Swarm: Beyond pair, beyond Scrum

DANIJEL ARSENOVSKI, SolutionsIQ

The report presents an overview of a nearly year-long experience in 'swarming' while addressing problems of a defective legacy application. Given that the application suffered from tremendous performance and functional problems, the work on its enhancement provided an excellent opportunity for testing the benefits of swarming. Swarming provides for collective intelligence and problem solving, real-time collaboration, fluid leadership with no explicitly defined roles. Because of this, swarming proved to be an excellent organizational pattern for resolving complex problems in a complex and tense environment where positive results had to be demonstrated almost immediately.

1. INTRODUCTION

During 2014 I lead a small team of tightly-knit developers who largely worked on a single screen and keyboard. Our team used an approach which included no formally defined roles, performed no estimates, and seldom worked on more than one task at a time. Later we called this swarming which proved very useful to our team including delivering a quality product to a satisfied customer.

Although I had previously used 'mob programming' (Zuill) in different settings, this was the first time I had applied the swarming approach to an entire project. What initially started as a typical agile project, soon evolved into a largely different approach to organization and management, primarily due to the addition of the mob programming "swarming" dynamic. This report will describe what we learned from the swarming experience.

2. BACKGROUND

I have been teaching developers in agile development practices for number of years. Originally I used a traditional workshop format where attendees performed exercises individually. Then I started introducing pair programming and finally mob programing.

"Mob programming" involves a group of programmers and other team members like product owners or testers discussing solutions at the same table and writing code in fast succession, on a shared screen and keyboard. I previously tested this format on a real project for short periods of time. The results were more than encouraging and this time I decided to execute the whole project in this manner.

3. THE STORY OF SWARMING

3.1 3.1. Problems

After performing some coaching for a support team and initial diagnostics of a recently put into production ERP solution, I was approached by a client to help them bring the system back into a working condition. The system would often stall and needed a reboot and had a number of known bugs - the features were mostly copied from a legacy system with no user or business expert input, etc. The initial diagnostics discovered a huge number of runtime exceptions, memory leakage, abysmal performance and other issues. In the meantime, the company that originally developed the system went bankrupt - not least because the client lost patience and decided to part ways.

Author's address: Danijel Arsenovski, 16200 SE Eastgate way, Bellevue, WA; email: danijel.arsenovski@empoweragile.com

One of our alternatives was to abandon the system and go with some of the leading ERP solution providers. Aside from the political and financial cost, this would result in another long period of customization and rollout; instead they decided to give the existing solution one more try.

Fixing a huge legacy application with tremendous problems and source code suffering from every known code smell is nothing short of a "hot potato". Still, we decided to give it a go, as long as the client was ready to do it on our terms. These weren't many - basically, at all times we were going to work on a single task. We'd do a test run for one month and then, if the client was happy with the results, we would renew the agreement for a longer period of time. To different methodologies proponents, probably more interesting than the experience itself were the things we decided we did not need to be doing.

3.2. What we did

A lot of requests we received had to do with fixing various defects such as: a user unable to submit a form, a report displaying different data for the same input parameters or taking too long to run, etc. During development, we used swarming and continuous integration. We would add some automated tests and would refactor and clean up the affected code. In most cases, we were able to deliver a solution for the particular issue within the same day.

Very often, after a bit of investigation, one bug would unearth a pervasive anti-pattern. For example, the application relied on a home-baked rudimentary object to relation mapping technique that would pull down a whole table from the database server and performed querying on the application server. Almost every web page contained some database querying code. Anywhere a database query was issued had to be replaced with more efficient alternative that would perform querying on the database server. This would mean a few additional days spent applying the solution globally and, on more than one occasion, a few weeks dedicated to resolving memory leaks and optimizing performance. But, for the most part, we were able to deliver a solution to the pressing problems in a matter of hours. We have also put into practice a manual, but efficient, solution for new version rollouts with the capability of rollback.

For those requests that implied developing a completely new feature, we would choose a small slice of relatively self-contained functionality and deliver it in a day or two. In this way, our client was able to make the projection himself: if this piece took two days, then the whole feature would probably take two or three weeks. This way no time was wasted on estimates. The key was delivering software in small consistently working increments.

3.3. The results

The team excelled in providing customers with solutions to pressing problems in very short cycles and excelled in the quality of the solutions. Not only were problems resolved, but were done so in a very thorough manner. Features were improved and the root cause was generally removed from the rest of the code base.

