Thinking between the frameworks

Mateusz Nowak

21.11.2019

code::dive

Who am I

Software Engineer, Team Leader @ habana

Located in Gdańsk, Poland

Works on: Enabling AI libraries on HW

Interested in: C++, team work organization, management, composing and recording music

Table of contents

All views expressed herein are those of my own and do not represent the opinions of any entity whatsoever with which I have been, am now, or will be affiliated.

Why frameworks?

Risks using frameworks

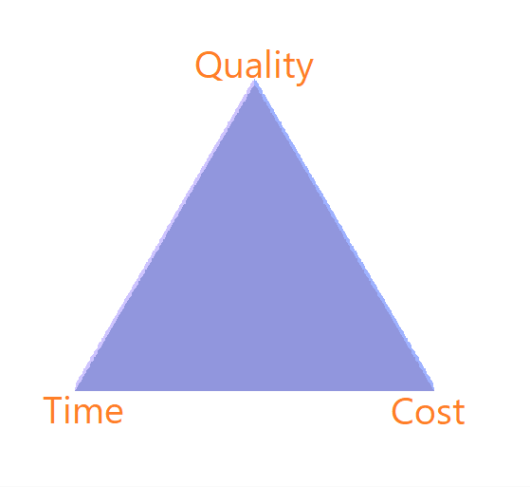
Executing framework instead of delivering value

Waiting too long for change

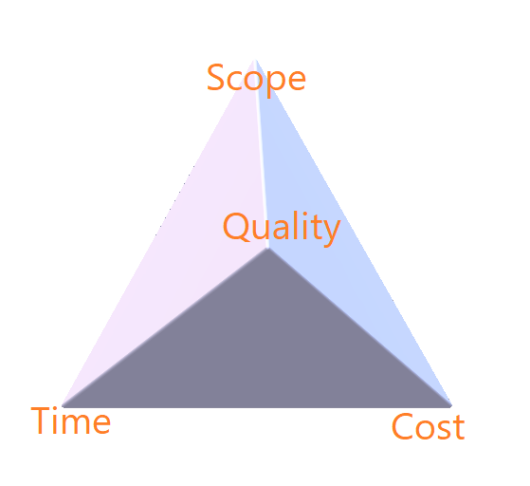
Getting lazy ;)

Back to the basics

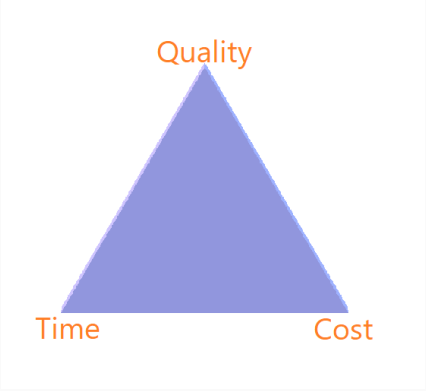
Project iron triangle



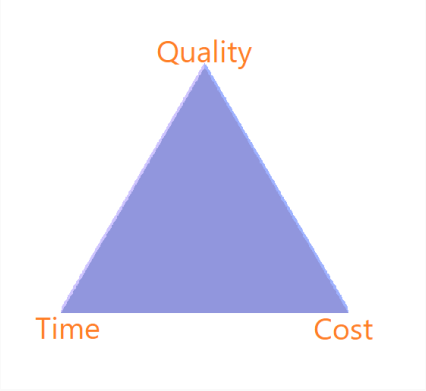
Project iron tetrahedron



Project iron triangle



Project iron triangle



$Scope \subset Quality$

Optimal team size

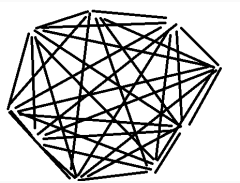
5 - 7 people

Optimal team size

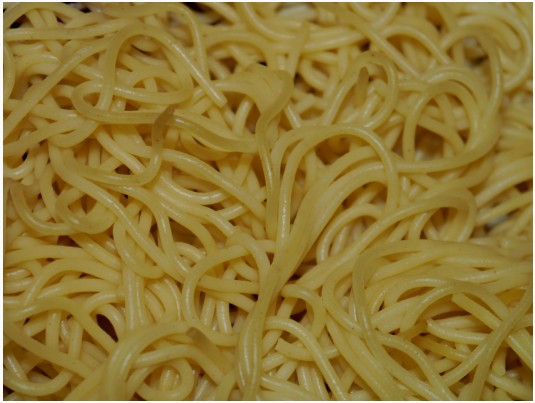
5 - 7 people

Too large:

