The Manager and Scrum

goodagile>

by Pete Deemer

Important Note

A thorough understanding of the principles and practices of Scrum is recommended prior reading this guide. We recommend *The Scrum Primer,* available for free at www.scrumprimer.com.

About the Author

Pete Deemer is a well-known figure in the Agile community, and has spent the last 20+ years leading teams building products and services at global companies. Pete is the founder of GoodAgile, and is the lead author of The Scrum Primer, one of the most widely read introductions to Agile development, as well as The Distributed Scrum Primer, a guide to multi-location Scrum.

Pete has provided Scrum training and consulting to some of the largest companies in the world, including services companies like Infosys, Wipro, TCS, Cognizant and Mindtree, and product companies like Microsoft, SAP, Dell, GE, HP, EMC, Ericsson, CA, Fidelity, JP Morgan, Mercedes Benz, Phillips, Oracle, Siemens, Unilever, and Verizon. Pete was Vice President of Product Development for Yahoo!, where he led Yahoo's first adoption of Scrum, and he was the co-founder of GameSpot, which was acquired by CBS Interactive.

Pete is an honors graduate of Harvard University, and has spent a number of years as adjunct faculty at University of California Berkeley and the Institute of Systems Science at the National University of Singapore.

THE MANAGER AND SCRUM

When an organization starts to explore Scrum, there's often an uncomfortable moment early on when someone points out that the role of "manager" seems to be missing entirely. "Well I guess we'll have to just get rid of 'em all!" wisecracks one of the developers, and all the managers in the room shift uncomfortably in their seats.

Scrum defines just three roles – Product Owner, Team, and ScrumMaster – and the basic direction given to others in the organization is to "support them, or get out of their way". This is not very detailed advice, especially if you're a manager expected by senior management to ensure everything goes well.

The traditional role of the manager in the corporate world is based on a model known as "command and control". Here, the job of the manager is to identify what needs to be done, to issue detailed instructions to the employee, and then to ensure the employee completes the work according to the instructions. The role of the employee in this model is simply to follow the directions as given, trusting the judgment and wisdom of the manager to ensure that the right work is being done in the right way.

In complex, dynamic environments such as software development, this approach tends to break down. First, it is difficult and time-consuming for a manager to understand every requirement in full detail and issue precise instructions to guide the work of every employee. Within a software development Team, the work is highly interconnected, with intricate dependencies, and frequent change and surprise. To expect a single manager to do all the basic thinking for his or her team is unrealistic, and it often constrains the team's productivity to the manager's ability to give instructions. In addition, this approach tends to be demoralizing for employees; their role is simply that of "order follower", and they often feel little sense of ownership of their work. Accountability is limited to answering the simple question, "did I complete the orders I was given?" If the answer is "yes", the job has been done – regardless of whether the right thing was built, built well, or built to satisfy the business goals of the customer.

Scrum is based on a different approach: The Self-Organizing Team. The difference is apparent from the first steps the Team takes. In Scrum, the Team decides how much work to commit to in a Sprint. Experience has shown that when Teams themselves decide how much to commit to – and when this commitment is realistic and achievable – the Team's focus, motivation, and drive is significantly higher, and they produce better results. When managers first learn of this practice, they often voice the concern, "What if the Team under-commits?" This is typically not a problem, since the process of deciding the commitment is very transparent and open. Indeed, it's much more common in the early Sprints for Teams to significantly overcommit; most Teams have very little experience doing their own estimation, and it takes a number of Sprints before their optimism is tempered by experience. Moreover, in the event the Team does wind up under-committing, they will either finish the Sprint early, or they will start work on the next item on the Product Backlog. No harm, no foul.

Team Tango had just completed their first Sprint Planning Meeting, for a two-week Sprint. They brought in their manager, Jason, and walked him through the work they'd decided to commit to in the Sprint.
Finally, they asked Jason, "Is this a good amount to commit to?"

Jason turned the question around to the Team:

"Do you truly believe you can finish this work, at a high level of quality, by the end of the Sprint? Do you really feel committed?"

The Team members all nodded to him, looking quite convinced.

"Then it's a good amount to commit to." Jason replied. "And if it turns out to be too much or too little, you'll know two weeks from now, and you'll have learned something about how much you should commit to in the following Sprint".

