Swarming Patterns

Joseph W. Yoder, The Refactory, Inc. – USA
Danijel Arsenovski, SolutinosIQ – USA
Ademar Aguiar, Faculdade de Engenharia, Universidade do Porto – Portugal
Hironori Washizaki, Waseda University – Japan

Many software development processes such as Agile and Lean focus on the delivery of working software that meets the needs of the end users. Many of these development processes help teams respond to unpredictability through incremental, iterative work cadences and through empirical feedback. There is a commitment to quickly deliver reliable working software that has the highest value to those using or benefiting from the software. A key principle to the long term success of a project is during the development and release cycles, to have confidence that changes will not break important parts of the system. Swarming is a technique where multiple people work closely together to complete one or more tasks, that help assure the delivery meets the requirements and proper validation and checks are done before release. This paper describes some swarming patterns and how they assist teams to deliver faster and with more confidence.

Categories and Subject Descriptors

• Software and its engineering ~ Agile software development • Social and professional topics
• *Software and its engineering ~*

General Terms

Agile, Sustainable Delivery, Patterns, Swarming, Mob-programming, Confidence

Additional Keywords and Phrases

Agile Software Development, Mob Programming, Innovation, Reliability

ACM Reference Format:

Yoder, J.W., Arsenovski, D., Aguiar A., Washizaki, H., “Swarming Patterns”, 2018. Procs. SugarLoafPLoP'18, November 20-23, Valparaiso, Chile. 11 pages.

Author's emails: joe@refactory.com, danijel.arsenovski@empoweragile.com, ademar.aguiar@fe.up.pt, washizaki@waseda.jp

Introduction

Being Agile, with its attention to extensive testing, frequent integration, and focusing on important product features, has proven invaluable to many software teams. However, blindly following Agile practices is not sufficient to help sustain delivering quality software at a good pace with confidence. There are many proven practices that are used during development to help sustain regular systematic deliveries. Some of these are related to Agile or Lean practices such as short delivery cycles, testing, clean code, and continuous integration. Small delivery size with regular feedback through incremental releases has proven itself over the years and has become the de-facto standard for most Agile practices.

Recently there has been observed success in collaboration techniques to assist teams working together and manage unexpected situations. For example, complex architectural decisions might need to be made which requires team participation.

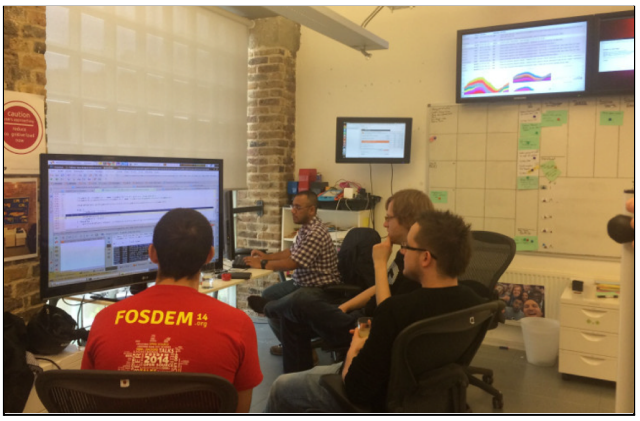
Swarming is one of these collaboration techniques, Swarming can be where only two team members work together on the same task or user story, similar to pair programming. The other end of the spectrum has all team members working on the same user story which is similar to "mob programming[1]". Mobbing is a form of swarming where the team works collectively to solve problems, releasing software regularly. In the later, all members focus on solving a particular problem by working on tasks together using a single computer.

Every programmer has experienced moments of mental exhaustion where it can take a lot of time to resolve even the most trivial problem. Most programmers (even those that do not practice pair or mob programming) are aware of this and will find ways to take breaks and invite fellow programmers to take a look at their code in those moments of frustration. In mob programming sessions, these moments are practically non-existent. The patterns presented in this paper are used during swarming to address common issues: *Branch Out*, *Spread Out*, and *Keep One in the Dark*. These patterns support swarming and provide the benefit for the team to react and reorganize allowing other team members to continue working on current items in "business as usual" fashion, while a few members would deal with impediments and urgent or unplanned issues. This means that the normal team dynamic remains unaffected, learning continues and all members are still aware of the current work in progress. An early experience report on these patterns were presented at Agile 2016 [Arsenovski].

[1] "Mob Programming – All the brilliant people working on the same thing ...." 8 Jan. 2018, http://mobprogramming.org/. Accessed 7 Oct. 2018.

Swarming

"The fiercest serpent may be overcome by a Swarm of ants." — Isoroku Yamamoto



Teams are working on completing their tasks. Team members are usually working individually on finishing these tasks.

