Colocation: A Case Study in the Rewards and Perils

Mary Panza

Parametric Portfolio Associates

mpanza@paraport.com

Abstract

Some Agile software teams struggle to colocate their members so the software developers and testers can be near each other at work. The Parametric development team members had the luxury of sitting together in the IT department from the beginning, but sought even greater flexibility and efficiencies. We took the bold step of colocating some development teams with the end-users of the software they write. We had lofty goals for the experiment and some specific expectations, but ended up with some unexpected results, as well.

This paper serves as a case study and a retrospective on our effort. It discusses our reasons for moving the teams, as well as the lessons we learned and the changes made to our process. The colocation experiment caused our development organization re-examine its best practices and processes. We would like to share our findings with other teams and organizations.

Biography

Mary Panza *is the scrum master at Parametric Portfolio Associates in Seattle, Washington. She has supported, tested and managed software for the past twenty years for various companies around the Puget Sound area. Mary's passion has always been for problem-solving and process improvement, as well as seeing the human side of software development.*

The views and opinions expressed in this article are those of the author and do not necessarily reflect the policies or positions of Parametric Portfolio Associates LLC.

1. Introduction

Parametric Portfolio Associates LLC (“Parametric”) is a registered investment adviser focused on the delivery of engineered portfolio solutions. Technology is leveraged to deliver a scientific and pragmatic approach to a rules-based, but dynamic investment approach. Parametric maintains a small development team of approximately twenty people that write, test and support custom software for in-house business users. This environment affords flexibility not commonly available to development teams in most standard software companies. The development team’s mission is to build proprietary software to carry out Parametric’s business strategy with the most efficiency and least risk possible. How the team meets these goals is unrestricted by company management. Several years ago, we, the development team, realized the method we used to develop software – in small increments with early user input and immediate feedback – already had an industry name: Agile. Rather than attempt to “reinvent the wheel”, we set out to learn more about Agile Software Development and began to apply Agile tenets to our own process within the team.

The Agile Manifesto speaks about “individuals and interactions”, and the principals behind it weigh open and easy communication heavily as a factor for success (Cunningham, 2001). Collaboration through face-to-face conversation is an ideal goal for sharing information among members of the software development team (the “delivery team”). It is also just as important for interactions between the delivery team and the actual end users of the software. Often the end users are represented by a single individual seen as the key stakeholder on a project (the “product owner”) to simplify the communication routes, but the goal remains the same: the people creating the software and the people using the software should talk openly and freely about the software. Colocation is seen by many Agile practitioners as the best way to achieve this collaboration. At a basic level, the members of the delivery team should sit close together to reduce the physical impediments of sharing information among the members and to enhance the willingness to share by building the team’s sense of community. (Cockburn, 2001) Having the product owner and delivery team physically located together, or at least in close proximity, increases the collaboration further. (Krumins-Beens, 2012)

At Parametric, the product owners are members of functional business units and have regular responsibilities to perform for their departments; they act as product owners in addition to their standard duties. This arrangement inhibited moving the product owners away from their business team to sit with the delivery team assigned to their business unit. Instead, we decided to move some delivery teams closer to their respective product owners. This paper serves as a retrospective on the effort to colocate some of the delivery teams with their product owners and business users. We had a few specific goals and expectations going into the change, and we received a few surprises along the way. It is our hope that others can benefit from our experience.

1.1. General description and overview of our original process

The overall development team is comprised of smaller teams assigned to work with specific business units; for simplicity’s sake, these delivery teams can be referred to as Team A, Team B, Team C and Team D. Originally, the entire development team sat together in the same physical space in an open building design. Application Support, DevOps, and Development Management were also located in this large group. Each delivery team had its own pod of desks to facilitate an easy flow of communication all day long. The desks and computer set up allowed for two or three members to work together easily at the same time, an Agile practice called “pairing”. White board space was abundant, and each team had a physical task board and calendar that were visible to all. All the teams were using some form of Scrum or Kanban to manage their work and had a quick daily meeting that included the product owner, whose desk was located elsewhere in the building with their functional business unit. The individual teams met regularly with their product owners to review completed work and plan upcoming projects. The frequency and formality of such meetings was determined by working agreements between the delivery team and the product owner.

1.2. Initial Perceptions: Strengths and Weaknesses

The working proximity of the delivery team members enabled them to eavesdrop on each other. If a software developer needed a code review, it was easy to find someone. If a tester couldn't figure out why tests were failing or the build was breaking, help was close by. Questions about why a feature was implemented in a particular manner could simply be asked out loud to the general group; chances were someone could explain in detail. When a question came up about a business area, it was easy to find someone who could answer the query or suggest an appropriate resource. The open atmosphere created a collective consciousness around the internally produced software, which benefitted not only the delivery teams, but the support teams, as well. Together, the overall development team had a shared understanding of the domain knowledge and technical skills needed to complete projects.