The next aspect of self-organization happens during the Sprint, when the Team works together closely to decide who will perform which tasks, and to make sure all the work is completed. When the Team is responsible for this decision-making, they remain focused on the fact that they own the commitment – and if the commitment is to be completed, they are the ones who must do it. When someone outside the Team is responsible for deciding what needs to be done and by

whom – for example, a manager – the Team receives a subtle but real signal that they are *not* responsible: it's the manager's job to worry about how to meet the commitment, not the Team's. This does not mean that managers are not providing support during the Sprint – on the contrary – but managers are careful not to send any signal to the Team that would reduce their sense of ownership of the goal, or responsibility for managing themselves during the Sprint.

On the first day of their first Sprint, the Team called their manager Sanjay over to join them for their Daily Scrum Meeting. Sanjay, wanting to be helpful, agreed to the request. He stood just outside the circle as the Team gave their reports to each other. Sanjay noticed that people seemed to be emphasizing how much they got done the day before, and weren't spending very much time reporting the blocks they were hitting. And after each person gave their short report, they looked over to Sanjay expectantly, hoping to catch a glance of approval. By the end of the Daily Scrum, Sanjay noticed that the entire circle of people had shifted, so they were now facing him.

After the last report was given, a Team member raised his hand, and asked "Sanjay, do you have any feedback or guidance for us?"

Sanjay knew that he had to take action.

"Guys, I'm really concerned. I feel like this meeting was for my benefit. I feel like you're still looking to me to make sure everyone's doing the right thing. Here's the deal: I'll give you any help you need, at any point in the Sprint. If you hit a block and you're not able to resolve it, I'm here to provide any assistance I can. But at the end of the day, <u>you</u> are responsible for doing what's necessary to meet the commitment <u>you've</u> made. So from now on, I'm not going to join this meeting. This is <u>your</u> meeting, to manage <u>your</u>selves, to meet <u>your</u> commitment. If I'm here, I'm afraid I'm just going to undermine that."

The Team was silent. Then Victor, a Team-member, spoke up.

"So let me get this straight. <u>We</u> are the ones responsible here? We really do own this…?"

One of the biggest challenges in successfully making the transition to self-organization is that the Team will not begin to self-organize until *everyone* outside the Team *stops* micromanaging them. Teams are so conditioned to follow orders that they will often not begin to self-organize until there are no orders available to follow. This requires a leap of faith for the manager, and it can be scary. This is not to say that the manager abandons their Team – rather, the manager needs change their style of interaction, and constantly signal to the Team that they are now the ones responsible.

Eileen was an Engineering Manager at RedAlpha Systems, working with a Team of 7 relatively junior developers. During the first Sprint Planning Meeting, she sat at the back of the room working on email, as the Team completed the task breakdown for a big feature at the top of the Product Backlog.

When they finished, they turned to Eileen and said "How does this look to you?"

Eileen could see immediately that the Team had overlooked several important database tasks. It would be very simple for her to simply point out the tasks they'd overlooked, and the problem would be solved. Or would it?

Eileen decided to try a different approach. She stood up and announced, "Guys, you've done a pretty good job, but you're not quite done. There are a couple important tasks that you've overlooked. I'm not going to tell you what they are. But I will give you a hint: Think more carefully about the user session data. Now I'm going to go and grab a cup of coffee, and I'll be back in about 5 minutes. See if you can figure it out before I get back."

And at that, Eileen strode out of the room.

The Team looked at each other, slightly bewildered. Eileen had always been quick to point out what they'd missed; they depended on her for that. But this time, she was making them

> *figure it out. They stood in silence for a moment at the whiteboard, then slowly*
>
> *discussion began. They went through task by task, looking at each from different angles. Then, after a few minutes of discussion, Tony spoke up.*
>
> *"Wait a minute… where are we going to store the user session data? We're going to have to set up a new table in the database for that, right?".*
>
> *There was a round of forehead slaps from the other Team-members.*
>
> *"Of course! How did we miss that!" several people murmured. There was a chuckle of embarrassment, and Sam started writing yellow Post-It Notes for each of these new tasks and putting them on the white-board. A few minutes later, Eileen returned with her cup of coffee. She looked at the white board, and nodded in agreement.*
>
> *"Good job, guys. Now why don't you all continue with your meeting, I've got a bunch more emails I need to get through."*
>
> *Eileen sat back at the end of the table, satisfied that she'd done her job well.*

In this example, it would have been faster and easier for Eileen simply to tell the Team what to do. But had she done that, she would have encouraged them to wait for solutions from her, and not think for themselves. Instead, Eileen did something harder, but ultimately much more valuable: She placed the responsibility on the shoulders of the Team to figure out what they had forgotten, and provided just enough help to enable them to get it done. Had Eileen returned to find the Team still struggling, she could have provided another hint or asked another probing question, and continued to do so until the Team finally figured out the missing tasks. Eileen could even have let the Team proceed, and discover their oversight during the Sprint; mistakes often produce the most powerful learning experiences.

