5

Agile roles

One of the most tangible and immediate effects of agile methods is to force a fresh look at the duties and privileges of project members. Agile development redefines in particular the roles of managers, customers, and the development team.

We will start with the manager's role, continue with the team and the customers, then examine other important roles specified by some or all of the agile methods.

5.1 MANAGER

The most striking prescription affects what agile managers do and particularly what they are *not* supposed to do. Much of the agile discussion of this topic is indeed negative; the manager does not:

- Assign tasks (in the non-agile world, perhaps the defining duty of a manager).
- Decide what functions to implement (also a traditional manager's privilege).
- Direct the work of team members.
- Request status reports.

Henry Ford and Steve Jobs need not apply.

The tasks listed, no longer the purview of managers, will have to be assigned to other actors as discussed in the next sections: mostly the team as a whole, but also new roles such as the Scrum Master.

What remains for the manager? Essentially, a supporting role. The tasks include:

- Establishing an environment that enables the team to work successfully.
- Ensuring a smooth interaction with the rest of the organization. In this role the manager is a **champion** of the team with higher management and other organizational units. Part of the difficulty of this task is to make sure that other divisions of the company, which may not have seen the full agile light yet, do not impede the progress of the agile project by applying old ways of thinking.
- Handling resources, including suppliers and outsourcing partners.

A popular way in Scrum circles to describe the shift is that the manager "plays guru" instead of "playing nanny".

Scrum goes further by not including a manager role at all. According to Schwaber:

There are only three Scrum roles: the Product Owner, the Team, and the Scrum Master. All management responsibilities in a project are divided among these three roles.

The next sections review these more specific roles. It is natural to ask about the consequences of removing the manager role, in particular the possible dilution of responsibility; the last section discusses this issue.

5.2 PRODUCT OWNER

Deciding on product functions is in Scrum the task of a member of the customer organization called the product owner. As stated by Pichler, the product owner *champions* the product, *facilitates* decisions about that product, and has the *final say* over these decisions.

Concretely, the principal responsibility of the product owner is to define and maintain the **product backlog**: the list of features. We are talking here of product-level units of functionality, not the individual tasks needed to implement them: these tasks will be defined by the team at the beginning of each sprint. The product owner is, however, crucially involved at the start and end of every sprint:

- At the start, to select user stories from the product backlog, and explain them in terms of their business role.
- At the end, to evaluate the result of the sprint.

The Scrum product owner role covers one of the traditional responsibilities of a project manager, deciding on functionality, but not the others: *enforcing rules* is the job of the Scrum Master; and *handing out individual development tasks* (to implement the selected user stories) is the job of the next character in our cast — the team.

The Product Owner idea is an important Scrum contribution. Its main benefit is to separate the job of defining project objectives and assessing their attainment from the day-to-day management of the project, and in particular of the tasks intended to achieve these objectives.

5.3 TEAM

The team is a group of people but, like the chorus in a Greek tragedy, can also be viewed as a single character. It takes over several traditional manager responsibilities, including the critical one of deciding, step after step, what tasks to implement.

5.3.1 Self-organizing

As we saw in the previous chapter, the team is not a group of people directed by a manager but is empowered and self-organizing.

As an example *a contrario* of these principles, Schwaber reports on his visit to a company that thought it was applying Scrum but was not doing it properly:

The ScrumMaster invited me to attend "his Daily Scrum". An alarm went off in my head. Why was it "his Daily Scrum" and not "the team's Daily Scrum"? At the meeting, he went around the room, asking each person present whether he or she had completed the tasks he had written by their name. He asked questions like, "Mary, did you finish designing the screen I gave you yesterday? Are you ready to start on the dialog boxes today?". Once he had exhausted his list, he asked whether the team needed any help from him. They were all silent. How could I tell him what I thought of his methods?

What he thought was less than flattering, of course, since they contradicted the idea of a team that decides by itself what it will do next, picking from the list of remaining tasks.

The team in agile approaches is *self-organizing*. Cockburn and Highsmith write:

Agile teams are characterized by self-organization and intense collaboration, within and across organizational boundaries. [*They*] *can organize again and again, in various configurations, to meet challenges as they arise.*

Note the key benefit claimed here: the ability to adapt quickly to new circumstances. The main task of a self-organizing team is to decide what to do next. In Scrum this means picking from the task list ("sprint backlog") the next task to be implemented.