This environment also fostered strong team-building opportunities. A delivery team could easily solicit and receive a design review from another member of the development department who had directly related experience; for example a developer who had written code for that application in the past or a support technician familiar with a targeted bug. Information and experience flowed between delivery teams seamlessly. The team members shared ideas through informal conversations, collaborated over coffee and worked out design issues during lunch.

The biggest perceived weakness in the original environment was the physical separation of delivery teams from the product owners and business users. In order to write software that serves specialized business needs, the delivery team must understand how the end users accomplish their tasks and what problems they face. In the initial arrangement, the delivery teams had limited daily interactions with the product owner via the daily meeting, as well as time-boxed interactions regularly through project planning. Additionally, some team members would infrequently shadow a business user to watch them work, but this did not provide enough information. This left the delivery teams with a potentially limited understanding of how their users really completed their work. When writing highly customized software, this arrangement proved to be problematic. In the year prior to the colocation exercise, 60% of hotfix releases could be tied to a lack of understanding of the business process. A misunderstood requirement or an incomplete test case resulted in the incorrect implementation or a bug. Almost one third of all feature release required a follow up release. And these numbers do not show the cost of time spent in extended rounds of user-acceptance testing where these problems were caught prior to release.

2. Colocation changes

Since the Product Owners have positions in their business units as functional team members, i.e., their "regular job", they didn't have abundant time to train, demonstrate or explain the daily processes and procedures. Realistically, the delivery teams required this critical business information in order to produce quality software. The developers and testers needed a way to gain this knowledge and spend less effort doing so. We hypothesized that since information and experience flowed easily between development team members simply because they sat in close proximity to each other, perhaps the same thing would happen if we located a delivery team to sit with their product owner and business users. The arrangement we hoped to create, where information could be exchanged without much interruption to either the delivery team or the business users, was termed "Osmotic Communication" by Alistair Cockburn (Cockburn A. , 2004). He described an environment where the cost of communication was low, yet provided team members with a rich information source. If a delivery team could glean information about business processes and problems simply by hearing about it in the background, then moving them to sit near their respective business unit could bridge the knowledge gap.

Over the past year, we moved three different delivery teams to sit with their respective product owner and business users. The following section covers our decision to move the teams and what we expected would happen.

2.1. Why – The Goals

Our goals were straightforward:

1. Deepen each delivery team's understanding of the business processes and problems typically found in the area for which they were writing software.

2. Improve the relationship between each delivery team and their respective product owner and business users.

3. Leverage that relationship to inspire new creative solutions to the current and future business needs.

We theorized the delivery teams would be more efficient in generating quality technology solutions if they better understood what the end-users needed to accomplish in their daily tasks. We wanted to improve our own productivity and reduce our own risk as we built software that would help our business to do the same.

2.2. What we wanted to happen

When we decided to move the first team – Team B, we thought they would gain enough knowledge of the daily tasks and processes to perform those same tasks later in their own development and test environments. Thanks to the cooperative benefit we expected, developers would ask questions, watch how the software was used, and understand how the business users really work. In turn, the product owner and the rest of the business team members would get a better idea how their software was developed. Ideally, together they could bounce ideas off each other to improve the quality of the product, determine the best approach to a given challenge and quickly obtain feedback on new feature development all without scheduling extra meetings or formalized training. At that point in the transformation, they would cease being "the Delivery Team" and "the Business Unit" and become "unified group who make portfolio management work".

One concern we did have was that the relationship between the development team and the business users would become so integrated that the normal controls in place for product support issues would be bypassed informally. If that happened, it would cause the delivery team to be unduly distracted with requests from the business users. In anticipation of that event, we reminded the delivery teams and their product owners of the accepted procedures for routing production support. The business users and application support team all agreed to follow the current protocols for production issue resolutions. Additionally, for the first few months, the application support manager planned to monitor the situation closely and intervene if necessary.

And finally, we thought the collective consciousness around development knowledge, practices and procedures would endure. In short we expected to improve standards; the move was supposed to increase overall quality. After an initial period with only Team B colocated with their product owner and business users, Teams A and D were moved to sit with their respective business teams, as well. Team C stayed in the main development location.

