Winning the Concept to Cash Game with Feature Teams

MARTINA ZIEGENFUSS, intuit
JIM ELVIDGE, AgileAccelerate LLC

As an Agile Coach and Technical Program Manager for a large organization, we embarked on a grand experiment to break existing product and engineering structures and form “feature teams”, driven by a charge to break down silos and delivery to market quicker. After many experiments to line up team cadence or to implement tools to weave together schedules for many years, the teams were tired of feeling just cogs in a big machine with no real visibility into the overall delivery scope. The idea of creating “feature teams” had been on the shelf for some time, but without executive support, it was hard to get these teams structures for success. Once we convinced company leadership to try a few experiments in this area, the concept became a reality. It was an exciting ride, watching that idea become reality. It allowed us to capture some powerful learnings about how this structure can work well for an organization and when to pivot when there are pitfalls. What happens though, when the foundations you build a concept on shift or even disappear? Sometimes those can bring even greater learnings than the successes you celebrate.

1. INTRODUCTION

The quote: “If you want to go fast, go alone. If you want to go far, go together.” is sort of true. In the world of most complex, layered software companies, multiple siloed teams trying to “go it alone” rarely works well, and certainly not painlessly. Trying to weave the schedules of multiple teams together to “go together” also presents challenges of work stop/start and priority changing over time. In those large companies, we often hear (and say), well, if this were a smaller company with engineering focused on a single thing, of course this would be easier, but we’re not like that!

But what if even a large corporation could “be like that.” In this way, the idea a single team working on a complete deliverable started to take shape for us. The team we established (call it a feature or an initiative team) could go fast because they were alone—free of dependencies. Everything they needed for success was contained within that team. They could also go far, because as a team, the felt the pride of delivering successfully and holistically, and they could continually grow their collective skills for the next deliverable handed to them.

2. BACKGROUND

Jim and Martina embarked on the Agile transformation of the company they worked for with a true “odd couple” mindset. Jim’s broad understanding of the destination we needed to get to sometimes clashed with Martina’s knowledge of the map and the pitfalls we would surely encounter on our journey. Engineering teams readily embraced the transformation story, but like many companies, they would at times diligently follow the “rules” of agile without really embracing the mindset of agile. It was in the ideation and creation of feature teams—a step taken when all teams were following similar scrum practices, but not aligned cadences—that this duo came together to launch a new concept for the company. The proposal was to create “feature teams” to tackle and solve complex initiatives that would normally span across multiple dependent teams and require many weeks of waiting for alignment and dependency mapping across siloed teams.

3. OUR STORY

Discussions about the possibilities of using feature teams were commonplace amongst agile coaches and engineering leadership from the early days of our agile transformation (roughly 2016). Engineers were eager to experiment with this model, but leadership support of the concept was always tentative. Two overwhelming factors continually prevented teams from taking the leap—territorial organizational structures and managers, and a basic fear of change. Many times, we heard the admonition that “now is just not the right time,” and “maybe when things slow down, we can try something new.” Our suspicion was that, per Conway's Law, the structure of software tends to resemble the organizational structure held true—our software delivery model
was the way it was with teams structured to fit the organizational model. Each engineering leader “owned” a piece of the product AND the designated developers who wrote the code for that piece. Layers of upper management owned bigger the chunk of the product and those boundaries were drawn even darker.

As a result, thinking through the idea of a full stack team led to two potential organizational implications:

1. Every person on a full stack team might report to a different manager and some managers might “lose control” of the teams they “owned.”
2. Managers would no longer own product components, the thing that made them (in their minds) relevant in the organization.

The reaction to the first option was: “How will I have the time to attend various meetings and keep track of the activities of all of my people if they are scattered across multiple teams?” The reaction to the second option was: “If managers no longer own components, they will leave because that is what motivates and defines them.”

If you’re paying attention at this point, you can see that this element of the Agile transformation of the company was still the highest hurdle to jump. Engineering teams may have embraced many of the agile concepts, but the role of the people manager in the organization had not taken hold, and this is what we ran up against in proposing what was considered such a radical idea.