How can individual team members address and resolve unforeseen issues and difficult impediments in an effective and timely way?

❖ ❖ ❖

Team members may be faced with unforeseen issues and impediments during the project that might be hard to address by themselves individually..

There is various expertise among individuals of the team, however it can be difficult to take advantage and benefit from these experiences.

Sharing knowledge across the team about a solution can be challenging and take considerable time. Also, as team members come and go, some of this knowledge can be lost.

Some team member(s) could come up with a solution to the problem but there might not be general agreement on the solution. For example, a solution might solve the problem but it could be so complex so that only one team member can understand and maintain it.

❖ ❖ ❖

Therefore, have multiple people work together (swarm) on the work item, be it an issue, user story, or impediment. The work can be done by a few team members or it can be the whole team working together collaboratively to finish the task.

Swarming is a collective behaviour that results from groups working together to solve a specific task or move in a specific direction. When dealing with more far-reaching issues like design and architectural questions, *swarming* provokes a creative dialogue and exchange of ideas without falling into a trap of long and heated arguments. After a while, the team starts working at the same "wavelength" and collective decisions are based on mutual trust. In cases when significant difference in opinion persists longer than usual, the issue can be settled by conducting experiments.

Swarming requires less involvement and less of an intimate relationship between participants than pair programming. A larger group fosters more open discussion and it is much more difficult to impose opinions based on authority and individual personalities. The tacit

knowledge and collective intelligence is facilitated by swarming. There is also a more varied mix of expertise and experience as members are free to join and leave the swarm as needed.

These points generally lead to swarming to be less strenuous on the participants. Practical application of swarming has proved that a group environment (as opposed to a pair) overcomes many objections voiced against pair programming [Sargent]. A recent study [Laughlin] suggests that groups of three or more are more effective at problem solving than a single person or a pair.

Swarming has been around for quite some time and good agile teams will do what it takes which includes pulling together to do whatever is needed to solve problems and remove impediments. Teams often swarm when there is a serious problem such as system crashes or dealing with customer problems. Great teams will use swarming as a planned activity. They recognize that difficult issues will be encountered during development and it is important for everyone to work together and assist each other with resolving problems when they arise.

* * *

The following are some advantages of *swarming*:

- Broader team understanding as the whole team is sharing knowledge about the system;
- Code is better reviewed as the team is constantly reviewing the code;
- "Bus factor[2]" equals the team member count;
- Continuous learning through sharing of ideas from everyone;
- Efficient problem solving by everyone focusing on single most important task;
- Feeling of fun while working - playful work;
- Less interruptions from email, chat tools or even from other people in person;
- Transparent form of work since all team member know what everybody is doing.

There are also some potential disadvantages to *swarming*:

- Group's successful will be related to the way its members interacted;
- Groupthink can become part of the swarm, thus not bringing in new ideas;
- Can be inefficient for solving many problems as all people focused on one problem;
- Can have communication overhead as many people interacting;
- Decision making can be slow as swarming looks for consensus.

As *swarming* has all people working together, there can be challenges where some tasks are too mundane or wasteful to have all participants working together to solve the task Also, there might be times where certain impediments are blocking the task which might require only 1-2 people a short amount of time to complete. In these situations, *Branching Out* allows for these tasks to be more efficiently addressed by a *swarming* team. Additionally there can be a task that requires repetitive work throughout the code or system such as fixing a bug. To solve this the group can figure out how to address the task together and then split up and *Spread Out* to solve the task in parallel. Finally to help bring new ideas into the team and to address the issues with groupthink, it is good to occasionally *Keep One in the Dark* by having a member leave the team for a while or switch with another team and then come back with fresh ideas and can also look at and challenge the group's thinking.

[2] **Bus factor**, or truck or lottery factor is the number of team members that have to be run over by a bus (or win a lottery) in order for some knowledge to be lost (https://en.wikipedia.org/wiki/Bus_factor)

Branch Out

"If you don't accept failure as a possibility, you don't set high goals, you don't branch out, you don't try - you don't take the risk." — Rosalynn Carter

The team is *swarming* while adding features. During the development, an impediment appears, preventing the team from collectively bringing items to a close.

How can a team deal with emerging impediments efficiently while swarming?

❖ ❖ ❖

Keeping the team together with everyone working as a whole is a core principle of swarming or mobbing but implies handling with a single work item at a time.

Some tasks can seem not demanding and look like a waste of time for the whole team to focus on each one at once.

Some tasks can be very complex where a spike solution might be needed.

Some tasks are so important and urgent that they need to be addresses as soon as possible.

