Abstract

The key to small teams & startups is agility/speed as against any well defined process/ framework. The only catch many a times is that the conditions of agility are also left to the teams. People believe that the power to be not bound by even the lightest of the processes, is agility. However, as time progresses and the team scales in leaps and bounds in terms of people and business, the need for a structured approach starts to emerge, unsettling many. The tussle between predictability vs speed starts to arise. The situation can be fondly called “The Controlled Chaos”.

The presentation draws case studies across projects, programs and a portfolio aiming to deliver multiple simultaneous releases at a sustainable pace, amidst a lot of moving pieces. The case study is based on of 30 months of an Agile adoption journey.

The Organization & Team Context

The organization in context is an ad network where the business needs are continuously changing due to the rapidly evolving advertising ecosystem. The players in the market are continuously thriving to bring in new set of features to beat each other, competing for the fast mover advantage. The dynamics of the business is such that even to plan a quarter is pretty tough for the business and product teams. This can potentially cause a lot of churn in the backlog if not prioritized well for the teams who are executing the various increments to the product. And the complexity either alleviates or enhances by how the teams are organized.

In this context of the organization, the teams were broken down as per the **architectural** stack, leading to a high dependency matrix for each of the releases in the organization. The stack team structure was decided based on the nature of the business and the level of effort across various teams in the value chain.
Case Study
Scaling from Project > Program > Portfolio

Context of Project

Project here refers to the releases done within each stack team (contained within the stack than cross-stack based). In the above figure, Team A, Team C etc can be called project (stack teams). There were releases being executed within the teams itself.

Context of Program

Multiple versions of releases defined a program. It was part of the portfolio as mentioned below but its primary objective is to meet the program goals. One of the example would be a SDK (software development kit) that was heavily developed within one team but had dependencies on other teams. Each version of the SDK had specific program goals to be met.

Context of the Portfolio

All releases in the organization fell under a set of 8-10 themes. Themes varied over years and the investments / releases were retagged or realigned.
The case study is about how a single framework was able to address the issues at project, program and portfolio level.

The Storyline

Q3’ 11  Q3’ 12  Q1’ 14

There was enough momentum in the system (with small teams) and so was a lot of churn in the backlog due to business priorities. Many stack teams were not having focused planning and execution cycles. They were either going by the flavor of the month (driven by competition, state of union) or the sudden priorities that crept up. The iteration and deployment rhythm was visibly shaky and they were just executing on a daily / weekly priorities. There were instances where the teams dropped what they were working and moved on to an item with higher priority as the business demanded. **Amidst all these, there were some teams who were diligently doing weekly releases.** All teams however wanted a clear priority to execute so that all teams could align and deliver timely releases. In short, the 20+ releases that were signed up for the quarter were not seeing a clear visibility of getting done. And, there was a huge leap (3x) projected in Q3 and Q4 of 2011 and the organization needed a significant planning and execution cadence. It was banking on Agile to bring in the clarity, visibility and predictability into the releases.

Step 1 : Gap Analysis
1. Clarity on the number of teams contributed to the release. This was a key gap b/w the PO and Engineering due to handoffs, discussions that were not followed up etc.
2. Details and ownership of the technical dependencies (between stack teams) – Which team owned it?
3. Status of each of the stack team deliverable and in turn the entire release?
4. How to breakdown of the business ask (of the release) into smaller requirements that each stack team can independently work on?
5. What will the backlog for the stack teams to pick during the sprints? – Who defines their priorities? Do they follow stack prioritization or release level prioritization? What if one of the items was a prerequisite for the release that needed prioritization?
6. Who and how will we tie up the dependencies to form a big-bang release?
7. How do we bring in visibility into the upcoming sprints (specifically for resource planning than just for technical dependencies)
8. How do we communicate the status of all releases that were being executed in parallel across various stakeholders
9. How to run the stack sprints with production fixes in parallel?
10. How do we manage Code merge and branch out when parallel product initiatives are being run.
    a. When and how to manage the code merge?
    b. Who manages the code merges?
    c. Who manages the merge down and merge up issues?
    d. What if there has been an update to the production by the time the merge happens?