In simplest terms, the manager in Scrum is less of a "nanny" for the Team and more of a mentor or "guru", helping them learn, grow and perform. This is the shift from "Manager 1.0" to "Manager 2.0".

In order for managers to be effective in this new mode, the organization must redefine the role and expectations of the manager. For example, in Scrum, the Team is responsible for completing their commitment in the Sprint, and for this to work, it must be clear to all that the manager is <u>not</u> responsible for this. Similarly, in Scrum, it is the Product Owner's responsibility to deliver the release on schedule, not the responsibility of engineering management, and the organization needs to make clear to everyone that this is the case. Problems occur when the organization "talks the talk" on the new role of the manager, but does not "walk the walk" when things get difficult.

The Galaxy Team had been doing Scrum for several months, and the Team was well on its way to being truly self-organizing. Their motivation was high, they were focused, and after a few Sprints of under-delivery, they were now showing a pattern of making reasonable commitments and delivering them 100% each Sprint. Morale was high, and there was a real sense of "flow" in the work they were doing. The engineering manager Francis had come a long way – once a habitual micromanager, he was now acting like much more of a mentor and coach for the Team. Unfortunately, though, in the eighth Sprint, the Team encountered some unexpected difficulties, and about halfway through the Sprint, they were significantly behind in their progress. The VP of the group, Simon, had ventured into the Team's work area to see their Sprint Burndown Chart, and called Francis to his office.

"Francis, it looks like this Sprint is a disaster. What's going on?" he asked.

Francis responded, "Well, the Team hit some bumps along the way, and they're trying hard to get everything done that they committed to, but it's a bit touch-and-go right now."

Simon grimaced.

"Francis, this project is critical, and we can't let it fall behind. I'm counting on you to make sure the Team finishes what they

commit to, this Sprint and every Sprint. As a manager, your job is to make sure the Team gets it done; if things are going well, then you can back off a bit, but the minute the going gets tough,

I want you in there making sure that no time is being wasted, and everyone is doing exactly what needs to be done."

Francis was exasperated. Simon had been too busy to attend the in-house Scrum trainings, but Francis had emailed him a Powerpoint presentation about self-organizing Teams and the new role of the manager, and Simon hadn't voiced any disagreement. Francis spoke up:

"But what about the self-organizing Team, Simon? What about our shift away from micromanagement?"

There was a glimmer of recognition, as Simon recalled a Powerpoint he'd seen a few months before.

"Yes, the Team is responsible, but when they start to fail, I hold you responsible. We want maximum accountability, so I'm holding them accountable and I'm holding you accountable. In our department, everyone is accountable! Now make it happen."

At that, Simon spun his chair around and started typing. Francis took the hint and left the office.

The next day, Francis showed up at the Daily Scrum Meeting.

"Guys, we're going to do a different format for the meeting today. Due to the criticality of this project, Simon has instructed me to more actively… uhhh…
'facilitate' your self-organization during the Sprint. So what I'd like to do this morning is get a status update on each of the features you've committed to – whose done what so far, and what's left to be done – and I'm going to be giving some more detailed feedback so hopefully we can get everything 100% finished by the end of next week."

The Team looked at each other. Philip, the ScrumMaster of the Team, spoke up.

"Francis… uhhh… does this mean that the Team is no longer responsible for managing itself?"

Several Team members nodded in agreement.

Francis replied, "Guys, we're all responsible. You're responsible for managing yourselves, and I'm responsible for making sure you get everything done. We're all responsible together!" Francis didn't see the eyeballs subtly roll.

As the Sprint proceeded, Francis was more and more involved. The Daily Scrum became an update meeting for the Team to tell Francis what they'd been able to complete, and for him to assign them the next day's tasks. The mood of the Team shifted; motivation seemed to go down, and Team members seemed to be

reverting to their previous role, what they used to sarcastically call "servants-of-Francis-the-Great". By the end of the Sprint, the Team

was fully back into "order-following" mode, and Francis was directing their efforts task-by-task.

At the Sprint Review, the Team was surprised when Simon joined the meeting just as it was starting.

"So…" Simon announced, "Did we get our commitment done?"

The Team looked at each other. Francis answered.

