Circle of Trust and Responsibility

NEIL HARRISON, Utah Valley University

Development organizations are highly interdependent, which means that each person's success depends on the success of others. This can lead to feelings of insecurity, embodied by meddling in others' work and work processes, and micromanaging. Therefore, develop a virtuous cycle of trust and responsibility where a person in a management role trusts that others will accomplish their tasks without direct supervision. In turn, those doing the tasks show that the trust was warranted by accomplishing the tasks. It generally begins with the leader/manager granting trust to the producers.

Categories and Subject Descriptors: •Software and its engineering - Software creation and management - Collaboration in software development - Programming teams

General Terms: Patterns

Additional Key Words and Phrases: Organizational Patterns, Leadership, Responsibility

ACM Reference Format:

Harrison, N. 2023. Circle of Trust and Responsibility. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 30 (October 2023), 6 pages.

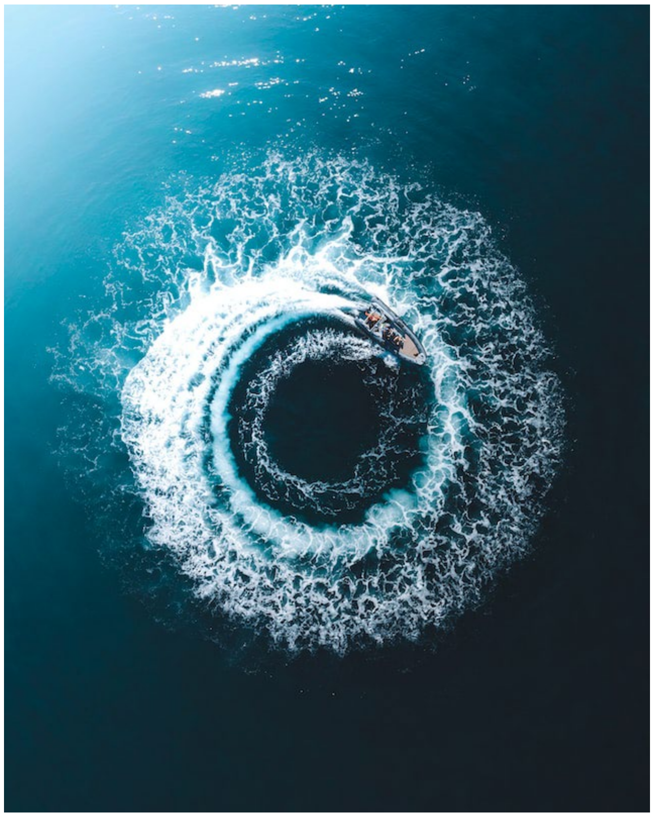
1. INTRODUCTION

Software development organizations are highly dependent. Each developer on a team has a responsibility to complete a portion of the software. Of course, they are not autonomous, if one person does not complete his or her portion, the entire system cannot work. Coplien and Harrison [1] and others have discussed that communication among team members is crucial to the success of a team. But not all team members are developers. Coplien and Harrison further explain that some roles in a team are support roles, not having direct contribution to the product. Rather, they contribute indirectly by providing support to the direct contributing roles (such as developers), and by supporting the management of the project. Managers are among those who in support roles.

The indirect nature of managers' contributions to the project, however, can lead to feelings of insecurity. This becomes embodied by meddling in others' work, often by micromanaging. In order to eliminate micromanaging, both managers and developers must work together to form a relationship of mutual trust. This paper describes a pattern, Community of Trust and Responsibility, which explains how managers and developers work together to form an effective relationship that supports both roles.

Author's address: Neil Harrison, 800 West University Parkway, Orem Utah 84663; email: nei.harrison@uvu.edu.

2. THE PATTERN



I once had a boss who was a micromanager. One time I was in the lab debugging my code. He came, stood behind me, and finally pushed me aside, saying, "Let me drive." He proceeded to debug my code for me.