Managers stuck in this component ownership mindset often say that separate component teams were more efficient because the team had expertise in their own component and can build shared to support ALL dependent teams. This argument aligns with a service-oriented architecture model. While there are merits to this concept, when we dug a little deeper, we often found more deeply rooted issues. Always at the root of these arguments was the manager’s fear of losing their relevance and their positions.

It was clear that such a significant change could only come with executive buy-in and with a new path for engineering managers coupled with a robust plan for change management.

Three years into our agile journey, the time might be right for such this shift in mindset and in practice. The question for us was how to convince leaders of this as well!

Company leadership, and especially data-driven engineering leadership, is influenced by real internal data. It is sometimes the only thing that is an effective counter to the argument that “we are different/industry research is academic/this just doesn’t apply to us.”

As those managing large-scale, complex projects for the organization, we were well aware of critical initiatives that seemed to take forever to get off the ground, much less muscle their way to completion. We also intuitively knew why—multiple handoffs and coordination between the multiple, highly-impacted teams. For any given initiative, the priority of a couple stories for one component relative to the rest of their backlog might be very different from the priority of a couple stories for another component relative to their backlog. This might be because of a lack of alignment amongst the myriad product managers involved, or more often than not, multiple high-priority initiatives jockeying for attention in the ever-shifting landscape of customer growth and retention. Yet even when the business and PMs were aligned with priorities, there was still the effect of passing partial work around with queuing delays in between real integration activities, not to mention varying release cadences for these component teams, which tended to add days to weeks in between handoffs.

In order for this to work, we needed proof that our intuition was accurate, and we needed executive approval to try some experimentation with our feature team idea. As Agile coach for the organization, Jim led the way in making that case for engineering leaders at their multi-day offsite meetings.

To put together our case, we decided to select 2 high-profile projects and do detailed value stream maps on each. In one case, we focused on the requirements elaboration process. In the other, the focus was on the development process. That second case is what we will cover in this report.

Figure 1 below is a graphical depiction of the value stream of that development process.
The 12 swimlanes indicate the activities of the 12 teams involved. Pink boxes are largely spikes and information flow and yellow indicates actual development stories. Digging into every Jira ticket, we analyzed the effort expended for each activity. The result was shocking! For a project that took 6 sprints to deliver, the aggregate effort expended was equivalent to what a single team could do in a single sprint. We calculated the flow efficiency of our current process (using this initiative as an example) to be 16%. It should be emphasized that this was by no means an outlier in terms of flow efficiency. Back of the napkin estimates of other similar cross-domain initiatives were also in the 15-25% range.

This became the catalyst for change. When presented with the facts of what was going on in the trenches, senior leaders were quick to recognize that while feature teams might sacrifice some near-term efficiency, they would ultimately allow us to get features and initiatives out the door much faster. In addition, there were possible long-term benefits such as: improved morale resulting from new challenges and ways of thinking; a distribution of expertise and opportunities for cross-training; and the promise of a stronger more resilient organization due to the implementation of these cross-functional teams.

A decision was made to identify a set of vertical stack teams—six in all. This was a start and represented less than 10% of our total number of Scrum teams. But we knew we had to prove the concept for it to gain acceptance. Six teams would have to do!

Individuals were selected to be on the teams based on specific criteria (cross-domain knowledge was a plus, but just as critical was a highly collaborative mindset openness to try new things—fail fast and recover quickly). Initiatives (defined as high-profile, complex, cross-component work in our company) were selected for teams based on similar criteria (involve multiple component work, but also have a bit of slack in terms of schedule expectations to allow for knowledge transfer and ramp up time). We recognized that self-selection, both in terms of team design as well as project selection, would be a more ideal mechanism for creating teams and selecting work. Recognizing, though, the lack of appetite for too much change, we made the conscious decision to postpone that until later when the concepts we were introducing were more accepted. There were plenty of other challenges we needed to tackle!

