our studies but not enough. So, we should be very careful in our claims. And in the end, Agile methodologies roles is not a replacement for more than 50 years of evolution of software engineering methodologies roles. It's a complement for those decades and all this accumulated wisdom. so, as a conclusion Agile is a mix of good and bad ideas and whatever its limitations, whatever the criticisms that one can make it's a major step in the evolution of software engineering.

References

[1] Leitão, M. V. Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo. Monografia (Trabalho de Conclusão de Curso) – Universidade Federal de Pernambuco, Recife – 2010. Available at: . Last access: apr.2011.

[2] Jeff Sutherland, Rini van Solingen and Telco Rustenberg: The power of Scrum, CreateSpace, 2012.

[3] Jeff Sutherland: Scrum: The Art of Doing Twice the Work in Half the Time, tutorial notes, 2013, available at http://www.scruminc.com/wp-content/uploads/2014/10/CSMjsv18a1.pdf

[4] Jeff Sutherland: Self-Organization: The Secret Sauce for Improving your Scrum Team, video at www.youtube.com/watch?v=M1q6b9JI2Wc.

[5] El-Ebiary, Y. A. B., Najam, I. S. M., & Abu-Ulbeh, W. (2018). The Influence of Management Information System (MIS) in Malaysian's Organisational Processes—Education Sector. Advanced Science Letters, 24(6), 4129-4131.

[6] Kent Beck, with Cynthia Andres: *Extreme Programming explained — Embrace Change*, Addison-Wesley, 2005 (Second Edition).

[7] Extreme Programming Pocket Guide, O'Reilly, 2003.

[8] Alistair Cockburn: *Crystal Clear — A Human-Powered Methodology for Small Teams*, Addison-Wesley, 2005.