Scaling the Agile Teams for Large Products

In my previous articles we have discussed importance of establishing cross functional structure and importance of cross skilling with examples from sports teams to ***establish truly independent*** teams *delivering high quality at speed consistently*.

The next challenge that confronts ***engineering leaders*** on this journey is to scale high performing teams for large, complex products. As products footprints grow, organizations often face a steep decline in velocity, increased complexity, and heightened risks. The intricate web of dependencies, coupled with the need for extensive coordination and automation, can stifle agility of the teams significantly resulting in lower speed of innovation with same quality.

In this article, we will explores the root causes of these challenges and provides actionable strategies to overcome them, enabling engineering leaders to successfully scale Agile and deliver value at speed.

Challenges of Scale

Before we deep dive into the strategies of solving this challenges, lets understand deeper as what causes this reduced speed of innovations for large products for the engineering leaders.

- **Increased Complexity: As the projects grow into large-scale it grows both horizontally (serving more needs of same customers) and vertically (depth wise) catering to many variety of business scenario for different users.**

  Complexity Due to Horizontal Growth: Complexity increases because product footprint has grown and while originally product was catering few services now its serving many more services. If we take example of a school software, while originally organization started with administration software, slowly it expanded into LMS, Teacher Management, Special needs for students etc.

  Complexity due to vertical growth: Increases because within each workflow, use cases a product serves is a lot more than it originally started with. This is depth of each function. As more and more customers are added, different types of needs arise and different customers require different business rules. In order to cater to these rules, product has many configurations, settings and business rules underneath.

Increased Cognitive Load: As the system grows horizontally and vertically cognitive load increases and a given set of people cannot comprehend existing system.

- **Changes in Technology: with rapid change in technology, products that are created few years back may require an overhaul technically. New Services may become available such as Serverless technologies or new services in cloud that can simplify existing functionality. Frameworks may get outdated and better frameworks may become available.**
- **Security Vulnerabilities: Security vulnerabilities are identified and these slow down teams in developing new features.**
- **Fragility and Rigidity: Because of the complexity mentioned above and the cognitive overload, large systems also become fragile, meaning small changes can have unintended consequences. This fragility, coupled with the significant investments made in the system, can lead to rigidity, making the system resistant to change and adaptation.**
- **Automation Gap: While for a smaller system and smaller team need of automation may not be that high, as the system grows cost of manual work starts going high and every human error may end up costing both time and money. A larger system would require more frequent changes to various parts of it and not having clear boundaries in the system would need testing larger portions of it every time. Automation gap in such cases becomes a bottleneck for high speed. The absence of robust automation in testing, deployment, and infrastructure management slows down delivery pipelines. Manual interventions introduce errors, increase lead times, and hinder the ability to respond to changing requirements.**
- **Increased Risk: As the system becomes large and its scale expands, the number of users and organizations that a small error can impact becomes significantly high. This increased risk makes engineering leaders to be more cautious and slow down in the rate of innovation.**
- **Knowledge Attrition: As people who have built the system have left and new people have joined, new people have limited understanding of the system leading to slow down in speed of innovation.**

These challenges collectively impact an organization's ability to deliver value rapidly, maintain quality, and adapt to changing market conditions. The consequences of failing to address these challenges are far-reaching.

Decreased speed of innovation leads to losing deals to your competitors and missing new opportunities. Dependency management issues result in project delays and increased costs. It may result in not being able to respond to customer demands within time. Automation gaps lead to quality issues and reduced efficiency. Ultimately, these challenges can erode customer satisfaction and damage the organization's reputation.

Strategies for Solving this challenge

Looking at the challenges above, it can be understood that a multi-prong strategy is required for organizations to gain speed of innovation back with high quality. We will discuss some of these strategies below.

Successful Agile scaling hinges on **architectural refinements** decomposing complex systems into manageable, independent **customer-centric components**, dev-ops/testing automations and establishing cross functional, cross skilled autonomous teams with excellent engineering culture delivering high value to its large customer base at fast speed.

Let's delve deeper into each of these areas to understand how organizations can effectively implement them.