3.1 Challenges
The largest challenge we faced with implementing this solution was managing the change alongside leadership expectations of the change. Although at the most senior level, there was support for the feature team idea, agreement and alignment was mixed at the Director level and fear often took over at the Manager level. So, there was by no means a full willing leadership supporting the change.

Add to that mixture the need for collaboration with the teams that natively support these existing components. These components didn’t go away just because we took team members away to work with other team members collaboratively. We had to also be mindful of the incremental load knowledge transfer and mentorship would be on these component team members, especially in light of other high-priority projects that depended on them.

Once we started forming the teams, defining roles and responsibilities within feature team because a point of great discussion. Having a single team be responsible for an end-to-end solution should have eliminated the need for release train engineers or traditional program managers, but still necessary was that single voice to
communicate expectations and changes to stakeholders. Could Product Owners or Scrum Masters fill that void? This model also put a greater reliance on full-stack solution architects, rather than the traditional component architects we normally worked with. Did we have any of those?

As we peeled back the onion, we discovered many challenges that were more practical. Could teams create an environment that allowed development, integration and testing across all of the components and foster this collaborative mindset? Would there be difficulties in sharing code bases that were previously accessed only by a small, siloed team? Who would help teams write stories in holistic vertical slices rather than the component-specific methods they were used to that they were used to? Our Product Management organization was also siloed to manage parts of the puzzle rather than look at the whole.

Most importantly, how would we know we were successful in our experiment? Did we need to develop new metrics for success?

3.2 Addressing the Challenges

Tackling the leadership change management challenge head on, we established a support structure consisting of senior engineering leaders from all development domains. That team met weekly, and their objective was to address any overall roadblocks that came up related to the newly formed feature teams. We tracked issues and resolutions on a fully transparent Confluence page; assigned owners at the meetings; and discussed solutions to each issue. These meetings also allowed us to keep up with the tolerance level for the change and provide feedback on where teams may need additional time or how configuring the teams differently might ease some of the roadblocks.

All engineering leaders were required to provide subject matter experts to assist the feature teams new to the model and learning new skillsets. Teams were encouraged to reach out to their designated SMEs for support and to be collaborative when it came to finding the right timing for these discussions. We knew the SMEs would see a higher workload as a result of this change, and the goal was not to frustrate existing teams delivering against other company goals.

To address new roles and responsibilities, the Scrum Master and Program Manager roles were combined into a single role, which we referred to as “Team Captain.” Although this had the sound of a directive role, it was a conscious decision to balance traditional Agile best practices with the fears that without a traditional Program Manager the team may lose focus. This Team Captain had their feet in both role—they worked with the team on best practices, organization, and guidance and managed up to leadership by clearly communicating the achievements, issues, and risks to the team and the overall initiative delivery.

No managerial changes were made in terms of reporting structure; hence, we followed the model of each team member reporting to their previous (domain) manager. The domain managers were not engaged in the feature team, but provided more employee mentoring and development to the feature team member. In addition, rules for engagement were worked out for all members of the team including architects and SMEs. Suggestions for success were worked out and presented to team members, team captains, product owners, component managers, people managers, architects, and SMEs with the goal of creating an environment of transparency and collaboration.

As mentioned before, other parts of the organization—specifically Product Management—were also hired and trained with this very siloed component mindset. We had to reach across to that organization to provide someone who could lead these new feature teams and write those vertical requirements along with the team. Or, in cases where we couldn’t take a single person away from their role to provide epics for the team, we had to train the PMs to write in a way that was different from their usual models.

The setup was all good, but once we started the teams on this path, we needed to know if our current metrics would need to be enhanced or even recreated to measure the outcome of this experiment. To that end, Jim identified a set of metrics we started with, both at the initiative level and the team level. Team metrics included our standard velocity and work breakdown (type of work categorization), but we added 2 new measures to track:

1. Job Satisfaction was measured by a simple survey with two questions.
   a. How satisfied are you with current role and situation? By measuring at team formation time and then regularly during times of success and times of challenge, we could see trends
   b. How does your satisfaction compare to when you were on a Component Team? This was measured a few months into the experiment and to provide a comparison.