Sharing knowledge across the team about various implementations is necessary.

❖ ❖ ❖

Therefore, branch out by having a team member or subset of the team break out to work on the emerging issue.

There are times when there are issues or impediments where *swarming* might not work best. This is a time to turn on the "parallel" mode and have a member, pair or a group break out from the team and work on removing the impediment. After the impediment is removed, those who *branched out* return to the main mob.

An important difference in the "parallel" mode approach compared to *swarming* or mob programming approach is that while the team as a whole is still focused on bringing single work items to a close, there still might some tasks that it is better to focus on individually, in pairs or in small group such as to remove different impediments that stand in the way of completing the current work item.

Branching out makes sense if you have something very simple and urgent. In this case it can be a better use of time to have an individual, or pair, go off and get rid of these simple tasks, allowing the team to stay focused. *Branching out* also can also work well when you have a very complex task that might require a spike. It could be better use of the team's time to have an individual or pair focus on the spike and then rejoin the team with options learned from the spike. After completing the task(s), a code review could be performed where the solution was presented by the team member that *Branched Out*.

Branching out can also be warranted when some routine, low-complexity work needs to be performed. Such work presents only a small opportunity for learning. Performing it in unison does not provide interesting benefits when all team members are already familiarized with the procedure and automation is not warranted. In these situations a member "*branches out*" to perform the work and joins the rest of the team once this "solitary" work is done.

A similar approach would take place when some urgent issue is reported: one or two members would *branch out* to investigate and, if possible, resolve the issue. In the meantime, the rest of the swarm would carry on with original work.

One situation when the swarm should avoid the urge to *branch out* is when an automated solution is viable, and can provide a long term win for the team. In this case, it is better to continue swarming and work on automation of the current problem, than to *branch out* to find a quick fix.

* * *

The following are some advantages of *Branching Out*:

- Freeing teams up from blindly doing tedious tasks can help keep the swarm focused and more productive;
- Some important critical tasks can be done more quickly.

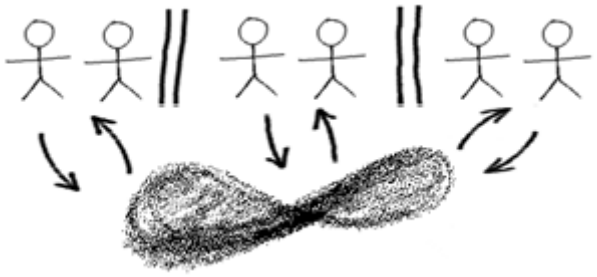
There are also some potential disadvantages to *Branching Out*:

- Knowledge sharing can be lost if not brought back to the team;
- Having people leave and rejoin the team can cause some issues with momentum as it takes time to reorganize the team dynamics.

Branching Out frees up the *swarming* team to stay focused and moving forward, keeping the heartbeat of the team developing. There are cases where the *Branch Out* items might include a lot of tedious repetitive tasks, in which case the team might consider to *Spread Out*. Also, *Branching Out* can be similar and achieve the same results as *Keeping One in the Dark*.

Spread Out

"Spread love everywhere you go. Let no one ever come to you without leaving happier."
— Mother Teresa



We are *swarming,* working on tasks and we found a defect or change that needs to be uniformly applied across the code base.

How can we apply a crosscutting change across a large code base efficiently and effectively?

❖ ❖ ❖

Some tasks require a lot of work across the code base possibly requiring many team members for a considerable amount of time.

To use all team's knowledge is useful to effectively and efficiently design and implement the change to fix an issue or defect.

There can be some repetition to apply some change or to fix something. It can be inefficient to have the whole team repeat these tasks across the system.

❖ ❖ ❖

Therefore, after resolving the first occurrence as a group, have the team "spread out" and individually apply the solution to the rest of the system.

The team can either do this individually or might break up into pairs to apply the solution. After the work is done, the team would rejoin to comment on different facets of the defect and how these were resolved, as well as to document the experience. They can also verify that the changes were applied correctly. *Spread Out* is similar to *Branch Out*, however, spread out implies putting parallelism to maximum use. *Branching out* implies a small group dedicated to resolving tangent issues. You would only *Spread Out* if there are a lot of tasks that can be done in parallel.

Once again the swarm should avoid the urge to *Spread Out* when an automated solution is viable, In this case, it is better to continue swarming and work on automation of a current problem rather than splitting the team up. It might make sense to have a pair *Branch Out* to experiment with a spike solution to see if automation is a viable alternative. In this case, if there is a viable solution to automation, these ideas should be brought back into the swarm for final integration and validation.