The agile literature goes to great lengths to explain that self-organizing does not imply rudderless: in some methods at least the manager still has a role to play, as discussed in the previous section, but this role does not include meddling in everyday decisions such as picking the next task.

5.3.2 Cross-functional

Another recommended characteristic for agile teams is to be cross-functional. The Poppendiecks write:

Agile development works best with cross-functional teams [*which have*] *the skill and authority necessary to deliver useful feature sets to customers independent*[*ly*] *of other teams. This means that whenever possible teams should be formed along the lines of features or services.*

The rejected alternative is a division into teams organized along areas of competence, for example a hardware team and a software team (for an embedded system), or a database team and an application logic team. The recommendation is instead to use a division along user-visible subsystems, each covering a subset of the functionality, in line with the reliance on user stories to define that functionality. For example part of the team might be in charge of the scenario "process a new purchase order" and another part in charge of "cancel purchase order", even if the basic infrastructure is shared.

Such an assignment implies only a temporary responsibility associated with a particular task, not a long-term specialization, even less any exclusivity. In a fully cross-functional team, any developer should be able to go to the task list and pick the next task, whatever it is, that the team has deemed to be of highest priority. The presentation of agile roles will discuss the benefits and limitations of cross-functional teams.

5.4 MEMBERS AND OBSERVERS

The agile world and Scrum in particular make a distinction, for any project, between two kinds of participants: those who are truly *committed* to the project, in the sense that its success is critical for them, and those who are also *involved* but from the sidelines. The accepted terms are respectively "pigs" and "chickens", a terminology that comes from a vulgar joke repeated in a zillion publications and not worth including here. With or without zoology, the concept is hardly new: committees routinely distinguish between *members* and *observers*. Another possible terminology would be "core participants" versus "fellow-travelers".

The distinction matters in particular for daily meetings, where the roles of the two categories are delineated: the members should dominate the discussion, with observers standing on the side. The observers will give their opinion if invited to do so, but actual project decisions, such as including or rejecting functionality, are the privilege of members.

5.5 CUSTOMER

We have seen, as one of the method's principles, that agile methods put the customer at the center. A concrete consequence is to emphasize the role of the customer throughout the project and — in some cases — the role of the customer *as a member* of the project.

Traditional development approaches also strive to build a system that will please its customers, of course, but they limit customers' involvement to specific phases at the beginning and end of the lifecycle; in the extreme form represented by the "V-model" variant of the waterfall, those would be the top-left and top-right phases.
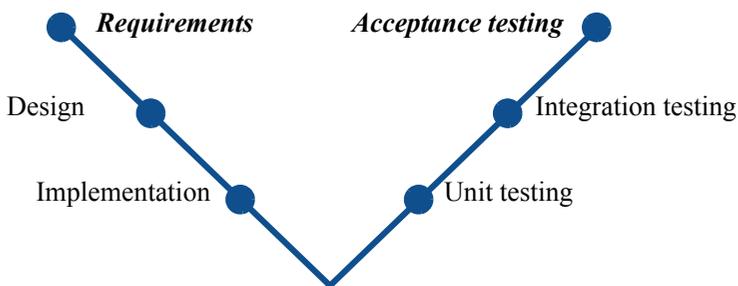
Simplified V-model of the software lifecycle

The simple V-model illustration shown here is not the most common one; usually implementation figures at the bottom, which makes little sense since it is the direct counterpart of (on the verification side) unit testing. In addition, some variants have more phases than shown here.

Even with an upfront requirements phase, many opportunities often arise later in the project for the developers to obtain more information from customers. Some project environments discourage such contacts or even prohibit them. Requiring that they happen through organized channels is reasonable, if only because — as mentioned in the discussion of the customer's role — different stakeholders have different views and you need to make sure you are talking to representative people. But *disallowing* any interaction between developers and customers is a sure way to obtain systems that do not meet customer objectives. Agile methods go further and *require* customer interaction.

While the basic idea is common to all agile approaches, the level of customer involvement differs. Extreme Programming, as explained by Ron Jeffries, directs the team to include a customer representative, part of the "whole team" experience:

> *The team must include a business representative — the "Customer" — who provides the requirements, sets the priorities, and steers the project. It is best if the Customer or one of her aides is a real end user who knows the domain and what is needed.*

[Jeffries site], xpmag/whatisxp #whole.

This role does not appear explicitly in Scrum, since the product owner is the person responsible for representing users, as part of the more general task of conveying to the team the business goals of the project.