3. The Outcome

After watching the colocated teams work for a few months, we were surprised to see that the expected changes weren’t happening as planned. The goals were not being met and more concerning, two of the delivery teams started to drift from our standard development practices. Initially we addressed the individual issues as they came up, thinking the issues were small adjustments needed along the way. Eventually, we realized a bit of a retrospective was needed to determine the root cause of the difficulties experienced. The development managers stepped back to sort out the pros and cons of colocating to

determine how and when it would work in this environment. This section covers the observed results of our experiment and some analysis of our experience.

3.1. Overall results

Happily, none of the teams were barraged with direct requests to handle production support issues. All of the business units respectfully followed the directive to continue using the standard procedures of opening a support ticket first. The Application Support team would handle the request and loop in the delivery team on escalations as necessary.

From a technical aspect, spontaneous and relatively instant design reviews were common in the old scenario, but no longer available for the colocated teams. The business users are not software developers; they can't give input on the delivery team's design. The other development team members, who were located in teams elsewhere, were simply not available to critique implementation plans as the plans were being discussed. There were no formal design reviews, and the colocated delivery teams did not seek out additional points of view. Their pool of problem-solving resources and idea-generators shrunk to only their team. This impacted all of the delivery teams, even those not colocated with their business users. The colocation experience highlighted a need for a formal process, which will be discussed in the next section.

All those good habits we thought our teams had? They were not habits after all. There was no alignment with accepted practices - just peer pressure to follow procedures. Without the peer group and the constant reminders about code reviews, robust integration tests and sound deployment plans, the teams started to abandon protocols.

3.2. Team A results

Team A was well-established with a defined process that worked for them. They already had a very good relationship with their product owner and business unit and had a good grasp of the business domain. While colocating was useful, it was not responsible for a major leap forward in efficiency for the delivery team. During the initial period of being embedded with the business users, the team’s productivity and quality of work stayed the same, and then eventually improved. The team’s release cycle time went from almost 3 weeks per release down to one and half weeks per release. Their hotfix rate remained less than 20%. The only re-work required of the team after colocating with their business users involved a project that touched other business groups. The team did initially decide to discontinue some of the routines included in scrum, specifically daily meetings and release retrospectives.

Overall the product owner for Team A reports increased satisfaction with the colocation and improved efficiency for himself by having the delivery team literally within arm’s reach. Team A’s members have gained a better understanding of the ways their end users interact with other business units in the company and have used this information to influence their design and user-acceptance testing practices. When asked about the change, this is what he had to say, “Closer collaboration has made both halves of our collective team better. The business owners cannot always “speak technology”; having the developers learn to “speak business” has proved successful in reducing implementation struggles arising from communication gaps. I have been pleasantly surprised at the rate of feature delivery. There have been numerous instances where the developers built with the future in mind; they were able to implement incremental features far quicker and easier than in the past.”

3.3. Team B results

Team B consisted of development team members who had experience in the general development department at Parametric, but the individuals had not spent much time as a small team when they were moved to sit with the product owner and business users. After colocating with the business users, the communication did improve between Team B and their product owner and business users. However the knowledge transfer regarding the business processes was not as fruitful as expected. During the initial period, the team discontinued most of project processes, including daily meetings, defining discrete tasks

during iteration planning and retrospectives. Their aim was to follow a very lean implementation of Kanban and felt it was feasible given the close proximity of the team members. Surprisingly, these changes resulted in longer development cycles and an increase in post-release production issues, including several that required a rollback to the previous version or immediate hotfixes. When the team first moved to sit with the business users, their release cycle time was roughly two weeks per release. By the end of the year, it crept up to more than 5 weeks per release. Over the year, one third of all their releases required a hotfix or unplanned rework of a feature.

While the relationship between the team B and their product owner and business users did improve in some areas, it was in a more casual sense. The groups don't share a common language to exchange "technical" information. The individual delivery team members possess a varying degree of understanding of the business domain and jargon. The greater the understanding, the more information the team member picked up during the immersion. For the team members with less knowledge, the business conversations going on around them were a source of noise and chaos rather than enlightenment - a distraction from their work. When that occurred, headphones went on and it didn't matter what was discussed around them; that information would not be assimilated. We misjudged the base level of business knowledge needed and the motivation required to gain that knowledge independently. In turn, this stifled the desired collaboration.

In retrospect, the very lean approach to development meant the team could not rely on specifications or detailed task-tracking to ensure all requirements were met before finishing a feature. Often acceptance-testing turned up missing criteria at the end of the cycle and required the team to rework the feature and delay the release. Sometimes a design flaw was not caught until after release resulting in the need to back out the new feature or create a critical patch. The lack of business knowledge hampered the team’s ability to be forward-thinking with regards to making design decisions that would also support future development.