- Risk of creating silos
- Too many interactions (n!)
- Team members may loose track who does what
- Fuzzy responsibility

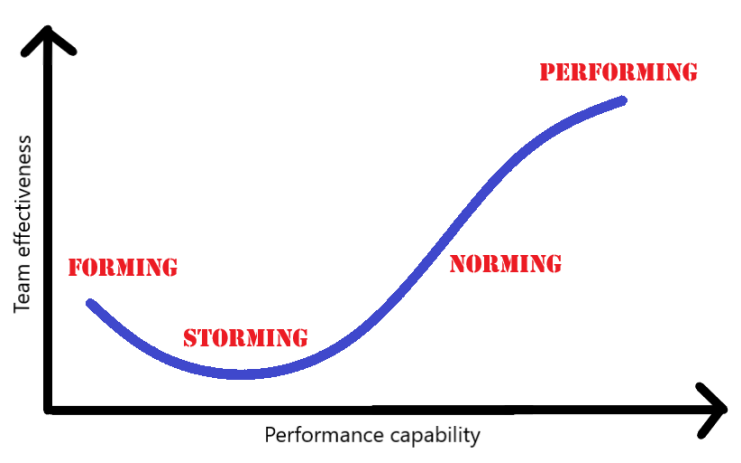


Too flat organization structure is too flat

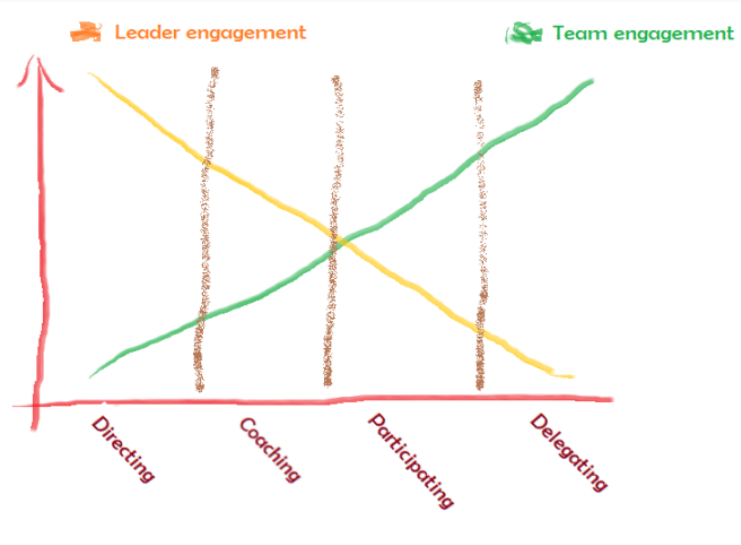


Source: https://pxhere.com/en/photo/757081

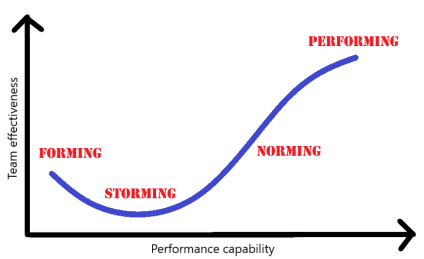
Tuckman model

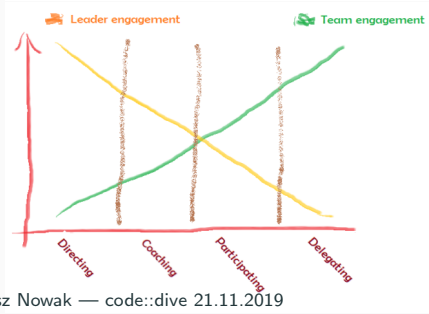


Four management styles



Four management styles vs Tuckman model





Kaizen

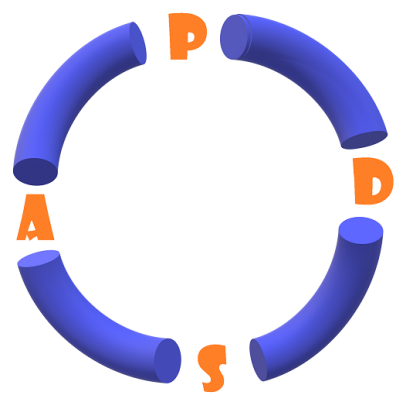
Continous improvement

Involves everyone in the organization

Mainly targeting processes

To eliminate waste

Deming cycle



