## Introduction

These notes are based primarily on two very informative talks about scaling Agile in large organizations. I compiled these into a format for my own use. The notes are fairly high level and are no substitute for the full content of the videos, but might be a useful reference. Thanks to Jez and Mike for making the content available online.

Jez Humble CTO of devops-research.com talks about tools and concepts for delivering value and general LEAN / Agile concepts. <https://www.youtube.com/watch?v=2zYxWEZ0gYg>

Mike Cottmeyer CEO of leadingagile.com talks about Why Agile Fails in Large Enterprises - Large Scale Agile Transformation <https://www.youtube.com/watch?v=Oo3zlOTbN2E>

## Key concepts and thoughts:

Agile Manifesto highest principle: Satisfy the customer through early and continuous delivery of valuable software.

Must have understandable, measurable goals.

Sustainable development. Must be able to maintain constant pace indefinitely.

Assume we are not doing it the best way. Mindset of constant improvement.

Experimental approach to achieving the goal. **Hypothesis > Run experiment > Measure > Learn.** Use retrospectives to see how experiment worked. This is the scientific method applied to software. Repeat the cycle as quickly as possible to avoid wasted effort. Applies to product AND processes.

Detect problems as soon as possible: software defects, hypothesis / proposed changes with no value, operational malfunctions, etc.

Transparency. Create shared understanding among all involved.

## Determining what to work on:

Something that has measurable impact. Not highest paid person’s opinion (HPPO)

Focus on value not estimating cost of development. Top value impacts: Will it be cancelled? Will it be used? Try and determine usefulness as soon as possible.

Cost of Delay[[1]](#footnote-1) - How much does it cost to NOT deliver a feature? Used to prioritize possible work.

Minimal implementation (Minimal Viable Product-MVP) to verify hypothesis (**may NOT be code**). Want to get feedback ASAP to know if the feature / product is the right thing (or not).

Hypothesis driven delivery instead of user stories[[2]](#footnote-2):

We believe that

[building this feature]

[for these people ]

will achieve [ this outcome].

We will know we are successful when we see

[this signal from the market].

Impact Mapping: Work backwards from desired impact (**Who** could do **what** to produce impact. Which option is minimal cost?) Value of the exercise is the shared understanding by the team not the map itself.

Features may provide 0 or negative value: (3 ways)

* Lost opportunity cost (could be implementing something useful instead)
* Added complexity increases ongoing maintenance costs.
* Future new development is slower due to complexity.

User Experience (UX): Not about just making it look pretty. Includes whether or not to include features, how to implement them. May not require code implementation to test hypothesis.

Backlog items should meet the INVEST test:

Independent - Can go in any order, is easy to pull in/out and non-overlapping.

Negotiable – Does not include the details of features; just the essence

Valuable - to the customer and in their language.

Estimable – To help the customer prioritize and schedule.

Small – An understandable chunk of work to fit in sprint (less than a few weeks work).

Testable – You know how to write a test for it and when it is “done”.

## How to work:

Work in small batches. The build pipeline should make this very economical.

Experimental approach for product.

Experimental approach for processes.

Developers need to be able to recreate test failures.

Software should always be in a provably releasable state (daily).

Rotate people; share the pain.

Automate deployment and testing.

Continuous incremental improvement of process. Should be habitual.

Kata cycle:

What is the challenge?

What is current condition?

Establish next target condition (not how, but what does it look like).

Iterate toward target condition.

Deployment pipeline

If tests fail (system breaks), all check-ins stop except fix to the issue.

Every change is a build.

Every build is a release candidate.

Pipeline's job is to kill the build (show that it is not releasable).

If errors get through, fix the tests.

“Done”: Must meet acceptance criteria, no known defects, no technical debt. Must be able to maintain a sustainable pace.

Teams: 6-8 people, encapsulated piece of architecture, complete team to implement

QA role: Represents customer. Role is to make the system quality transparent (reveal it).

High Risk Changes: Always change in an additive way. Add toggles to enable / disable new functionality. Roll out in stages.

Example: Database change to split *addr* into *addr1* / *addr2*

Add new columns.

Application checks new columns. If null, read from old *addr* value, write only to new columns.

After confident that change is working, then bulk migrate rest of the data and clean up.

## How to implement Agile:

Need a sense of urgency for lasting change.

Needed for Agile: Backlogs, teams, working tested software.

* Quality Backlogs: INVEST criteria
* Teams: 6-8 people, encapsulated piece of architecture, complete team to implement
* Working tested software: meets acceptance criteria, no known defects, no technical debt.

Also Known As (AKA): Clarity, Team Accountability, and Measurable Progress. Matches with what developers want[[3]](#footnote-3):

* Purpose – do something that has real meaning
* Autonomy – freedom work their own way, decide for themselves
* Mastery – get better at what they do

When implementing in traditional organizations, provide proper structure first, then practices. Culture change takes time.

Large Scale implementation: (SAFe, other frameworks)

Governance - How to prioritize value with scarce resources.

Structure - How form teams and large-scale collaboration

Metrics & Tools - How to baseline and show improvement

When ther are many teams, you have to manage dependencies between sprints. SAFe - Mainly to plan / wrap dependencies.

Reduce orchestration overhead by increasing encapsulation so each team can work more independently.

Reduce overhead / planning by organization change.

Must adapt methods (e.g. SCRUM) to what works (be agile)

Migration to Agile steps for traditional organizations.

1. Stabilize the system. Create trust / predictability, form teams, create structure and processes to handle backlogs and dependencies, work within contracts and existing governance structure.
2. Reduce batch size (more frequent releases) [financial back ends / Systems of Record (SORs) may stop here]
3. Make refactoring / platform changes. Encapsulation. Break dependencies between components.
4. Team based project deliverables. (single product owner, team) Governance change
5. Fully decoupled team based funding (goal based development) Change innovation strategy

Start with front-end non-SOR systems

Teams hierarchy (small to large)- Services, Products, Programs, Governance

Services use SCRUM, others use Kanban/LEAN

Incremental implementation

Team to decompose requirements to backlog should include these representatives: Product, Architecture, Project Management, Analyst.

Types of companies:

|  |  |
| --- | --- |
| Predictive | Adaptive |
| Scheduled predictable long period deliveries | Small quick batches, change as needed |

|  |  |
| --- | --- |
| Emergent | Convergent |
| highly agile:build whatever to solve issue | risk / uncertainty 🡺 time / cost/ scope |

Four quadrants:

* Predictive / Emergent - Ad Hoc. Trying to adopt agile (to keep up)
* Predictive / Convergent – Traditional, but small batch size. Longer planning times and higher governance than regular Agile. (e.g. financial, SORs)
* Adaptive / Emergent - Lean Startup model
* Adaptive / Convergent - Agile.

Obstacles to Agile:

* Matrixed organizations - violate team forming
* Non-instantly available resources - who ask questions to?
* Too much WIP
* Limited SME access
* Large Diverse technology
* Shared requirements between teams
* Technical Debt
* Low Cohesion / Tight coupling

## Business case example: HP printer firmware.

[Transformation example from Jez’s talk.]

2 measurable goals: get firmware off critical path, 10X productivity increase

23% of time going to automated test development after the transformation. However, time spent on innovation increased from 5% to 40% due to eliminating waste in development process.

1. See blackswanfarming.com Maersk study [↑](#footnote-ref-1)
2. From Jeff Gothelf [↑](#footnote-ref-2)
3. Dan Pink *Drive* [↑](#footnote-ref-3)