- ### Decompose Systems into smaller units
  Purpose of decomposition should be to arrive at independent modules or services that can be managed independently by different teams without having to have too frequent communication and dependencies on each other. Marvel of a good leader is to be able to successfully decompose a system in such smaller pieces. While this may sound simple, this can become complex in case systems have grown too large and there has not been great architectural and systematic oversight around it to contain unorganized growth of the system.
  - **Modularization:** Breaking down complex systems into smaller, independently deployable modules to enhance maintainability, scalability, and resilience. Modules should be divided in a manner such that each module deals with specific set of customer actions/workflows or experiences. This way, teams focusing on these modules can independently enhance these modules without coming in the way of other teams.
    - **Horizontal Layers**: Such as layers of technical architecture.
      - Create boundaries and rules between these layers. *Architectural rules*
      - e.g. Frontend layer or client layers can call business service layers and not the vice versa. Business layer can call the database layer and not the vice versa.
      - Create clear separation of concern between these layers such as change in one layer keeps the other layers completely isolated from the rest.
    - **Slice your application Vertically in independently manageable services**: By set of features customers use for solving specific set of tasks or problems. Each vertical module may be composed of all layers of technical architecture such as database, backend, frontend etc. However, each vertical module serves specific needs of the customers. These vertical modules should not be too narrow also not too wide. If these can be brought to a size where these can be wrapped into

one service and can be served by maximum one to two teams that would be an ideal size. In this way you can create focus for each of your scrum teams. Each of your scrum team should independently be able to focus on certain aspect of your application that you want to keep working on for long period of time and keep improving upon. Something that gives you significant advantage wrt your competition in short and long run.
    - Create
- The Spotify Model:
    - Under spotify model squads, chapters and guilds are created to ensure that squads are customer focused catering to improving set of experiences the team is going to improve upon while chapters will take care of common practices across squads for squads to use best practices and remain agile. While squads have end user of the system as their customers, chapters have squads as their customers. So Chapters are usually low value adding elements and so company should invest accordingly on these chapters.
- Volatility Based Architecture:
    - In his book "" Juvi Lowel explains concept of volatility based architecture. This style of architecture comprises of boxes that are quite independent of each other. Juvi Lowel talks about not architecting the system based on functional modules. This is because architecture is something that spans across functional modules. e.g. payments and cataloging modules both can have same architecture but these are different modules within the system. Having one architecture across all your modules with specific services for specific needs provides you with a simple system thats easy to handle in longer run.

- ## Investments in Automations

As a system gets larger, the risk of breaking something that's working becomes higher. Also, the amount of time needed to test the whole system with minor changes go out of proportion. Hence if leaders want to keep making small changes to the system continuously, investment in automations that reduce time to market would help immensely.

- **Test Automation:** As the workflow a system supports becomes wide and deep the number of use cases that need to be tested for every small change grows exponentially. There are various types of test automations that would help engineering leaders keep delivering small changes quickly with high quality.
    - **End to End Automations**: End to end automation suits deal with end user workflow level test cases. Ensuring with every small change none of these critical workflows are broken help engineering leaders deliver high quality fast. Even if teams are making changes everyday to the software, but if the full software is tested before all changes are delivered reduces time to manually test and validate the whole system yet deliver changes quickly out.

        - **Unit Test Automations**: These tests are written at more granular level such as for each class or function / methods. Embedding these test cases early in cycle protects the engineers making changes to these classes and function that would break something that's already working. Having such test cases surrounding these classes and methods allows engineers to make changes quickly without worrying about causing regression to existing functionality.
        - **Service Level Automations**: As engineering leaders separate out the system in different services and having test cases to validate these services whenever there is a change in the service would protect the system from any changes that can disrupt workflows that interact with changed set of services.
    - Deployment Automations
        - Implementing automated pipelines for building, testing, and deploying code.
    - **Infrastructure as Code (IaC):** Managing infrastructure through code for consistency and efficiency.
    - Refactor into Microservices
    - **Robotic Process Automation (RPA):** Automating repetitive tasks to free up human resources for higher-value activities.
    - **Slowly Refactor into Microservices :** A large system with complex dependencies between its module becomes hard to maintain. Microservices is an opinionated service oriented architecture. In a microservice design services are separated from each other and then these are enabled for independent deployment. Right size microservices such one microservice can be fully developed and managed by one scrum team and each team is not managing too many microservices. Also, communication between microservices should be clearly defined and clearly separated to avoid any further complexity such as changing one microservice has cascading effect on many other microservices.
- ## Establish culture and practices of continuous refactoring and continuous improvement
    - Arrive at a long term vision. Since large systems are catering to large number of users, its seldom possible to radically change the system and disrupt its use. Hence engineering leaders need to establish practices such as 'leave it better' to constantly evolve the system over a period of time. e.g. start building a microservice with few end points and then over a period of time keep moving more features to the new service while outdating the older one.
- ## Build Autonomous Self Sufficient Teams Building Innovation at Speed and high quality

Creating self-sufficient, cross-functional teams is essential for Agile success. Key strategies include:

- **Team Autonomy:** Granting teams the authority to make decisions and own their work.
- **Cross-Functional Capabilities:** Ensuring teams have the necessary skills to deliver complete features.
- **Agile Frameworks:** Adopting frameworks like Scrum, Kanban, or SAFe to provide structure and guidance.
- **Continuous Learning:** Fostering a culture of learning and development within teams.