Once one accepts the idea of including customer representation in the team, the Scrum approach is superior to the XP notion of an embedded customer representative. There is evidence (anecdotal rather than based on systematic studies) that it is difficult to integrate even a well-meaning customer representative; sometimes the formula jells, but often the representative feels left out, since much of the interesting stuff occurs in technical discussions which he cannot easily follow; and a good deal of the time he just sits bored. In addition, a customer representative with no decision power can do harm as well as good. It is difficult to determine how much he represents the needs of the customer as a whole, and how much just his own. The odds are not good: think of the kind of person whom an organization would wish to assign full-time to a project but without any decision power (taxation without representation, as it were); is that going to be the most competent expert of the application domain? Probably not: such people are typically in high demand and very busy — with application domain tasks. Whoever has enough free time to be posted to a development group for many months may raise some suspicion: is the customer organization trying to help you, or to get rid of someone?

With the Scrum notion of product owner, you also get a customer representative, not necessarily full-time, but with a clearly acknowledged strategic decision role: defining the last word on what goes into the product and what does not. This role justifies putting at the project's disposal a product owner who truly understands the business and will provide operationally valuable input to the developers.

5.6 COACH, SCRUM MASTER

Agile methods raise frequent problems in their daily application and require enforcement, lest the team stray from the recommended principles. Sometimes the project manager plays this role, but the recommendation is to assign it to a specific individual: a *coach* in Extreme Programming; a *Scrum Master* in Scrum.

Larman encourages putting in place a "central" coaching team which advises many different groups. He also insists that the role of coaches should be to advise, not prescribe; this view is in line with the agile mistrust of consultants or managers who tell everyone what to do but are not ready to do some of the real work themselves. *[Larman 2010], page 399.*

"Coach" suggests a training role. Scrum Masters, in addition, take on a management role. The border can be thin; as Cohn writes:

> *A ScrumMaster may not be able to say "You're fired", but* **can** *say "I've decided we're going to try two-week sprints for the next month".* *[Cohn 2010], page 399.*

More generally,

> *The Scrum Master is responsible for making sure a Scrum team lives by the values and practices of Scrum.* *[Schwaber 2012], page 164.*

But the role goes beyond that of a political commissar; one of the primary tasks is to **remove impediments** identified by team members in daily meetings. An impediment is any obstacle, technical or organizational, that prevents the team from operating at full productivity (implementing as many user stories as possible). Some impediments are technical, such as a developer getting stuck because he does not know of an appropriate algorithm to solve a certain task; others are political or organizational, such as computers choking up on not enough memory or a subcontractor failing to deliver a component of the system.

The Scrum Master is also responsible for protecting the team from distractions and undue interference from the rest of the organization, since it is an agile tenet that developers should be able to concentrate on one task at one time.

The Scrum Master concept has met with considerable success. Some of that success is due to non-technical factors: to be worthy of consideration as a Scrum Master you should be a *certified* Scrum Master, meaning that you have followed appropriate training and paid your fee. This certification aspect of Scrum is good business. It provides a self-reinforcing loop: certified masters are natural advocates for the method, and the more companies they convince the more Scrum Masters will be needed.

For a new method, the basic concept of having a coach to help apply the method right is sound. More debatable is the expectation that a Scrum Master will do only that job, and will not be a developer. While staying away from absolutely ruling out such a possibility, agile authors clearly state that a Scrum Master should only be a coach; if the project is too small, rather than doubling up on other duties on the project, the Scrum Master should double up on projects, coaching several teams. Scrimshire writes of the risks of a coach who also programs:

> *Being directly involved in the work, being an agent in the system, being directly affected by difficulties arising in the team means the Scrum Master could lose objectivity. They could be too close to a problem to be able to coach the team effectively.*
>
> *As a developer, there is opportunity for directive or controlling behavior to creep in. Is the developer of sufficient character to be able to retain a sense of objectivity and unbiased questioning in the role coach or facilitator? If the developer had a differing technical opinion with the team would they be willing to accept the team's approach or mandate?*

[Scrimshire site].

My experience runs directly against this advice. I have seen too many times the sad spectacle of advisors who do not want to dirty their hands. That is what is so great about being a consultant: if the project succeeds it is thanks to your wonderful advice; if it fails it is for not following it properly. In the Scrum case, consultants make it even easier for themselves because the Scrum Master also stays away from programming but from the other core responsibility-laden task: management.