**Step 2: Define The Framework**

As in Agile, it’s about “experimenting” using the manifesto, principles and methodologies. There is no one silver bullet framework that can apply to these situations. A few team members from Product, Engineering, QA and Project Management debated, deliberated and devised the framework that works on delivering multiple releases in parallel with the finite set of resources. The input from the Senior Management / Executive Management was the priority list of releases (with no specific absolute priority as the company had to go in for flank attack than through a stack based priority)

**Gap Analysis**

The key thing was to identify the gaps in the system before the teams could adopt the new way of release planning and execution. To that effect, a quick retrospective was held between the mid and senior leadership team to analyze what were the core areas that needed attention. The focus was deliberately not on tools or techniques but to address those bottlenecks that would help the teams to plan and execute faster and better. The Pareto principle of identifying those 20% things that would give 80% lift was the focus of the team.

A few top-level things that were identified were:
- Release Dependencies that currently go unnoticed and also unattended mid-flight (once the releases were in execution). It was a given that 90% of releases had dependencies.
Case Study
Scaling from Project > Program > Portfolio

- Inter team sequencing of the dependencies and deployments
- Intra-team sequencing of code merges, release signoffs by QA, regressing merged builds
- Constant and consistent visibility to teams and upper management related to state of releases
- Coordination with technical and business operations in communicating the release calendars

Plan the Phase 1
There was a concise quarterly plan to attack all of the problems above. The first quarter was totally dedicated towards:
- Developing a plan of record for tracking all releases (fix the communication issue)
- Analysis of the current state of the release (many were struck at 10%, some at 80% for weeks) – find out the dependencies
- Identifying the hardware requirements delaying releases and address them on war footing basis
- Developing a framework of release thrashing where all stakeholders across various stacks contributed
- Devising a production push process that outlined basic requirements that should be met (outlining the code changes, install steps, pre-requisites, notification and approvals)

The framework of push / pull of backlog items
Releases were tracked as “Epics” (Parent User Stories in Agile terminology). Product Owners owned the Epics and would work with Architects (forming the core team) to break it down into child user stories that carried the technical details related to how the business requirement can be met. These “child user stories” would reside in the team backlogs, get executed and deployed in the sequence that would be defined by the lead architect of the release. So, as you can imagine, stack team A would have an a-la-carte list of requirements from various releases in the portfolio.

The focus during the first quarter was not on any agile tool but a simple Google Doc that was shared with the stack team members. The Google Docs had the child user stories (the sprint backlog), the capacity allocation per stack team, the vacation plans of team members, customized burn down charts using the planned v/s actual values. The Scrum masters were the glue between the teams and used to manage the coordination between the various teams (using Scrum of Scrums as a forum).
Sprints, Stand-ups...

Typically 50% of releases would span across multiple sprints across multiple teams. The core and Scrum teams perform a high level release planning (and breaks it to sprints) to identify the sequence of execution, the dependencies etc. In order to ensure the technical dependencies were managed well, each contributing stack team has to agree to a “contract” that gets defined between the teams that would define how a release would cut across the entire system. The lead architect who is part of the core team would define this contract and pass it on to the teams to elaborate and suggest alternatives or changes.

Also, the expectation was that technical bottleneck of the management of code branches and merge would need to be handled within the stack teams.

Execution Framework - Stack Sprints

Given the nature of the teams, there were thoughts on how to run the various team sprints so that the releases can be executed seamlessly.

Each team had their sprint cycle and had planned to pick the stories up based on their capacity availability and using this as the framework, the ETA for various releases were arrived at.
Though this worked largely in a productive way, the slowness of one of the teams due to unforeseen priority changes caused due to production outages and dynamics of business (getting ahead of the competition, winning an account by making tweaks). This caused the other releases to be delayed.

Virtual Teams

There were thoughts on how to run this seamlessly without disturbing the release flow due to stack team interruptions (production outages, people unavailability due to sick leaves etc) as stated above. The concept of forming virtual teams by bringing in people from various stack teams to accomplish the release goal was tried out.