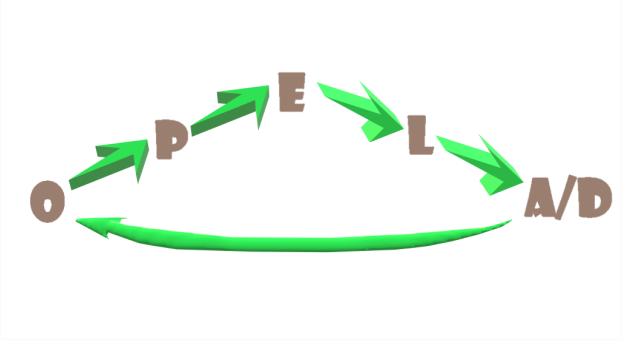
Plan

Do

Study

Act

Active improvement cycle



Observe

Plan

Experiment

Learn and store results

Apply or Discard

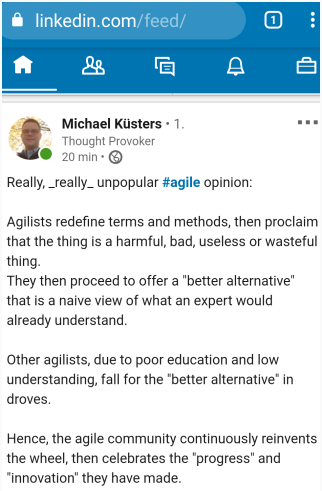
Everyone shall learn and experiment

Coffee meetings

Mixing frameworks

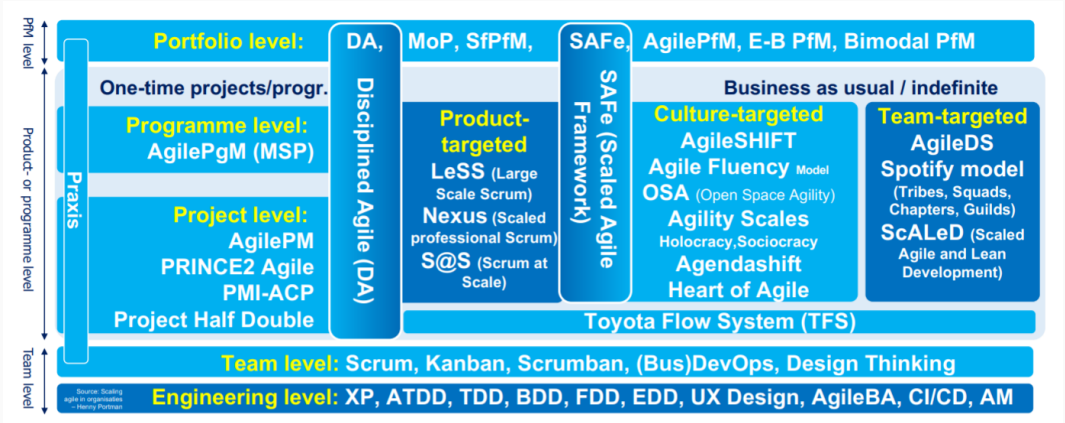
Solid base

- Daily stand-up
- Several commited people



https://www.linkedin.com/posts/michaelkuesters_agile-activity-6602538509729320960-iCir 19.11.2019

Agile forest



Source https://hennyportman.files.wordpress.com/2019/10/a-birds-eye-view-on-the-agile-forest-v1.9.pdf

Lessons learned log

Lessons learned log

* Shall exist from the beginning of the project.