"Simon, unfortunately there are a couple backlog items that weren't finished."

There was a flash of anger in Simon's eyes.

"How did this happen? Who is responsible for this?"

The Team was silent, but their heads all turned slowly to Francis.

Simon continued. "Francis, I told you to get it done. Next Sprint, I don't want to see this happen again. If it does, there will be hell to pay…"

Upon hearing this, everyone on the Team made a mental note to think very carefully about just how much to commit to in the next Sprint. The last thing they wanted was to get shouted at again two weeks from now.

As the Sprints passed, Francis became more and more involved in directing the Team at every stage of their work. Gone was any semblance of self-organization, and with it disappeared the improved motivation, drive, and focus that the Team had started to display. Morale had plummeted, and so too had productivity. Lunch breaks were getting longer, coffee-breaks more frequent,

> *and Francis felt like he was spending more and more of his time just making sure people were at their desks working. Those amazing*
>
> *few Sprints, when the Team was truly self-organizing, and performing at the level they were really capable of, were becoming more and more of a distant memory. The return to micromanagement was made all the worse because they'd had a taste of the self-organization "good life".*

There were errors of judgment at every step of this situation. The ScrumMaster didn't protect the Team from Francis' micromanagement, or call Francis on the "double-talk". Francis didn't make any effort to reason with Simon, or help him see the consequences of his actions. But perhaps the biggest mistake was an early mistake: Simon was never properly educated about the shift in the management model that Scrum requires to be successful, and how this applies not only in good times but also when the going gets tough; and this shift was never made "official" in the form of a change to Francis's job description. And as a result, a successful, high-performance Scrum Team rapidly deteriorated back to its previous under-performing state.

The above scenario is extremely common and is a frequent point of failure for Scrum transitions. Furthermore, in an organization where this scenario plays out, word spreads very quickly, often causing other managers to proactively return to micro-management as a self-protective measure. So how does one prevent this kind of failure from occurring?

First, one has to make a clear-eyed assessment of management's willingness and ability to change, at every level. If there exists a fundamental belief in the effectiveness of the "command and control" approach within the management and executive ranks, and a heavy dependence on intimidation, threats, or punishment (such as shaming or humiliation) as a management tool, it is going to be particularly difficult to make the transition to a new way of thinking. As a result, an adoption of Scrum risks being incomplete and dysfunctional, producing little if any improvement for the organization.

However, if there is an openness to change, and a recognition that the existing command and control habits may possibly not be the most effective approach, then there needs to be education and coaching at <u>every</u> level of management; in practice, this means high-quality Scrum training for all managers, from the lowest functional manager all the way up to the senior members of the organization.

The final necessary step for completing this redefinition of the role of the manager is to "make it official" within the organization. One option is to use the pre-written job description included below as guide. The other option is to complete the interactive exercise that follows with managers in the organization, to break down their existing job descriptions and rebuild them to be compatible with Scrum values and practices. With either of these approaches, it is critical to get formal approval of the manager's new job description from *his or her manager* (for example, the Engineering Director, or department head). Without a clear, "official" approval from senior management, the manager's new role will be more difficult to protect when difficulties arise.

THE MANAGER AS SCRUMMASTER

Another approach to redefining the role of the manager is to convert them into the ScrumMaster for their Team. This has a poor track record of success. When the manager plays the role of ScrumMaster, it's highly unlikely the Team will ever begin to self-organize. The previous habits of "order-giver" and "order-follower" are very difficult to break, and what will likely happen instead is that pre-existing command-and-control values and patterns will be transplanted into the heart of the Scrum practices. As a result, the benefits that flow from a self-organizing Team – ownership, focus, drive, pride in quality, improved morale, and better productivity –will likely not be realized. It would be better in most cases to have a Team-member play the role of ScrumMaster, even if they must do this in parallel with development responsibilities.