Though the advantage of this approach provided dedicated capacity for each of the releases, there were productivity hits due to the disparate level of effort needed by each stack team. There were team members who just waited and “feeding the beast” syndrome started to emerge. Due to this, some team members ended up being part of multiple releases which led to them attending multiple standups and time slicing issues when some defects or bugs were found. Burndowns went away as there were no stack burndowns anymore. There was no chaos but this definitely did not seem to scale the way the organization was set up.
What was the approach taken?

Given that both the stack team and the virtual team approach was showing varied issues, we decided to have a hybrid approach.

Storyline Moves

Q3’11  Q3’12  Q1’14

@Scale

Number of concurrent releases: 45 + (from 20 releases in 2011)
(50% releases had cut across more than 3 teams)
Number of engineers: 80 (from 60 in 2011)

The key issues that were hampering the scalability in planning and execution:
1. Prioritization Planning Process - Which releases out of 45 are supercritical?
2. Visibility on each of the release and where each of them were struck in the release lifecycle.
Planning: Prioritization of Releases

A product planning process (PPP) was brought in to manage the entire portfolio prioritization. Given that there was no standard technique of ROI, Payback etc that could be applied for releases, there were often queries on which releases that need to be picked up in the order. Constant intervention from the Chief Product Officer was required to mitigate such prioritization issues, especially when slippages and other production outages were delaying releases. Kanban based approach was provided by which the high priority items were done first.

Tracking Releases Across The Lifecycle

The key thing by which releases could be tracked was to park them in the right state. Each release traversed through:

**Pipelined > Approved > In Ideation > In Execution > In Deployment > Under Review > Closed**

Each of these release states had a gating criterion through which the states moved by the release managers.

Pipelined, Approved & In Ideation

Each release gets pipelined for approval to the executive team where the releases are scrutinized for completeness, business need and ROI and then approved. The approvals are obtained in a larger forum where all other releases compete in the portfolio and stakeholders can substantiate their case. This used to be the global prioritization meeting chaired by the executive management (Chief Product Officer and Chief Technology Officer)

Process: (Primarily driven by core team with help of Scrum Master) - Iterative Process

1. Ideation between Product Owner, UX and Architect (Core team) to arrive at the first level Release Backlog (Owner: PO)
2. Iterate on the low-fidelity mockups by the UX team, if applicable. (Owner: PO and UX)
3. Lead Architects discusses with their peers (within other stack teams) to arrive at the dependencies between stack teams and document them (Owner: Architect)
4. Generate artifacts on the system level design changes (Owner: Architect + Lead Engineers)
5. Discuss with Engineering managers, Leads, Technology operations and other stakeholders (Owner: Architect) (Iterative Process)
6. Discuss with Biz Ops, Business, Sales (Owner: PO) (Iterative Process)
   a. Does sales have a need for this release? Do they have a potential customer base?
   b. Does this require a PR? Does this require an organization wide training?
7. Discuss with PO continuously to identify if the system that is being developed is what is really being envisioned, tying everything together (Owner: Architect)

Output: Release moves into Ideation

1. Child User Stories under the PI (Owner: Architect)
3. Low Fidelity Mocks (Owner: UX)

In Execution

This is the phase where the team can begin the sprints using the release backlog. This is essentially the sprint planning and execution cycle where there is a constant collaboration between Product, Scrum Team, Core Team and other stakeholders with the help of daily standup meetings, status reporting, retrospectives and demos. The key to this phase is that the teams self-organize, work together, inspect and adapt using all the agile tools (planning, daily reporting, retrospectives). Each stack team would use the pull method to move the requirements from phases (dev > qa > deployment)

Input:
1. Child User Stories updated from the previous stage
2. Architectural Artifacts (Design WIKI, diagrams)
3. Sign-offs from the dependent stakeholders (Ops, Sales)

Process: (Primarily moderated / driven by Scrum Master)
1. Release Planning (with few stakeholders) - To identify the tentative TTM / number of sprints (Owner: Architect + PO + Scrum Master + Few key stakeholders from the scrum team + Tech and Biz Ops)
2. Capacity Planning - To identify the availability of time of each team member (Owner: Engineering Manager(s))
3. Sprint Planning and Sprint Execution

Output:
1. Deployments based on Release Dependency Matrix (Owner: Scrum Team)
2. Updated Communication Plan (blocking calendars for demos, look ahead planning for the future sprints etc) (Owner: Scrum Master)

In Deployment / Under Review / Closed

Each release (from all stacks) post-deployment moves and stays in Deployed till reviewed by the Technology QA team. Under Review is the phase where the release is reviewed by the PO for its adoption metrics in the field. Eventually the release goes to a closed state once the release objectives are met.