2. Cross Functionality was measured at the Initiative level, using our standard metrics and artifacts of Relative Business Value (on each epic) and Burnup chart. In addition, we added the metric of Flow
Velocity. This was determined by simply asking each member of the team, every sprint, the question “How much of your time was spent directly developing on the initiative versus non-initiative activities, such as learning, struggling with environments, or unplanned work unrelated to the initiative?”

3.3 Results

Overall, the feature team experiment was a strong success. The metrics collected verified the key results we were seeking. The main objective of course was to improve flow efficiency of cross-domain initiatives, i.e. get stuff out the door faster. Figure 2 below shows the results for the 6 initiatives that were delivered by feature teams. The average flow efficiency was 80%, which compares extremely favorably against the average of 15%-25% that we saw from component-team-based initiatives.

<table>
<thead>
<tr>
<th>Initiative</th>
<th>Team</th>
<th>Average Flow Efficiency</th>
<th>Steady State</th>
<th>Ramp up Time (sprints)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiative A</td>
<td></td>
<td>98%</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>Initiative B</td>
<td></td>
<td>67%</td>
<td>100%</td>
<td>2</td>
</tr>
<tr>
<td>Initiative C</td>
<td></td>
<td>91%</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>Initiative D</td>
<td></td>
<td>93%</td>
<td>95%</td>
<td>0</td>
</tr>
<tr>
<td>Initiative E</td>
<td></td>
<td>28%</td>
<td>40%</td>
<td>4</td>
</tr>
<tr>
<td>Initiative F</td>
<td></td>
<td>100%</td>
<td>100%</td>
<td>0</td>
</tr>
<tr>
<td>AVERAGE</td>
<td></td>
<td>80%</td>
<td>89%</td>
<td>1.33</td>
</tr>
</tbody>
</table>

![Figure 2. Flow efficiency metrics for cross-domain initiatives](image)

Since we asked each team to estimate their flow efficiency every sprint, we were able to see how it changed over time. At the beginning of each initiative, it tended to be lower than at the end, due mainly to the need for knowledge transfer and time to learn new components, code bases, and domains. The steady state level averaged to 89% (measured toward the end of each initiative) and the average time that it took to ramp to that level was 1.33 sprints, or about 2-3 weeks.

We expected that both the ramp up time and the initial flow efficiency for new initiatives would increase as teams took on new work, assuming, of course, that there would be less learning needs as teams broadened their subject-matter expertise. Figure 3 below demonstrates the metrics that we collected regarding team satisfaction and subject matter expertise.

![Figure 3. Team satisfaction and Subject Matter Expertise (SME) metrics](image)
As a comparison against how team members felt when they were on component-based teams, average team satisfaction increased by 30% over a period of 6 months. Note that there were a couple outliers, and it was valuable to identify the reason that they gave for those cases. Our goal ultimately was to improve the concept and any information was good information, even if it didn’t fit with the model we wanted to see.

In one notable case, the team’s satisfaction level dropped 50%. The reason that they gave was that they essentially felt that they were “constantly learning new things and not given time to apply what they learned.” This particular team had been taken off of their core domain and given to a completely different product area a few months before the feature team experiment began. Then, as a feature team, they were given yet a third unrelated product area to work on rather than one of the two that they were already familiar with. We believed that over time, this effect would have subsided, but ultimately, the management team decided to pull them back to their original product area, so there is no way to tell.

The other team that registered a drop in satisfaction did so more because that particular initiative was plagued with poorly prioritized and groomed work, so the team was frustrated waiting for things to work on. Anecdotally, their satisfaction level increased as time went by. Efforts to add POs to take ownership and properly groomed the work ultimately resulted in a more positive experience over time. This taught us that keeping a robust backlog of work was critical to maintain as the team’s expertise grew. They, in effect, became too fast for the product managers to keep up.