HANDS-ON: REDEFINING THE MANAGER'S ROLE

Step 1. Ask the manager to write down all of their current job responsibilities on Post-It Notes. The manager should try to be as comprehensive and complete as possible, including both official and unofficial responsibilities, and things they should be doing but haven't had time to do. Here's a sample list:

Decide what work needs to be done	Assign the work to Team members	Keep track of what everyone on the Team is doing	Make sure the Team gets their work done	Give input on what features / functionality the Team should build
Give input on how to make features better	Help remove blocks that the Team is not able to resolve by themselves	Make commitments to mgt about how much Team can do by a certain date	Be responsible for the Team meeting the commitments made to management	Provide advice and input to the Team on technical difficulties that come up
Do weekly Team staff meeting	Do weekly status update report for management	Have regular 1:1 meetings with Team to provide coaching and mentoring	Plan training and other skills development for Team-members	Do career-development and career planning with Team-members
Recruit, interview and hire new Team-members	Stay up to date on developments in tools and technologies	Stay up to date on industry news and develop-ments	Remove Team-members who are not able to perform well within the Team	Plan and manage budgets and financials
Anticipate tools, skills and other future needs	Do performance evaluations and provide feedback to Team-members			

Step 2. Draw two columns on the whiteboard: "Good in Scrum" and "Conflicts with Scrum / Not Needed in Scrum". Ask the manager to go through the Post-It notes one by one, and place them in one column or the other.

Good in Scrum

Conflicts with Scrum or Not Needed in Scrum

1 Help remove blocks that the Team is not able to resolve by themselves	2 Provide advice and input to the Team on technical difficulties that come up
3 Do regular 1:1 meetings with team-members, to provide coaching and mentoring	4 Give input on how to make features better
5 Stay abreast of developments in tools and technologies Team is using	6 Plan training and other skills development for Team-members
7 Stay up to date on industry news and developments	8 Anticipate tools, skills and other future needs

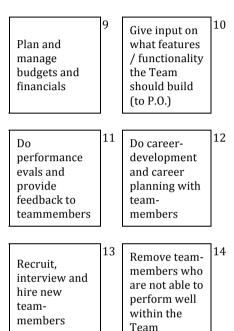|	
15 Decide what work needs to be done	16 Assign the work to Team members
17 Keep track of what everyone on the Team is doing	18 Make sure the Team gets their work done
19 Make commitments to mgt about how much Team can do by a certain date	20 Be responsible for the Team meeting the commitments I've made to management
21 Do weekly status update report for management	22 Do weekly Team staff meeting

Good in Scrum (cont'd)

Plan and manage budgets and financials	9
Give input on what features / functionality the Team should build (to P.O.)	10

Do performance evals and provide feedback to teammembers	11
Do career-development and career planning with team-members	12

Recruit, interview and hire new team-members	13
Remove team-members who are not able to perform well within the Team	14

Step 3. Take all the items in the "Fine in Scrum" column, and turn them into a new job description for the manager.

Step 4. Ask the manager, "Will you be more useful or less useful to the organization in this new role?" and "Will this role be more interesting or less interesting for you to do?" In most cases, the immediate response will be "more useful" and "more interesting".

Step 5. Get formal approval of the manager's new job description from *his or her manager*. This is a critically important final step. Without formal agreement, the manager's new role will not be considered "official", and there will be an even greater risk of falling back into prior patterns of micromanagement and command and control.

Good in Scrum

1. Help remove blocks that the Team is not able to resolve by themselves
While the ScrumMaster does this hour-to-hour / day-to-day, managers will need to focus on removing more systemic or companywide blocks. These are often the most vexing problems in the organization, and will require the management's influence, authority, or spending power to overcome. In *The Enterprise and Scrum,* Ken Schwaber recommends creating an enterprise transition team of managers and executives, who are responsible for evolving the organization based on a backlog of impediments.

2. Provide advice and input to the Team on technical difficulties that come up
Managers should be available to give advice or assistance whenever the Team asks for it.

3. Do regular 1:1 meetings with Team-members, to provide coaching and mentoring
Managers should spend 1:1 time with Team-members, at a frequency that feels right. This is not a task update meeting – this is time for coaching and mentorship. Some managers like to do this sitting side-by-side, writing code!

4. Give input on how to make features better
This input goes directly to the Product Owner, typically during the Sprint Review.

5. Stay abreast of developments in tools, technologies, and techniques the Team is using
A very important and often neglected activity. Managers are can sometimes be "frozen in time" at the technology and development practices that were current when they were last doing actual development themselves.

6. Plan training and other skills development for Team-members
Managers should think carefully about areas where the Team's skills

could use development, or capabilities the Team will need to have to handle upcoming Product Backlog items.

7. Stay up to date on industry news and developments
Again, an important and often neglected activity.

8. Anticipate tools, skills and other future needs
Another important and often neglected activity. Be sure to get input from the Team.

9. Plan and manage budgets and financials
Another important and often neglected activity. Be sure to get input from the Team.