Lessons learned log

* Shall exist from the beginning of the project.
* If does not exist shall be formal product for first sprint or project pre-work.

Lessons learned log

* Shall exist from the beginning of the project.
* If does not exist shall be formal product for first sprint or project pre-work.
* Shall be updated as the project progress.

Lessons learned log

* Shall exist from the beginning of the project.
* If does not exist shall be formal product for first sprint or project pre-work.
* Shall be updated as the project progress.
* Shall be available for other teams / groups / future projects.

Lessons learned log

* Shall exist from the beginning of the project.
* If does not exist shall be formal product for first sprint or project pre-work.
* Shall be updated as the project progress.
* Shall be available for other teams / groups / future projects.
* Never allow for the 'secret knowledge' to exist in the organization.

Exception report

When something unusual happen

When there is change needed

Know your stakeholders

Prepare a register of stakeholders

Know who is responsible for what

Update it continously

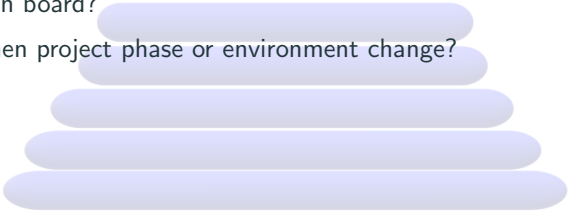
CI/CD from day 0

Solid ground for quality verification

Always be ready to describe what you already have

Determine how to plan work

- Maybe sprints + planning poker or equivalent?
- Maybe stage planning as in Prince2?
- Maybe use Kanban board?
- Maybe change when project phase or environment change?



Kanbanizing Scrum

Start step: "The Kanban Guide for Scrum Teams"

`https://www.scrum.org/resources/kanban-guide-scrum-teams`

Step two: Discarding some Scrum elements

Scrumming kanban

Add retrospectives

Add backlog

Use "sprints"

Using waterfall

For discovery / research tasks

For initialization phase

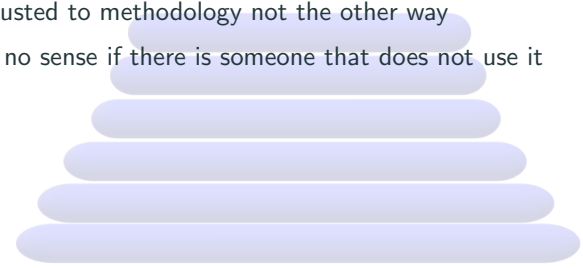
When in "directing" mode

When tasks are so hard trying to iterate would be too distracting

Select and commit to tracking tool

Tool shall be adjusted to methodology not the other way

Using tool make no sense if there is someone that does not use it



Adjust approach to the case

Mutable PBS

Clear visibility how product is constructed

Architecture documents are not the same

Set of user stories is not the same

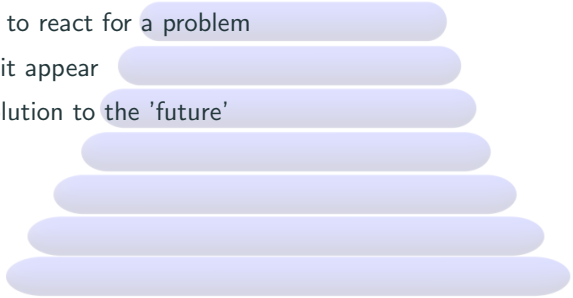
Shall evolve as constraints and requirements change

Retrospective shall be everyday habit

Sprint is too long to react for a problem

Raise an issue as it appear

Do not delay resolution to the 'future'



Everyone shall learn and experiment !

Everyone shall share knowledge and experiences !

Adjust approach to the specific case.

Thanks for your attention!

Mateusz Nowak

https://www.linkedin.com/in/mateusz-nowak-qq/

https://github.com/noqqaqq

Feedback is appreciated ☺