Because we delivered new versions that contained bug fixes or new features daily we were able to respond to bug requests almost immediately. Users felt like they were listened to and that their needs were taken into account. This meant that trust between the IT department and the rest of organization was regained. High-level management was also happy. While they were not really aware of the approach we used, they realized that the current system will be put into usable state in the foreseeable future. This meant they will not have to pay for a new "off the shelf" product (building new custom system after current debacle was definitely discarded). It also meant that the organization will not have to be put through the pain of a lengthy implementation and customization of a newly acquired system.

4. WHAT WE LEARNED

4.1 Collective Intelligence

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 or pair programming) are aware of this and will invite a fellow programmer to take a look at their code in those moments of frustration. In a mob programming session, such moments are practically non-existent.

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.

4.2 Three's a company

A subtle psychological effect of swarming compared to pair programming is a less involved and less intimate relationship between participants. A larger group fosters a more open discussion. Moreover, it is much more difficult to impose opinions based on authority. There is also a more varied mix of expertise and experience. Because members are free to join and leave the main mob at will, swarming tends to be less strenuous. Practical application of swarming has proved that a group environment (as opposed to a pair) overcomes many objections voiced against pair programming (Sargent). Finally, a recent study (Laughlin) suggests that groups of 3 or more are more effective at problem solving than a single person or a pair.

4.3 Learning and mentoring

In the knowledge industry, learning and mentoring should be an integral part of everyday work activity. A swarm provides an ideal environment for that, as long it is done in a deliberate manner and at a steady pace. By observing others, team members learn almost immediately how to make better use of tools and perform their day-to-day work. More importantly, they will soon gain deeper understanding of a problem domain and will improve their problem solving skills as they observe and question others on proposed solutions.

4.4 Swarm Patterns

While most of our time was spent mob-programming as a whole, on occasion, we would deliberately turn the "parallel" mode on. The most important difference in this approach compared to traditional or even pair or mob programming team setup is the fact that the team as a whole is always focused on a bringing a single work item to a close. However, they still might focus individually, in pairs or in smaller groups to remove different impediments that stand in the way of completing the current work item.

Probably the most important benefit of the ability to react and reorganize was the capacity of the main group 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 very much aware of the current work in progress. Here are several patterns I was able to identify.

Branch out

Some kind of impediment appears, preventing a team from bringing certain item to a close. A member, pair or a group will 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.

Another situation when branching out is warranted is 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 those 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 was 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.

Figure 1. **Branch Out**

Spread out

From time to time a defect was discovered that was uniformly applied across the code base effectively forming an anti-pattern. In such a situation, after resolving the first occurrence as a group, the team would "spread out" to apply the solution to the rest of the defect instances. After the work is done, the team would join to comment

on different facets of the defect and how these were resolved, as well as to document the experience. Spread out is similar to branch out, however, spread out implies putting parallelism to maximum use. Branching implies a small group dedicated to resolving a tangent issue.

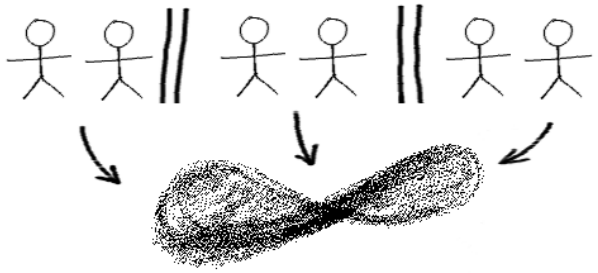

Figure 2. **Spread Out**

Keep one in the dark

Some people argue that groups can stifle creativity and lead to irrational decisions, motivated by conformity, a phenomenon known as groupthink (Janis). In order to avoid this happening to our team, from time to time one person would purposefully abandon the swarm until a certain task was finished. After completing the task, a code review would be performed where the solution was presented to the team member temporarily excluded from the swarm and where he was able to question the decisions taken. His suggestions would then be incorporated into the solution if warranted.

Figure 3. **Keep one in the dark**

4.5 No Roles

No roles were defined explicitly during the engagement nor was there any form of team hierarchy. As the engagement progressed, the team members interchangeably started performing different kinds of tasks, including negotiating commercial aspects of the engagement with the client, working on marketing, procuring new opportunities etc.

4.6 Fluid Leadership

In most cases, teams are formed with a pre-established leadership and other roles. Team roles are often allocated based on an employee's position in a company, which frequently produces an uneasy relationship between the two. It is not realistic to expect the same person to show superior skills and initiative in each and every situation.