10. Give input on what features the Team should build
This input goes directly to the Product Owner.

11. Do performance evaluations and provide feedback to Team-members.
A necessity within most organizations (despite well-documented flaws in the methodologies typically used). Managers should base their evaluations on their own observations and well as on feedback from the employees' fellow Team-members.

12. Do career-development and career planning with Team-members
Career opportunities are one of the most significant valuable forms of compensation people receive from their employer.

13. Recruit, interview and hire new Team-members
Some of the best – and in other cases, worst – management actions are hiring decisions. Great hires pay dividends every single day they are employed – and poor hires are an invisible daily "tax" on the Team's ability to deliver business value.

14. Remove Team-members who are not able to perform well within the Team
If even after extensive coaching a Team-member is not able to contribute, work harmoniously with other Team-members, or perform at the level required, they may need to be moved off the Team, or out of the organization. Typically managers will need to guide this process, in coordination with HR.

Conflicts with Scrum or Not Needed in Scrum

15. Decide what work needs to be done.
The Product Owner decides the features and functionality that needs to be built, and the Team determines what tasks are necessary to deliver this.

16. Assign the work to Team members
The Team does this itself, during the Sprint.

17. Keep track of what everyone on the Team is doing
The Team does this, using the Daily Scrum Meeting and the Sprint Backlog.

18. Make sure the Team gets their work done
The Team is responsible for this.

19. Make commitments to management about how much Team can do by a certain date
The Product Owner measures or estimates the Team's velocity, and makes forecasts of how much of the Product Backlog the Team can complete by a specified date. If the Product Owner makes a hard-date release commitment, the Product Owner is responsible for including the necessary scope and schedule buffer to achieve it.

20. Be responsible for the Team meeting the commitments I've made to management.
The Product Owner is responsible for making decisions about what to do if velocity is lower than anticipated – either moving the release date, removing Product Backlog items, or simplifying Product Backlog items.

21. Do weekly status update report for management
Not needed in Scrum. If management wants to know how the project is going, they ask the Product Owner for the Release Burndown chart.

22. Do weekly Team staff meeting
Not needed in Scrum. The Team updates each other daily, and

managers can get an update on the Sprint in the Sprint Review Meeting.

HANDS-ON: SAMPLE JOB DESCRIPTION FOR A MANAGER IN A SCRUM-BASED ORGANIZATION

- Lead the recruitment and hiring of new Team-members (with the active involvement and input of the existing Team-members)
- Provide input to the Product Owner on the product strategy and vision, and give feedback to the Product Owner on the content and prioritization of the Product Backlog.
- Provide support and assistance to Teams and their ScrumMasters. Be prompt and proactive in helping remove impediments that are harming Teams' ability to be effective.
- Actively support ScrumMasters' efforts to protect Teams from disturbance, disruption, or outside interference.
- Be available to provide advice and assistance to Teams on technical difficulties that arise in the course of doing their work.
- Identify issues to Teams that they might overlook, such as scalability, performance, security, etc.
- Provide mentorship and career development advice and guidance to Team-members. This mentorship should include both technical mentorship, as well as soft-skills and other aspects of being effective and successful in a development organization.
- Plan and manage skills development and training for Team-members. Think carefully about areas where their skills need greatest development, or where the most opportunity for improvement exists; work with the person to identify appropriate training; and obtain budget and time allowance to complete it.
- Stay abreast of developments in the tools and technologies that Teams are using. Solicit input from Teams and other stakeholders on tools and technologies that could be useful. Spend time getting hands-on familiarity with these tools and technologies.

- Stay up to date on industry news. Be knowledgable about developments from our company, our competitors, and our largest customers, including financial performance, marketshare, product roadmap, and overall business strategy.
- Remove Team-members are not able to perform well within a given Team, work effectively with their fellow Team-members, or perform work at the level of expertise or quality required. This should come only after coaching and training has failed to correct the under-performance.
- Do financial planning and budgeting for Teams, including anticipating future people requirements, skills development and training needs, tools and technologies required, hardware, travel, and any other resources that people will require.
- Provide performance feedback and complete performance evaluations for Team-members. Informal performance feedback should be provided on a frequent basis, and should include feedback from fellow Team-members. Feedback should be focused on recognition for achievement, and opportunities for growth.

RECOMMENDED READING

The Enterprise and Scrum, Ken Schwaber (Microsoft Press, 2007)

Conscious Business: How to Build Value Through Values, Fred Kofman (Sounds True, 2006)

The Fifth Discipline: The Art & Practice of The Learning Organization, Peter M. Senge (Doubleday Business, 1994)