3.4. Team C results

Team C remained in place, not moving to sit near their business users. Their process and performance remained steady during this time.

3.5. Team D results

Team D did not experience any benefit by being closer to their product owner and business users. The team was a newly formed team, with a majority of the members new to the company. When located with the business users, the only space available was near staff members who were also new to the company. This arrangement did not provide Team D with a useful source of expert information in the business domain. After two months of inconsistent efforts to perform maintenance updates to an existing product base, it became obvious the members were struggling to work together as a team and keep up with the accepted practices and processes while located away from the main development department. During that time, they did not meet the required standards for release quality. Their work re-introduced some bugs they had fixed in prior releases and resulted in the need to roll back the released versions. The decision was made to move Team D back to the general development location to give them support from another existing development team and the department management while they established themselves as a team.

4. Process changes and results so far

In response to these findings, we have made some significant process changes and continue to adjust as we go forward. The goals for these changes are aimed at improving the communication among the teams as well as giving management a better overview of all the projects currently in process and upcoming. By exposing the high level details of each team’s projects, we are able to take advantage once again of the knowledge contained in other teams. We have also gained more control on planning for upcoming

projects. These changes applied to all teams, whether they were located with their business units or in the development area.

We also looked at the processes used by the individual delivery teams. After separating from the main development team, two delivery teams (Teams B and D) drifted into inconsistent practices; if they thought no one would notice or be impacted. (And they deemed this decision to be valid under the guise of self-organization.) The declining condition of their output - inconsistent quality and sometimes chaotic delivery schedules resulted from that decision. To that end, the management team and scrum master began to work with each team to establish processes that meet the company and development department goals, while still allowing the individual teams some ownership over their work.

4.1. Overall development team goals

The first change was a rework of our daily "Big Scrum"; the meeting consists of a representative from each delivery team, plus a representative from DevOps. It is still a time-boxed, ten minute meeting, but rather than giving a generic status for each group, high level details are presented. Reported items include general areas of the code base that are affected by current work, changes in expected release dates and the progress of the feature. This raises the level of notification among the teams and has cut down on implementation conflicts in co-owned code and dependent services. Deployment scheduling has improved, as well. When a potential conflict arises or a blocking issue is raised, the attendees can figure out the best way to address the issue or seek out additional resources to assist the resolution.

In an effort to correct the deviation from standards we observed and prevent it from occurring further, we formalized our architecture and design review processes. Rather than relying on it happening organically, we added official "steps" to our overall project plan to review architecture, test planning, infrastructure and documentation needs. The extra process may not sound particularly "Agile" to some, but the small increase in planning is showing results. We have seen a decrease in post-release production problems and repeated refactoring. Additionally, a small group of developers embarked on an exemplar project to illustrate our accepted patterns and practices. The overall team has this project as a reference to guide their design decisions and verify they are meeting the expected standards. After applying these changes, Team B’s release cycle time dropped to slightly less than two weeks per release, and their hotfix rate dropped to less than 20 percent. Team D had several successful releases.

Our functional teams now meet regularly to determine and discuss best practices, explore new directions and technologies, and work out problems facing individuals in their roles on the delivery teams. For example, the Quality Assurance team gets together weekly to review their latest test efforts and generate ideas for approaching upcoming projects on their assigned team. One tester reported she “felt disconnected from the rest of the testers after moving to the business area, but the weekly meeting helps to know what everyone else is doing. What they learn and what they struggle with.” Another example is an architecture team that meets regularly to discuss the big picture for our projects and make platform level decisions. If one team wants to employ a new technology, it is reviewed by the architecture group to determine whether it's a good fit overall for our project portfolio. This alleviated the need for each team to make its own third-party choices for implementing grid-handling, data-mapping and test tools.

We have also made an active effort to generate more team-building opportunities while the development team members are located in disparate physical areas. We hope to strengthen the personal relationships and contribute to the pool of knowledge by building connections and developing common interests. Some examples of this include a weekly lunch outing that any member of the development department can attend and a subscription to a technology webinar series attended by a cross-functional group interested in exploring new options for software development. After all, it is easier to seek out advice or a second point of view from a coworker with whom you feel comfortable.

4.2. Individual delivery team goals

Each delivery team met to define their own working agreements to guide projects and daily work. This activity enabled them to take ownership and be self-organizing. Their only requirement was to have a defined process and follow it. For Team A, it was a matter of capturing the process they already followed and getting formal agreement from all members of the team. For Team B, it took several iterations to come to an agreement to which everyone could commit and follow consistently. Team D continued to refine their processes even after moving back to the main development area. This effort is still a work in progress, but is starting to pay dividends.