In traditional settings developers typically do not have much respect for advice-only consultants. There is still enough reverence around agile methods and Scrum that advice-only Scrum Masters are taken seriously. The hypnotism will not last forever, and companies will focus on work that brings real benefits. (Even the Red Army no longer needs political commissars.) Already today, not everyone buys the idea; a reader from India commented, à propos Scrimshire's article cited above:

> *I have seen the trend that organizations look forward to hire people with technical skills. Specially in India, they do not consider Scrum Master as [an] independent role but always club with developer (they call it technical scrum master).*

It is good to encounter some common sense, at least in India. A Scrum Master who also programs has the advantage of being close to the problem; "*too close*" perhaps, but it beats being too far. There is nothing like having to wrestle with the toughest part yourself to know how to advise the rest of the team.

Assigning the coaching role to a manager, rather than a developer, also makes sense. A good technical manager should be experienced enough to serve as coach; this is one of the traditional roles of managers, and there is no clear argument for not continuing it when the personalities involved fit the bill.

Harlan Mills developed long ago the concept of chief programmer: the project manager
who just happens to be the best programmer on the team and in addition has management
capabilities and like a general who has risen through the ranks leads the team into battle.
The chief programmer is a technical manager, but one who is not afraid to roll up sleeves
once in a while and do the design and implementation for the toughest parts of the system.
This technique is not for every team — if only because good potential chief programmers
are few — but can be effective with the person and team. A good chief programmer will
also play the role of coach.

5.7 SEPARATING ROLES

What should we make of the Scrum insistence on three and exactly three roles (Scrum
Master, Team, Product Owner)? As usual, there is something to be taken and something
to be left.

The most interesting idea is the separation of the *product owner* role from other man-
agement responsibilities. In many contexts it can indeed be helpful to hand out to two
different persons (or groups, such as "the team" in Scrum) the tasks of :

• Directing the project, day after day.

• Defining what it must do for the business, and assessing whether it actually does it.

This distinction is applicable in projects where no one is equally at ease with the business
and technical sides. Such a situation arises in enterprise-style projects ("business" or
"commercial" data processing), the area from which agile methods seem to have drawn
most of their experience. In a technical company, and particularly in a software company
— Microsoft, Google, Facebook… — the classic distinction between "the software" and
"the business" disappears, since the business *is* software and often the software is the
business. In such environments one can often find an executive who is both thoroughly
attuned to the business needs and perfectly capable of leading the project. If you intend
to have a project manager — an idea anathema to Scrum and most other agile approaches
— that person may also be qualified to serve as the product owner.

The argument *against* merging the manager and product owner roles is the risk of
being, in Scrimshire's terms, "*too close to the problem*". He invoked that risk as a reason
to separate the roles of *developer* and *coach*; we saw that there is in fact little cause for
concern in that case, but the risk becomes more serious if we consider the roles of *project
manager* and *product owner.* The manager could become so involved with the project —
so "embedded" in it — as to develop a kind of Stockholm Syndrome and lose track of
the needs of the business, which are the reason the project exists in the first place. A dis-
tinct "product owner" will not succumb to that temptation, and will provide an indepen-
dent check on the project's real progress.

The decision — assign two people as manager and product owner, or keep the roles
separate — is a tradeoff between consistency, favoring a single project manager defining
a clear vision for the team, and independence, favoring the inclusion of a second view-
point. Every project must examine that tradeoff in light of its own circumstances; there
is no universal, dogmatic answer.

Many projects, especially when they have limited resources, consider other mergings:

- It may be legitimate — not just in India — to let one of the more experienced developers double up as coach (Scrum Master).

- The manager can also be the coach. This is particularly appropriate, and common, when the manager is a *technical* manager, in the "chief programmer" style, who has more experience than the rest of the team and is naturally qualified to serve as mentor and coach in addition to performing management tasks.

- On the other hand it makes no sense to merge the "coach" and "product owner" role (if the latter is distinct from "manager"). A separate product owner should represent the business needs and not meddle into how the team works.

More generally, while ensuring the presence of a method coach in the project is often a good idea, insistence on keeping it a separate role is not. No doubt it is a good business strategy for consultants; but businesses, their budgets and their projects are better off with doers than with talkers.

Agile!
The Good, the Hype and the Ugly
Meyer, B.
2014, XIX, 170 p. 15 illus. in color., Softcover
ISBN: 978-3-319-05154-3