Real-life Agile Scaling
Keynote - Agile Tour Bangkok
Nov 21, 2015

Henrik Kniberg
henrik.kniberg@crisp.se
@HenrikKniberg

Consultant
www.crisp.se

Parent

Agile & Lean coach

Author

Lean from the Trenches
Scrum and XP from the Trenches

Spotify
LEGO
skype
Not too hard
A bit trickier

Henrik Kniberg
Hard!
How do we avoid THIS?
Beware of Scaling
Beware of Scaling

Potential Downside (Risk):
• Longer delivery time because of misalignment & dependencies
• Worse product because of bad communication

Potential Upside:
• Shorter delivery time more hands on deck, parallel work
• Better product access to wider range of competences

Guaranteed Downside
• Cost
• Complexity

Henrik Kniberg
Stuff you need to figure out with multiple teams

How to slice the elephant

Team structure

Feedback loop

Team sync / alignment

Dependencies!
Slice the elephant
Big Bang = Big Risk

10% succeed
38% totally fail

Source: Chaos Manifesto 2013, Standish group

Cumulative Cost

Cumulative Value

Henrik Kniberg
Henrik Kniberg

≈250 people involved
4 years to first public release
Shut down after 2 years of operation

R I P
Lego Universe
Not like this....

Like this!

Henrik Kniberg
Slice the elephant!

Region
Östergötland, Uppsala, etc

Crime types
(weapon, drunk driving, shoplifting, etc)

Integrations

Henrik Kniberg
MVP – Minimum Viable Product

Henrik Kniberg
Minimum viable $\Rightarrow$ Earliest testable/usable/lovable

Earliest Testable Product

Earliest Usable Product

Earliest Lovable Product

Aim for the Clouds...
But deliver in Small Steps

Henrik Kniberg
Two types of slicing

1. Release 1.0
   - Slicing to enable early & frequent release

2. Release 1.1
   - Slicing to enable parallel development

Henrik Kniberg
Build a suitable Team Structure
Component team vs Feature team

Client team
Server team
DB team
Test team

Feature team 1
Feature team 2

User
Client
Server
DB

Communities of interest

Henrik Kniberg
Two conflicting goals (at scale):

1. Team should be “full-stack”
2. Team should be small
Team types - finding the right balance

100% feature teams

Trade-off

100% component teams

Small orgs

Large orgs

Henrik Kniberg
Types of dependencies

- **independent teams**
  - Knowledge sharing
  - EASY

- **Building different products, but have dependencies**
  - Knowledge sharing
  - Dependency sync
  - TRICKY

- **Building the same product (implicit dependency!)**
  - Knowledge sharing
  - Dependency sync
  - Product integration
  - HARD
Dependencies

Good Dependency (aka “collaboration”)

Bad Dependency

Henrik Kniberg
Example: Visualizing team dependencies

Henrik Kniberg & Jan Grape
Example:
Visualizing team dependencies

<table>
<thead>
<tr>
<th>Squad</th>
<th>Depends on</th>
<th>Dependency</th>
<th>Comment</th>
<th>Same tribe?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Music Player</td>
<td>Ops</td>
<td>Slowing</td>
<td>Need machines, connections, help set-up things etc.</td>
<td>No</td>
</tr>
<tr>
<td>Content</td>
<td>Next</td>
<td>No problem</td>
<td>Storage. Not big, mostly information/communication needs to happen</td>
<td>No</td>
</tr>
<tr>
<td>Content</td>
<td>BFS</td>
<td>No problem</td>
<td>Replacement service</td>
<td>Yes</td>
</tr>
<tr>
<td>Content</td>
<td>Team 2</td>
<td>No problem</td>
<td>Communication around next story</td>
<td>No</td>
</tr>
<tr>
<td>Content</td>
<td>Team 1</td>
<td>Future</td>
<td>Content ingestion</td>
<td>No</td>
</tr>
<tr>
<td>BFS</td>
<td>UX</td>
<td>Slow</td>
<td>Need UX to discuss, review and provide mock-ups</td>
<td>No</td>
</tr>
<tr>
<td>BFS</td>
<td>Content</td>
<td>No problem</td>
<td>Normal dependencies, sprint work</td>
<td>Yes</td>
</tr>
<tr>
<td>BFS</td>
<td>Mobile</td>
<td>Slow</td>
<td>No internal mobile developers within Squad</td>
<td>Yes</td>
</tr>
<tr>
<td>BFS</td>
<td>Analytics</td>
<td>Slow</td>
<td>A/B test results slowing down roll outs of features</td>
<td>No</td>
</tr>
<tr>
<td>BFS</td>
<td>Team 3</td>
<td>Slow</td>
<td>Waiting for data dumps</td>
<td>No</td>
</tr>
<tr>
<td>BFS</td>
<td>Team 1</td>
<td>Future</td>
<td>Waiting for data dumps</td>
<td>No</td>
</tr>
</tbody>
</table>

Henrik Kniberg
Good vs bad dependencies

**Full-stack team.**
Can deliver customer value independently.

**Coupled teams**
A must sync with B in order to deliver customer value.

**Platformized teams**
Team A1 and A2 are more effective because of team B’s platform.
Customer-driven platform teams

**External-facing teams**
Focus on delivering value to external customers

**Internal-facing teams**
Focus on making other teams more effective at delivering value to their customers.

The other teams are our customers!

The other teams must obey us!
Key decision:

Where can we accept low-bandwidth communication?

The speed of development is determined by the speed of ideas spreading.

Henrik Kniberg

Alistair Cockburn
Teams of Teams!