Teams A and B had drifted away from daily standup meetings. Their rationalization was that they all sat together in close proximity to each other and the product owner, so everyone should know what the other members of the team were doing and when they needed help. By not having a daily meeting, it became apparent that the quick meeting served a bigger purpose than to simply inform. It served to reinforce the agreement between team members to finish the tasks to which they committed by the next meeting. We reinstituted the daily meeting as "little scrum" as a requirement for each team. The product owners had a clearer picture of work progress and could make better decisions to adjust the scope and target release date for the current project. The team members checked each other on the progress of individual tasks. For example, if a team member said he would complete a particular task on Tuesday, yet he still reported it as "in process" at Wednesday's stand up, the rest of the team offered to help or at least would want an explanation why it wasn't done yet. This motivated a struggling team member to ask for help early, rather than hiding his problem. The daily meetings also had the benefit of quickly exposing any ambiguities in the acceptance criteria or implementation design. More than once a simple status update spawned an hour-long discussion that clarified a misunderstanding, saving the team complicated rework later.

Teams B and D suffer from their lack of business knowledge. Rather than relying solely on their product owners and business users to fill in the gaps, the more knowledgeable members of the development team stepped in. They organized a series of presentations to give the newer members of the department a baseline understanding of the business we do, the jargon and where to find more information. As the team members better grasp the industry specifics, they ask more intuitive questions of their product owner and business users.

And finally, we revamped our tooling to make each team's work visible to the team and the product owner. We switched our electronic tracking tool to one that would let each team design their own task board and backlog, rather than making them all fit into the same template. This change tied in with the working agreements and gave the teams the freedom to be truly self-organizing in their process and to make that individuality visible. We found the delivery teams are more likely to keep their task boards up to date in both content and structure if they have ownership of that board and can tailor it to their needs.

5. Conclusions and recommendations

In conclusion, we found that well-established delivery teams who have engaged product owners make good candidates for colocating with their business users. Teams that are in the "Norming" (Tuckman, 1977) stage of development may find that colocating with their product owner launches them into the "Performing" stage. They have standard practices and working agreements set; they are "in the groove". Further immersion in the business domain elevates them to the next level of productivity. Colocating with the business users is not a reason to abandon the accepted practices and processes that make a team cohesive and effective. If anything, it is more important to hold to those best practices and routines that bring consistent quality to the software development effort.

On the other hand, new teams or teams with existing problems will likely not benefit from the move. They need support of their peers to establish themselves as a team first. It may help to have the product owner to move closer to the development team, if possible, and be a part of the team formation. They need to get comfortable as a team before embedding with the business users. Likewise, the team should work out any team issues, i.e., get out of the "Storming" stage, before attempting to be present with the business

users. These teams have all the instability they can handle; moving them to a new environment would incur too much change. At the very least, a move could stall their effort to become a cohesive team; at worst, a move could decrease their productivity or damage their credibility with the business users. These teams can be assisted by coaching to define – and refine - their working agreements and processes to improve the team first. After the team stabilizes, consider colocation with the business users.

At Parametric, we plan to continue offering the option to colocate a delivery team with the product owner and business users, but will make the decision on a team by team basis. When we have a delivery team willing to take the next step in their performance progression and an engaged product owner able to incorporate the delivery team into the normal business flow, we will make the move. As always in an Agile environment we will continue to inspect the outcome and adapt as necessary.

References

Cockburn, A. &. (2001, November). Agile Software Development: The People Factor. *Computer, 34*(11), pp. 131-134.

Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams.* Addison-Wesley Professional.

Cunningham, W. e. (2001). *Manifesto for Agile Software Development*. Retrieved June 23, 2013, from Agile Manifesto: http://agilemanifesto.org/

Eccles, M. e. (2010). "The impact of collocation on the effectiveness of agile is development teams.". *Communications of the IBIMA.*

Krumins-Beens, I. (2012, February 14). *Sitting the Teams Together: Value of Colocation*. Retrieved from Ilio on Agile: http://www.iliokb.com/2012/02/sitting-teams-together-value-of-co.html

Tuckman, B. W. (1977). "Stages of small-group development revisited.". *Group & Organization Management, 2*(4), pp. 419-427.

Waters, K. (2008, May 20). *Agile Colocation*. Retrieved May 2013, from All About Agile: http://www.allaboutagile.com/agile-colocation/