Cultural Transformation

A strong Agile culture is essential for long-term success. Key focus areas include:

- **Collaboration and Communication:** Encouraging open communication and collaboration across teams.
- **Experimentation and Innovation:** Creating a safe environment for experimentation and learning.
- **Continuous Improvement:** Embracing a culture of continuous learning and adaptation.
- **Leadership Support:** Ensuring strong leadership commitment to Agile principles and values.
- Leave it better
- Continuous Refactoring and continuous improvements:

By deeply understanding and implementing these strategies, organizations can effectively scale Agile and achieve their business objectives.

Conclusion

Scaling Agile requires a combination of architectural improvements, automation, and team empowerment. By implementing these strategies and fostering a supportive organizational culture, organizations can overcome challenges and achieve greater agility, efficiency, and innovation.

Would you like to add more specific examples or case studies to illustrate these points?

Why did you choose Agile ? What is the outcome you prefer from lets say 10 teams working on a single product as Agile teams ?

Somewhere i feel that as the organizations and projects get large, they get lost in the mechanics of Agile. which is doing the scrum, daily standups, maintaining product backlogs etc. However, in essence teams are still delivering mega releases once in a month or once in two

months with thousands if not millions of lines of code getting delivered in this major releases. Somewhere in this process if you zoom out you will see that organizations have increased their quotient of waterfallishness in the process. The releases are still very big, these require multiple rounds of testing and towards the end of the release teams are really not working on new features. Also, teams have so high dependencies on each other that when there are issues found in the release it takes longer period of time to identify where the issues are and what caused those issues. Forget about ability to release the software to slowly to a large customer base and forget about ability to rollback changes. Its a big bang release with high impact and something Agile principles should have helped you to avoid getting into.

To me, the above scenario is not where you wanted to be and you expected Agile to help you in that. However, this did not become possible because even though your teams are following mecanics of agile, they are really not following the core principles of agile. They are truly not agile.

To me, Agility means being able to ship software changes in as small increments as possible. 'Ability' is the key word. Does not mean you release everything in small increments. The reason you created 10 scrum teams for 50 engineers should have been so that each team can deliver their own changes as and when they intend to. Rather than all 50 engineers having to wait for the big release and wait to learn from each release.

The purpose of Agile practices is to be able to release frequently in smaller increments and for each team to learn as quickly as possible from their experiments so that they can iterate over these features to make these better. Teams also need instrumentation in place where they can act like a startup. Which means teams can release a small feature to smaller set of customers before rolling it out to a larger set of customers.

To me, unless you are enabling these mechanisms in your software delivery process, you are not moving towards Agile. Your teams are only following the motions and not really truly practicing agile.

This means, there are some pre-requisite that you need to work upon before you can scale multiple teams working on the same product. This is the investment that you need to make in order to get to a state where you can deal at a larger scale for your product.

Pre-requisite to Scaling Agile for larger projects/products:

- High amount of DevOps as a culture and CICD Automations.
- Good level of identified separation of concern in your services
    - In a product in order to assign these to different teams for a longer period where they can work on part of these pieces of software as separate products.

Principle 1: What is your goal ?
While establishing multiple teams, organizations should ask these questions. Why are they establishing multiple teams or so called scrum teams. Why not just create one large team and let the whole team deliver what features they need.

Well the constraint here comes is that teams can only function when they are 2 pizza size teams. This is where humans can communicate effectively within this size and team meetings can be conducted effectively. This also gives a structure where you have one team lead for each team, a shared engineering manager and a dedicated product owner.

So now, that we know we cannot function as one large team we must break this down into smaller teams of two pizza team, how can we set boundaries of these teams so that these teams do not clash with each other quite often. This takes us to second principle.

Principle 2: Teams should be able to function as independently from each other as possible.