… If an organization is built upon a foundation of trust, people feel safe in their interactions with the other members of the community. You can be open and honest without fear of reprisal. This is referred to as a Community of Trust [1]. This relationship is the essential foundation for any successful organization. However, it is incomplete. In production organizations such as software development teams, someone generally has responsibility to answer for the success of the entire team, while not doing (all) the work him- or herself. The natural pressure to complete work, and to ensure that work for which one has responsibility gets done can cause tension and even undermine the trust relationship.

The shift from command-and-control to self-managing teams, especially in organizations following the Scrum development process, has pushed much of the development responsibility from managers to the teams. This is reflected in the Scrum pattern, Autonomous Team, which states that the team charts its own direction without undue external influence. This creates the same situation – a person responsible for success has limited control over the success of the individuals who create that success.

❖ ❖ ❖

Development organizations are highly interdependent, which means that each person's success depends on the success of others. This can lead to feelings of insecurity, embodied by meddling in others' work and work processes, and micromanaging.

Whenever we are in a position where we depend on others, it is natural to worry that someone team members won't complete their work, causing everyone to fail. Even if the team is highly competent, this lack of control creates uneasiness. If the uneasiness is sufficiently large, even if it is unwarranted, people take defensive measures, such as frequently checking on progress, and giving advice on how to work. People may even try to take over work from others in order to get it done.

People in support roles (such as various types of management) are in a particularly difficult position. Unlike producer roles, these roles contribute their value solely by helping the teams or individuals they manage to succeed. A conscientious manager naturally wants to know – all the time – how well each member of the team is doing. Therefore, the natural tendency is to request status frequently. And managers may want to help, even to the point of jumping in and working. (This is particularly true if the manager was "promoted" from a development role.) But this sends two implicit messages to the team: first, the manager doesn't trust them to get their work done and second, that the manager doesn't believe that they are technically competent.

Roger Martin calls this problem "the Responsibility Virus" [2]. An over-responsible leader removes responsibility (for example, by doing the work themself). As a result, a passive follower doesn't have to do as much work, thus becoming "under responsible." This causes the leader to take on more responsibility, which in turn may cause the follower to take on even less responsibility. This creates a viscous circle where the leader cannot trust the follower to do any work, and the follower has fewer and fewer opportunities to actually do work.

Even well-meaning help can go too far. If you give someone too much help, they can gradually lose their ability to perform the work themselves.

Therefore:

Develop a virtuous cycle of trust and responsibility where a person in a management role trusts that others will accomplish their tasks without direct supervision. In turn, those doing the tasks show that the trust was warranted by accomplishing the tasks. It generally begins with the leader/manager granting trust to the producers.

The idea is that properly given trust and responsibility duly completed reinforce each other. If you are a manager, you trust that the team members are competent and will do their best without constant supervision. They in turn show that supervision is not needed by completing their tasks. This increases the trust level. So next time you are more willing to trust them, and they have more freedom to do their tasks, which increases their motivation, and even increases their ability to do their work.

To make this work, it is critical to establish regular status reporting intervals. The intervals should be agreed upon in advance and must be a team decision. The length of the interval should be short enough that the manager does not have to check to see how things are going more often than the interval length. For example, if the team decides to report status every Monday, the manager should only rarely check on status in the middle of the week.

If you are using Scrum [3] you have two tiers of regular reporting intervals. At the small extreme, the Scrum team meets daily in which status is generally given. And status is also formally reported at the conclusion of every Sprint. Note that regular intervals (such as Daily Scrums) are not set up to be micromanagement, but rather opportunities to share information and very small victories.

A complementary approach is to break the work down into pieces that can be completed within the reporting intervals. This is closely related to the pattern, "Small Items" [3]. Note that it is also crucial that the tasks are realistic: they cannot be so large that only a herculean effort will accomplish them; nor can they be so lightweight that people become bored, and the overall goals are not reached. It is the joint responsibility of both managers and developers to come up with best-faith estimates of work tasks.

While this is mainly a pattern for managers, producers must complete the circle. As a producer, you show that you can be trusted to complete your work without detailed supervision. By completing tasks, you also demonstrate technical competence by proving that you know how to do the assigned tasks.