- Bunch of individuals
- Small teams
- Big teams
- Teams of small teams

Henrik Kniberg
Decoupling to enable frequent releases

Henrik Kniberg
Guidelines for team structure

Try to ensure that each team:

- is 3-9 people
- is stable(ish), full-time & co-located.
- has a mission
- has clear customers
- can prioritize between customers (ex: via a PO role, or via clear strategic guidelines)
- cross-functional: has all skills and tools needed to deliver value to customers
- autonomous: doesn’t get blocked waiting for other teams and individuals.
Scale the feedback loop

Henrik Kniberg
Single team feedback loops

- Daily Standup
- Retrospective
- Continuous Integration
- Unit tests
- Sprint review
Multiteam feedback loops

Cross team sync, retro, etc

Whole Product review
Pattern: Integration Cadence

Henrik Kniberg
Pattern: 2-tier planning/alignment

Henrik Kniberg
Pattern: Plan on a cadence, release on demand
Continuous Integration = Mandatory!

Continuous Delivery = Aspirational.

- Build
- Test & Integrate
- Deploy to Staging
- Deploy to Prod

Automatic

Manual Test

Code & Commit

Single Click

Henrik Kniberg
Get everyone aligned
Misaligned teams move very slowly
More teams = more likely that you will need dedicated leader(s)

Henrik Kniberg
Example: Leadership "Trios"

Tech P Product D Design

Henrik Kniberg
Alignment & Autonomy

False dichotomy

Do what I say!

Do whatever
Alignment enables Autonomy

High Alignment

Authoritative organization
Conformist culture

Micromanaging organization
Indifferent culture

Entrepreneurial organization
Chaotic culture

Low Alignment

Innovative organization
Collaborative culture

We need to cross the river
Build a bridge!

We need to cross the river
Figure out how!

Hope someone is working on the river problem...

We need to cross the river

Aligned Autonomy!
Example:
Big-room planning/alignment at Lego
Planning as a social event

Henrik Kniberg
2 days, 19 teams, 150 people

Henrik Kniberg
Lightning talks

High level priorities:
1. ...
2. ....
3. ....

Architecture vision / priorities / constraints

<table>
<thead>
<tr>
<th>Mission</th>
<th>Inspire and develop the builders of tomorrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspiration</td>
<td>Globalize and innovate the LEGO system-in-play</td>
</tr>
<tr>
<td>Promises</td>
<td>Play Promise (Joy of building, Pride of creation)</td>
</tr>
<tr>
<td>Planet Promise</td>
<td>Positive impact</td>
</tr>
<tr>
<td>Spirit</td>
<td>Only the best is good enough</td>
</tr>
<tr>
<td>Values</td>
<td>Imagination - Creativity - Fun - Learning - Caring - Quality</td>
</tr>
</tbody>
</table>

Digital Child Safety

Global Insights

Data Privacy Law

Henrik Kniberg
Pattern: Different levels of granularity

Program/Product backlog

Feature/Epic
- Marketable
- Releasable

Team Backlog
- Story
  - Testable
  - Fits in a sprint

Team Backlog
- Story
  - Testable
  - Fits in a sprint

Henrik Kniberg
Team breakout: Pulling from the program backlog

Henrik Kniberg
Team breakout: Pulling from the program backlog (digital version)
Team breakouts

Law of 2 feet....

Henrik Kniberg
Program Board
(a.k.a. Dependency Board)

Henrik Kniberg
Early detection of dependency problems
Scrum of Scrums = dependency sync
Simpler version of dependency sync

Dependency board
"right now, who’s waiting for what from whom"

Henrik Kniberg
Risk board
(per project/epic)

DIGITAL EXPERIENCE PLATFORM

RISK / IMPEDIMENT
- Relocation not done in time for market expansion
- Currently unstable infrastructure

RESOLVED
- Has been addressed, no longer a problem

OWNED
- Someone has taken responsibility for this issue

ACCEPTED
- Will take our chances

MITIGATED
- We have a plan for how to deal with the issue (limit and manage)

Henrik Kniberg
Management review / problem solving

**Resolved**
- has been addressed, no longer a problem

**Owned**
- someone has taken responsibility for this (who?)

**Accepted**
- we'll take our chances

**Mitigated**
- we have a plan for how to deal with this (reduce the likelihood or damage)

Henrik Kniberg
Management feedback & commitment to help

Henrik Kniberg
Pattern: Information radiators

- In the hallway
- Or in a War Room Zen Room
Big Picture - features/epics

Team 1 - stories

Team 2 - stories

Team 3 - stories

Henrik Kniberg
Example: Measuring velocity by counting cards

Count cards

Velocity per week

<table>
<thead>
<tr>
<th>Weeks</th>
<th>v10</th>
<th>v11</th>
<th>v12</th>
<th>v13</th>
<th>v14</th>
<th>v15</th>
<th>v16</th>
<th>v17</th>
<th>v18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Henrik Kniberg
Example: Release planning using a burnup chart

All of these will be done

Some of these will be done, but not all

None of these will be done
Wrapup
Don’t go overboard with Agile!

Bad Agile: No plan, No architecture

Good Agile: Rough, adaptive plan, Rough, adaptive architecture

Waterfall: Big up front plan, Big up front architecture

Henrik Kniberg
Stuff you need to figure out with multiple teams

How to slice the elephant

Team structure

Feedback loop

Team sync / alignment

Dependencies!

Henrik Kniberg
Real-life agile scaling – take aways

- **Scaling hurts**
  Keep things as small as possible

- **Agile is a means, not a goal**
  Don’t go Agile Jihad. Don’t dump old practices that work

- **There is no “right” or “wrong” way**
  Just tradeoffs

- **There is no one-size-fits-all**
  But plenty of good practices

- **Build feedback loops at all levels**
  Gives you better products and a self-improving organization.

Henrik Kniberg