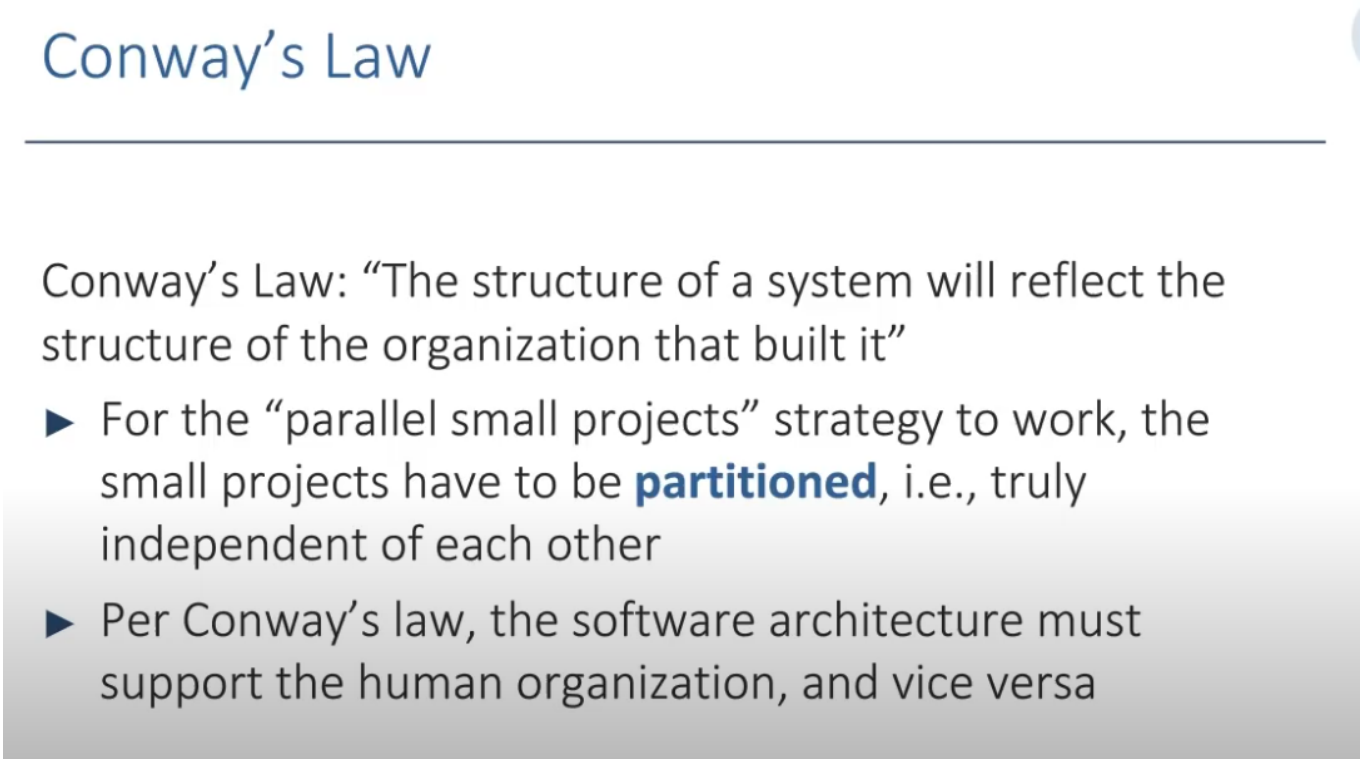
Key concepts: Autonomy.

- Autonomy:
  - Mini-Startups
- "How" Autonomy: (Team choose their own process)
  - When they run their standup
  - Which flavor of agile they use (some use kanban or scrum or they switch)
- Ownership
  - Dedicated product owner
  - Agile coach
  -
- Autonomy to Build what teams want to
  - Influence autonomy
  - Each person in a team is an idea person. Mutual Respect.
- Release autonomy
- A Mission
  - Every squad has a mission. They are building a product (a set of features): Who is benefited from these features and what are the benefits.
- Org Support
- Build and Improve over long period of time
- Metrics for measuring success:
  - Value is the only metric that you should depend upon
  - Think it -> Build it -> Ship it -> Tweak It
  - Blue Green Releases

    - Be data driven and success is based on data
- Moving from Squads to Tribes
    - < 50 People
    - Same mission across squads.
    - Co-located
    - scrum of scrum on demand
- Chapter
    - horizontal : squads.

https://www.youtube.com/watch?v=TkkfBLQFI2Q&t=208s

convay's law:



Conway's Law, Partitioning, and Agile Philosophy

- This requires a focus on **software architecture** before most of the human teams are staffed and organized—which is considered by many to be an unnatural fit with Agile projects

Agile Organizations resist getting into technical designs and architectures earlier in the cycle of projects while scaling challenge of agile requires someone to understand the architecture of the product.

Cocomo II Model: Its meant for estimation model.

Common Response to Challenges on Larger-Than-Usual Projects

- This is a natural response, but it doesn't work, because the factors that matter the most **change** at scale

What got you here won't get you there.

Success on small projects does not prepare organizations to succeed on large projects

"What got you here won't get you there"