The other four teams registered a high level of satisfaction increase, with three of them claiming an improvement on satisfaction of 50% or more. Anecdotally, many members of the teams cited these reasons for their increased satisfaction:

- Ability to work on a full solution rather than a small component generate a greater sense of value and satisfaction;
- Opportunity to learn new things was a welcome refresher from years of being stuck in the same code base;
- This structure involved less micromanagement and statusing than before, due to the fact that teams were now fairly autonomous, and their composition included members from different domains.

The other team metric that we measured was “increase in component Subject Matter Expertise.” Since teams filled this out at various times during the projects, we normalized it by annualizing the metric. On the average, teams felt that they increased their knowledge of and ability to understand the code base of previously unfamiliar components by 38% annualized. Interestingly, the team that registered the greatest increase in subject matter expertise was the one that had the lower satisfaction rate while the teams that had the highest satisfaction rates registered the lowest increase in subject matter expertise. The other three data points, however, are not correlated to that trend and with only six data points, we would caution too much of a conclusion from this observation. For us, it opened up more interest in exploring and iterating on this idea.

With any aspirational vision comes the realities of the environment you are working with. As alluded to above, a major change in leadership occurred about 6 months after the creation of the feature teams. Senior management who had been supportive of the feature team’s work left the company amid major layoffs and restructuring. New leadership did not embrace the concepts of Agile and did not particularly understand the benefits and values that the feature teams were bringing. Their focus was on the majority of the engineering teams who were still dunkily “delivering something” and employee satisfaction wasn’t their primary focus. Our greatest learning from this was the original ingredient needed for this recipe to work—leadership support for feature teams and the mindset shift needed there. As a result, executive support was lost, and some of the teams were disbanded and pulled back into their original component-based ways of working. There was extreme disappointment (as well as resulting attrition) among many of the members of these teams as they had tasted a way of working that involved greater autonomy and a large solution-level vision.

4. WHAT WE LEARNED

In the end, we came away with key learnings and guidance which we feel would help any organization thinking of this sort of team structure:

- Getting and maintaining leadership support for innovation is critical. A structure like this is very difficult to implement from a “grassroots” level, so finding ways to “sell” your proposal using data-driven and also anecdotal reasoning is powerful.
- Change management is key to make a transition to feature teams successful. Senior engineering leaders must be on board and fully supportive of the change, or the journey will be painful, with many
challenges. Key to change management is establishing methods for code sharing that allow for new teams to develop in a code base without fear of significant disruption in quality or impact to regular release schedules of that component. Automation and good DevOps practices will be key to facilitating this. It helps to have a support management team to address the impediments as they arrive. The team can be loosely structured as a Scrum or Kanban team.

- It’s best to give the team some schedule space to learn. Hitting them with high pressure tightly scheduled projects will not be a recipe for success.
- Create your first teams with SMEs from each of the components that they will typically work with or around the structure of an initial 2-3 example initiatives. We learned the hard way on some teams that just taking individuals with a single set of skills and putting them on a separate team does not a “feature team” make.
- Flow efficiency can be drastically improved with a feature team model over large programs of many component teams. Such a benefit can be realized immediately.

One of the key intangible successes we both got from this experiment was to see feature teams work in good and in not so good times. We were able to discuss and measure individual and team successes with this model. Members of those teams that experienced the transition (including us) have moved on to other companies as well. Our hope is that the spark we set in this one controlled area may find its way to burn brightly in other organizations.

5. ACKNOWLEDGEMENTS

This paper would not have been possible without the actual teams whose stories we are telling. They are the heroes who went through the triumphs and pains to make this grand experiment a success. They had fun and inspiring names like Lighthouse and Polygots, and Honeycomb. But for us, they will always be inspiring friends who taught us that “going together” is best. So, thank you to the teams who went along with this crazy idea. Also, our success would not have been possible without two key champions—the Senior Director of the PMO and the VP of Engineering at the company we worked for. They gave us the “aircover” to try, fail, iterate, and succeed. Lastly, many thanks to Niels Harre for his evaluation and feedback on this paper and to XP for allowing us to tell our story!