A mild turbulence - Has Agile Really Worked For Us ?

Q3’ 11        Q3’ 12        Q1’ 14
In Q3 2012, there was a sense of skepticism about whether Agile was really reaping the required benefits to business and engineering. There was a noise in the system that kept asking for speed. Agile efforts were stalled for couple of months and people wanted to do timeboxing with an accelerated effort across all teams. 30 releases were to be completed in 45 days. At the end of the experiment, it was realized that sustainable pace using Agile was far better than adding to the chaos. Agility was brought back as the key force behind the planning and execution cadence.

However, the need to do time boxed “accelerated” release planning and execution was still the flavor of the month. In order to meet the expectation from execs and also operate in an agile manner, the following was proposed.

The product team would come up with the priority of releases and engineering would plan for 2 x 3-week sprints to get into execution simultaneously across all the stacks.

![Team’s Sprint Cycles](image-url)
In this case, the improvisations were done for reducing the number of planning meetings by planning 2 x 3 week sprints upfront. The release standups, weekly sync up between leadership teams, feedback mechanisms continued to drive the sprints and the execution forward.

**Solving for Engineering Gaps**

<table>
<thead>
<tr>
<th>Q3’ 11</th>
<th>Q3’ 12</th>
<th>Q1’13</th>
<th>Now</th>
</tr>
</thead>
</table>

Once the prioritization issues were mitigated, the core issues of engineering lifecycle remained and continued to slow down the progress of the teams. The teams deliberated and brought in better SCM tools and extreme programming practices.

The other core issue that slowed the teams down was the issue of Team A (for instance) not knowing when will the part of work to be done by Team C and D. In short, the visibility into which sprint the part of the dependency would be picked up was unclear.

This was mitigated using the planning approach when all teams had to work on planning their backlogs upfront for at least 3-4 sprints (with lowering % of commitment or capacity loading as we can call it in order to absorb any spillover or slippages from previous sprints) as shown in the table below

<table>
<thead>
<tr>
<th></th>
<th>Team A</th>
<th>Team B</th>
<th>Team C</th>
<th>Team D….</th>
<th>Team X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 1</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>90%</td>
</tr>
<tr>
<td>Sprint 2</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>Sprint 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sprint 4</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>

Using this board, Team A would be aware of when Team C and D for instance would pick up their piece of the release and deliver. The PMO team would then publish the release sequencing and dependencies on a weekly basis and send updates on status, slippages and risks.

Kanban was brought in as a measure to mitigate the issues faced about “Where” in the lifecycle the bottlenecks were. Was it in the code merge state? Was it getting bottlenecked in code reviews ahead of merging to the trunk? Scrum was used during the planning and backlog grooming phase and Kanban...
was used to pick the priority items in the queue and also the swimlanes and the WIP enabled to remove bottlenecks in the entire lifecycle during the 10 day sprints.

Extreme programming practices of TDD, Continuous Deployment and Delivery, Pair programming, collective code ownership than silo-ed development (as in case of virtual teams) were introduced to improve the productivity of the team.

**Scalability in a nutshell - What & How ?**

One might wonder what is scalability in this entire framework. The key thing is that it was designed to accommodate scale as one of the major area of focus given that the organization was anticipating a phenomenal growth. And it happened. There was a multi-million $ funding that landed in the bag and the organization was poised for growth.

- Scaled from 20 releases in Q2 2011 > 45 releases in Q1 2012 > 60 in Q4 2012 > 120+ in Q4 2013
- Engineering team size scaled from 45 to 180 in 8 quarters
- # Teams scaled from 6 to 14 in 8 quarters, growing the dependency matrix

As and when new teams got formed, the release dependency matrix grew and also the framework accommodated.