In our swarm, a different team member would show initiative depending on a particular situation. So, if a particular technology was a specific team member's area of expertise, he would lead the way, while the rest of the team was getting familiar with the technology. If a team member had a closer, pre-established relationship with a client, he would lead the conversation and make sure to involve the others. In time, all team members become capable of autonomously performing a great variety of tasks and are at ease with taking important decisions.

4.7 No (Formal) Decision-Making

Decision-making is one of the most difficult and challenging tasks any team has to perform. Before swarming, my experience was more or less limited to 'command and control', or 'traditional democratic' teams. However, 'command and control' teams can stifle a member's autonomy and initiative. Those teams often belong to hierarchical organizations where internal politics and bureaucracy can have a significant impact on team performance. On the other hand, 'traditional democratic' decision-making can often result in a lengthy process, protracted meetings and heated discussions, where a majority based decision-making, often delivered as a voting exercise, still leaves a number of members dissatisfied.

One of the unplanned effects of the swarm was the way the decision process would play out. Decision-making became a part of everyday work flow, not different from most "grunt work". A team is already aligned, shares the same vision and most agreement is generated collaboratively and almost instantaneously - for example, while writing a certain document or email in a "mobbing" fashion.

4.8 No Ceremonies

For a collocated team that is always informed of the current work in progress, many of typical Scrum ceremonies, like daily standups or sprint planning and review, make little sense. They might be appropriate for novice teams, but more mature teams are more likely to feel restrained by them. We would speak to a client often and whenever necessary. We preferred direct and live communication to email. In these meetings we would present the finished product to the client and we would agree on the next work item to pull. As a general rule, the whole team would participate in all meeting with the client and the stakeholders.

We would discuss and often implement different and better ways of doing things as a part of a normal daily routine. With other teams, retrospectives were taking place at the end of sprint and we would hope that some of the action items would make it into the next sprint. With swarm, we strived to take any immediate opportunity for improvement.

4.9 Playful work

In a healthy and respectful team environment, swarming becomes a playful and enjoyable social experience. While during intermissions there is a lot of humor and laid-back conversation, it is the work itself that becomes an engaging activity and almost never drudgery. As a result, there is much less stress - problems are dealt with in a collective manner and at no point is a single person left on their own to deal with a difficult problem.

Most of us writing code enjoy the mental challenge it entails and derive satisfaction from creating something used by other people. Swarming turns coding into an enjoyable and playful social event, surprisingly revealing that programmers are in fact "social animals" and can greatly enjoy collective problem solving.

4.10 Different kind of flow

Programming demands a lot of mental effort, while attaining the best results requires intense concentration that is best achieved in a quiet environment with no interruptions. In literature, this state of high immersion is often referred to as "the flow". At first glance, it may seem that a swarm-type social environment might prevent reaching this highly productive state.

However, it turns out that "swarming" members are also highly immersed, although probably in a slightly different kind of flow. One member will easily take over after a switch in a mob programming session, since generally everybody is highly focused on the work being performed.

This "social flow" is most visible during the most challenging problem-solving periods. Often, the work will start with one member leading along a certain path, another helping resolve impediments that might crop up, the third suggesting a slightly different direction or a detour and so on, until the problem is solved.

5. SUMMARY

A rather simple change in a working routine, namely whole team working in a fast succession on a shared workstation, proved to be a catalyst for a profound change in team structure and organization. Intense collaboration, coupled with mutual respect and autonomy harnessed collective intelligence, enabled continuous learning and helped team reach new levels of productivity. Team spontaneously dispensed of superfluous and started exhibiting new patterns of behavior: splitting and merging when appropriate, taking initiative, committing to ad-hoc leadership and roles. All of this resulted in gratifying and motivating work

experience. Socialization, collaboration and altruism are hallmarks of humans as a species. When work environment is enabling these, a team will flourish.

6. ACKNOWLEDGEMENTS

I would like to acknowledge my fellow swarm members: Dragan Mijatović, Vukan Đurović, and Igor Orlić. They were happy to indulge my unorthodox ideas on team organization. I would also like to thank Dejan Arsenovski, who co-wrote this article with me. Finally, special thanks to Joseph W. Yoder who provided detailed feedback and helped me express my ideas much more clearly.

REFERENCES

Sargent, Will, "Where Pair Programming fails for me", https://tersesystems.com/2010/12/29/where-pair-programming-fails-for-me/
Patrick R. Laughlin, Erin C. Hatch, Jonathan S. Silver, Lee Boh "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
Zuill, Woody, Mob Programming – A Whole Team Approach
Janis, Irving L. (1972). Victims of Groupthink. New York: Houghton Mifflin.