* * *

The following are some advantages of *Spreading Out*:

- Splitting the team up addressing these repetitive tedious tasks can help keep the team efficient, thus more productive;
- Tasks that are repetitive through a large part of the system can be done more quickly.

There are also some potential disadvantages to *Spreading Out*:

- When the team *spreads out* and needs to rejoin, there can be overhead on getting the team back together and refocused on *swarming* tasks. In this case the swarm is completely lost and has to reform;
- People may become stuck or prefer separating, thus losing the swarm mentality.

Spreading Out works well when there are many repetitive tasks the team needs to complete. If this is not the case, then *Branch Out* is a better way to achieve this, thus not breaking up the team and losing the benefits of *Swarming*.

Keep One in the Dark

"Walking with a friend in the dark is better than walking alone in the light."

— Helen Keller

We are *swarming,* evolving the system applying the best practices. However the teams seem to have lost the creativity and everyone seems to have the same ideas, or simply reduced motivation to do better.

How can we continue to evolve, grow and learn as a team?

❖ ❖ ❖

Uniformity can stifle creativity and lead to biased decisions, motivated by conformity, a phenomenon known as groupthink [Janis].

Diversity in a team's knowledge and skills helps to generate options for solving problems but may lead to long debates and decision-making processes.

Having team members come and go in a team creates challenges to the team dynamics.

❖ ❖ ❖

Therefore, occasionally have one person abandon the swarm until a certain task or a set of tasks is finished. This can include having another person join the swarm from outside the team to bring new ideas into the team.

After completing the task(s), a code review would be performed where the solution was presented to the team member temporarily excluded from the swarm and where they are able to question the decisions taken. Their suggestions could then be incorporated into the solution if warranted.

One way to do this when there are multiple teams is to swap team members on a regular basis. This can help bring new ideas and to reduce groupthink. Another way is to have the team member *Branch Out* or just simply do other things independent of the Swarm team.

When a team gets stuck, it is important to remember that if nothing changes, then nothing changes. Therefore it is good to experiment and try different things. *Keeping one in the dark* can help move the team out of their comfort zone, possibly opening up for new ideas.

* * *

The following are some advantages of *Keeping One in the Dark*:

- The group is kept fresh with new ideas and motivated;
- When a person steps out of the group,, they get an opportunity to reflect and get a better perspective of how the group is running and the work the group is doing.

There are also some potential disadvantages to *Keeping One in the Dark*:

- When a member is *Kept in the Dark* and rejoins, there can be overhead on getting the team back together and refocused on *swarming* tasks;
- Team members might resist the new ideas from the team member that separated.

Keeping One in the Dark can be done by *Branching Out*.

Summary

Swarming is the act of two or more team members coming together to work on tasks together. Mobbing is a form of swarming where the whole team works together fulltime in a fast succession on a shared workstation. Various swarming techniques have been successful in various environments and can be a catalyst for a profound change in team structure and organization. Swarming usually requires intense collaboration, coupled with mutual respect and autonomy harnessed collective intelligence. When properly done, teams have had results of continuous learning and reach new levels of productivity [Zull].

This paper presents patterns that address issues that arise when swarming. For example, how does one split and merge when appropriate (*Branch Out* and *Spread Out*). Additionally by *Keeping One in the Dark*, a team can counteract groupthink and continue to grow and challenge itself.

Acknowledgements

We'd like to thank our shepherd Yann-Gaël Guéhéneuc for his valuable comments and feedback during the SugarLoaf PLoP 2018 shepherding process. Finally we'd like to thank our 2018 SugarLoaf PLoP Writers Workshop Group, Audreas Seitz, Eduardo Guerra, Monica Michelle Villegas, Hernán Astudillo, Joelma Choma, and Paulina Silva for their valuable comments and suggestions.

References

[Arsenovski] Arsenovski, D., *Swarm: Beyond pair, beyond Scrum*, 2016. Agile Alliance Experience Reports. agilealliance.org/resources/experience-reports/swarm-beyond-pair-beyond-scrum/

[Janis] Janis, I. L., Victims of Groupthink, 1972. New York: Houghton Mifflin.

[Laughlin] Laughlin, P., Hatch, E., Silver J., Boh L.., *Groups perform better than the best individuals on letters-to-numbers problems: effects of group size.* Journal of personality and social psychology, Vol. 90, No. 4. (April 2006), pp. 644-651.

[Sargent] Sargent, W., *Where Pair Programming fails for me.* https://content.pivotal.io/blog/pairing-isnt-for-everyone.

[Zull] Zull, W., *Mob Programming – A Whole Team Approach*, 2014. Agile Alliance Experience Reports. agilealliance.org/resources/experience-reports/mob-programming-agile2014