**Things that were not in focus**

Though these are important, the following things were never done (which could have been)

**Story Points:** Teams focused on either task based effort or sometimes no effort

**Team Velocity:** Given that the team’s backlog was an al-a-carte of stories across various releases and the fact that there was not a single story point guidance that was followed, there was no concrete way of measuring a team’s velocity.

**Recommendations**

It is strongly recommended that the team who defines the framework needs to start off with an open mind for feedback and failures. The first step to defining any framework is a complete analysis of the existing bottlenecks rather than to go with a silver bullet of Scrum, XP or mix of other methodologies. The second significant step is to work with key stakeholders to see if all their demands are being met (else the adoption becomes bleak). It is extremely important to set expectations with senior and executive management on what milestones will be achieved in the adoption cycle. This framework helps teams who have dependencies in managing releases. In a nutshell, the chain is as strong as the weakest link. Hence, it is imperative that there is a unified process across the entire organization so that everybody moves at a sustainable pace.
Here are the top recommendations from the case study:

**Consistent and Continued Sponsor Support**
Nothing much can be stressed as significantly important than getting the sponsor and key stakeholder buy-in especially when it relates to scaling the process and the adoption across the enterprise. The PMO team or the Agile Coaching team should focus heavily on keeping the stakeholders informed and should be in a good position to influence the decisions that need to be taken. Any change management program within the organization is futile without the support flowing all the way from the top. All IT Governance frameworks seem to stress this fact quite significantly.

**Have an open mind to experiment**
One of the fundamentals in the agile manifesto is “Responding to change”. There is no single silver bullet in organizations that will always work. The willingness to change, the openness to incorporate changes in the process, in the tools and the ceremonies will go a long way in obtaining the buy-in and conformance to processes.

**Focus on Engineering Issues**
In our experience, 80-90% of the slowness reported were seemingly within engineering and the practices that were followed. The problem of prioritization, Scrum or kanban processes or the various meetings and their effectiveness were less of an issue than the **engineering gaps**. The key is to identify them as bulk of the time of engineers is to resolve engineering issues and deliver the “product right” with quality. The focus of the program managers, scrum masters and the leads / line managers should be towards mitigating them, if not removing it completely.

**Focus on Architecture and Design evolution than assuming them to be static**
The misconception about Agile being not architecture or design friendly needs to be removed by ensuring these are “shifted left”. Architecture and Design sprints (with the help of architects in the organization) need to be running couple of sprints ahead of the regular team and release sprints so that the requirements / backlog items can be interspersed as and when needed. As organizations scale, systems need to scale too and it is imperative that architecture and design evolve to meet the business needs.

**Break the wall of confusion - Embrace Devops**
An amazingly critical but mostly ignored piece is the way the work gets tossed around (Dev - QA - Ops). Including Operations as part and as a partner in the entire sprint process and also in the larger scheme of releases would ensure they work hand-in-hand than being an impediment to a release being done on time.

**Enhance the collaboration (and feedback)**
Continuous feedback not only during retrospectives but also during informal meetings, hallway discussions will need to be taken seriously. As the teams find the scrum masters, program team being
receptive and removing the impediments in their day-day activities, the adoption would be automatically expedited. The key is about enabling collaboration across the teams.

**Quality is Prime**
Having clear “definition of done” (DoD) at each level (sprint, release, portfolio) is one of the prime requirements to ensure there are enough checks and balances in the processes. Ensuring that many of the XP practices are put in place, there is definitely a significant uplift in the quality. Also, ensuring that QA is shifted left and frequent build drops than doing a “Scrum-fall” approach by doing QA followed by dev complete needs to be avoided to ensure there are no escaped defects and surprises towards the end of the release.

**Measure (what you can manage)**
Agile by itself can provide a vast matrix of metrics that one can capture. However, unless one knows what to capture, there is no benefit in measuring what cannot be managed or acted upon. It is important to know what metrics are important for which stakeholder and make it available that at all times, as part of the governance structure in organizations. Data availability is one of the important decision making tools and hence they need to be measured and automated to ensure there is less skewness due to human intervention.

Q3’ 11   Q3’ 12   Q1’13   Now

----- The Scaling & Adoption Journey Continues :) ----