Of course, there must be a balance between responsibility given and management of tasks. As a manager, you cannot completely leave individuals to their own devices, especially if people are technical novices or otherwise new to the organization. (The author's first technical job was like that, and I generally did not know what to do.) Early supervision/mentoring requires more detailed management. As a person matures, the level of management relaxes.

Newly formed teams and long-established teams will generally take somewhat different approaches to applying this pattern. New teams are just creating their culture, so this is the perfect time to establish the Circle of Responsibility. Managers and producers should agree they all want a self-managing team, and that the Circle of Trust and Responsibility makes that happen. Of course, they must first establish a Community of Trust [1] to form a safe, supportive environment.

On the other hand, mature teams that are trying to learn this pattern have significant challenges, because it means undoing dysfunctional habits. It's not just the manager: while the manager may have a habit of micromanaging, the producers may also have evolved habits of behavior that accommodate and perhaps even tacitly encourage micromanaging. Together, they may have formed a process that works – kind of – but nobody is happy with. But old habits die hard. The following two examples show how to break these habits.

3. EXAMPLES

3.1 The Broken Team

Suppose you are a manager new to an existing team. The previous manager was a notorious micromanager, and the team is jaded. How do you establish a healthy working environment that includes the Circle of Responsibility?

First, you must set clear expectations. Explain that you trust them in their work, and that you will avoid micromanaging. However, regular status is necessary. Propose a regular interval for status reporting and explain your rationale for it. Come to an agreement on the status reporting interval.

Explain that you assume that failure to meet goals is not because of laziness or incompetence, but by external forces (in Scrum parlance, "impediments"). Pledge to work to remove them.

In return for this trust, they have a responsibility to avoid surprises. If impediments arise, team members are responsible to tell you, the manager, as soon as they can. Don't leave it to the regular cycle of status reporting. In this way they show that they are worthy of the trust you place in them.

3.2 The Meddlesome Boss

In the second example, suppose that you are a developer or other worker, and your manager is micromanaging you to death. This is particularly difficult, because the tendency to micromanage is natural in support roles, as explained earlier. In addition, your manager may have had experiences where micromanaging appeared be successful. The manager is likely unaware of the detrimental effects on the team, though.

The key here is to anticipate and deflect the micromanaging. Report to your manager the success of some endeavor before your manager askes. For example, you might write an email saying, "I know that the whangdoodle feature is really important to the success of this next release, and I wanted to let you know that I just got it done." In this way you render the micromanaging redundant. It will probably take many repetitions for your manager to stop micromanaging. If your manager persists in micromanaging, then point out that you have a strong record of completing your work, and he or she should stop worrying and asking.

If you are a developer, much of the burden of maintaining the Circle of Responsibility falls on you. Report problems immediately; unless reports are done daily (e.g., Daily Scrum), don't wait for the next report. Where possible, come up with a remediation plan. Here is a rather extreme example: I once worked on a system where deep into implementation, we realized that the design approach would not work. We came up with the correct approach and then went to our boss and said, "Good news! After we redesign the system, it will work!" Fortunately, we had a circle of trust and responsibility with our boss, and although he wasn't very happy about it, he accepted our explanation.

This brings up the question of how to insulate the team against such problems. After all, occasional similar failures are all but inevitable in creative enterprises. One pattern which can help is "Completion Headroom", in which you build in some slack in the schedule to accommodate unforeseen problems.

4. IMPORTANT SPECIFIC APPLICATIONS IN SCRUM

Product Owner and Scrum Team: The Product Owner plays at least part of the role of manager with respect to work assignments. The Circle of Trust and Responsibility between the Product Owner and the Scrum team is essential for an effective team. There are numerous opportunities for the Product Owner to intrude on the Scrum Team, but the Product Owner must exercise great self-restraint.

Team Members and the Team in a Daily Scrum: The Scrum Master implicitly takes the role of the manager. Occasionally a team member or pair will not complete the day's work. The Scrum Master must not abuse the role by berating anyone or by using the Daily Scrum in any way to force anyone to work extra time to finish the day's work. This author has seen instances of Scrum Masters who have done exactly this.

5. CONSEQUENCES

❖ ❖ ❖

What does this pattern create? The result is a self-reinforcing circle of commitment and trust: people recognize that others are counting on them to accomplish what they said they would, so they feel a moral obligation to complete their tasks. This increases the likelihood of accomplishment, which in turn strengthens the trust in the team. Other team members may be motivated by example. If on the other hand, work occasionally does not get completed, you recognize that the person or persons did their best. It gives you an opportunity to examine what impediments prevented them from accomplishing it. And that in turn provides an opportunity to strengthen the entire organization.

Ideally executed, this pattern can contribute to a fun work environment, where everyone is able to be creative in attacking challenges and developing great things. Trust enables creativity. People are more likely to want to come to work and find joy in it.

This pattern creates the necessary foundation upon which self-managing teams can be established and can thrive.

6. CHALLENGES

This pattern requires significant maturity and commitment from everyone. If even a few people in the team are not on board, the Circle of Responsibility might not even get established.

Furthermore, it must be actively maintained. It is easy to slip and fall into old patterns of micromanagement especially if team members have such old habits. Status meetings (as opposed to collecting status via email may serve to reinforce the culture of the Circle of Responsibility. In addition, regular retrospectives may help to remind people and recommit them to their responsibilities.

Of course, things do not always go as planned. Development tasks sometimes take longer than expected, and this can threaten the Circle of Trust and Responsibility. Coplien and Harrison found that in stressful situations, managers in organizations tended to assume more central roles [1]. What should you do to keep the Circle of Trust and Responsibility intact? If you are a manager, remember that the developers were the technical experts before, and are still the technical experts. If the problems are external forces, you may be able to remove them. In Scrum, this is the impediment list. Planning in advance for unforeseen difficulties can help; make the development interval short enough to enable recovery from development difficulties. Also, consider using the pattern, Completion Headroom, as mentioned earlier.

What if you have a member of the team who just doesn't pull his or her weight? We hope that this is rare. But if it happens, offer trust until proven otherwise. It may fall to the rest of the team to make up difference. However, beware of the trap of getting into the habit of doing the slacker's work in order to complete the team's goals. The reckoning must happen eventually, and better sooner than later. Dealing with such problematic co-workers is beyond the scope of this pattern. However, public status meetings such as Daily Scrum meetings may serve to implicitly highlight such slackers and might encourage them to help complete their part of the responsibility circle. No guarantees.

7. RELATED PATTERNS

Work Flows Inward [1] describes how a self-managing team should receive work to do, and that the team is the focal point – not the manager. How is this different from Community of Trust? Community of Trust is about general relationship building and focuses on trust and safety in those relationships. Circle of Responsibility focuses on how tasks are accomplished and managed. Community of Trust is a necessary foundation for Circle of Responsibility. Without it, managers will not trust developers to do what they say, and developers will not feel safe to give honest reports.

Completion Headroom [1] explains that development episodes should have buffer time built in to accommodate unforeseen events.

Small Items [3]: Break work down into Small Items that are steps to deliver the big ideas, so the Development Team can master its understanding of each small item individually.

ACKNOWLEDGMENTS

Special thanks to my shepherd, Mary Lynn Manns, for her very insightful comments. She helped organize the pattern and sharpen its focus. Thanks also to the PLoP writer's workshop.

Image credit: Photo by Matheo JBT on Unsplash

REFERENCES

1. Coplien, J. and Harrison, N., "Organizational Patterns of Agile Software Development", Prentice-Hall, 2005.
2. Roger Martin, "The Responsibility Virus: How Control Freaks, Shrinking Violets – and The Rest of Us – can Harness The Power of True Partnership", Basic Books, December 2003.
3. J. Sutherland, et al., "A Scrum Book: The Spirit of the Game", Pragmatic Bookshelf, 2019.
4. W. Cunningham, "Episodes Pattern Language", Pattern Languages of Programming, vol 2, J. Vlisiddes, J. Coplien and N. Kerth, eds. Addison-Wesley, 1995, pp. 371-390.John G. Daugman. 1985. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two dimensional visual cortical filters. *J. Optical Soc. Amer. A: Optics, Image Science, Vision* 2, 7 (1985), 1160–1169.