DAT257

Lecture 4: Building the product

Jonas Petrén

Welcome! We will begin in a few minutes

Jonas Petrén

10 years in software testing, development and as test manager. **6+ years of Scrum Master experience.** Full-time Scrum Master since beginning of 2018. Employed by HiQ since 2012.

Civilingenjör Informationsteknologi, Linköping University, 2003-2008

Today: Senior Scrum Master/Agile coach

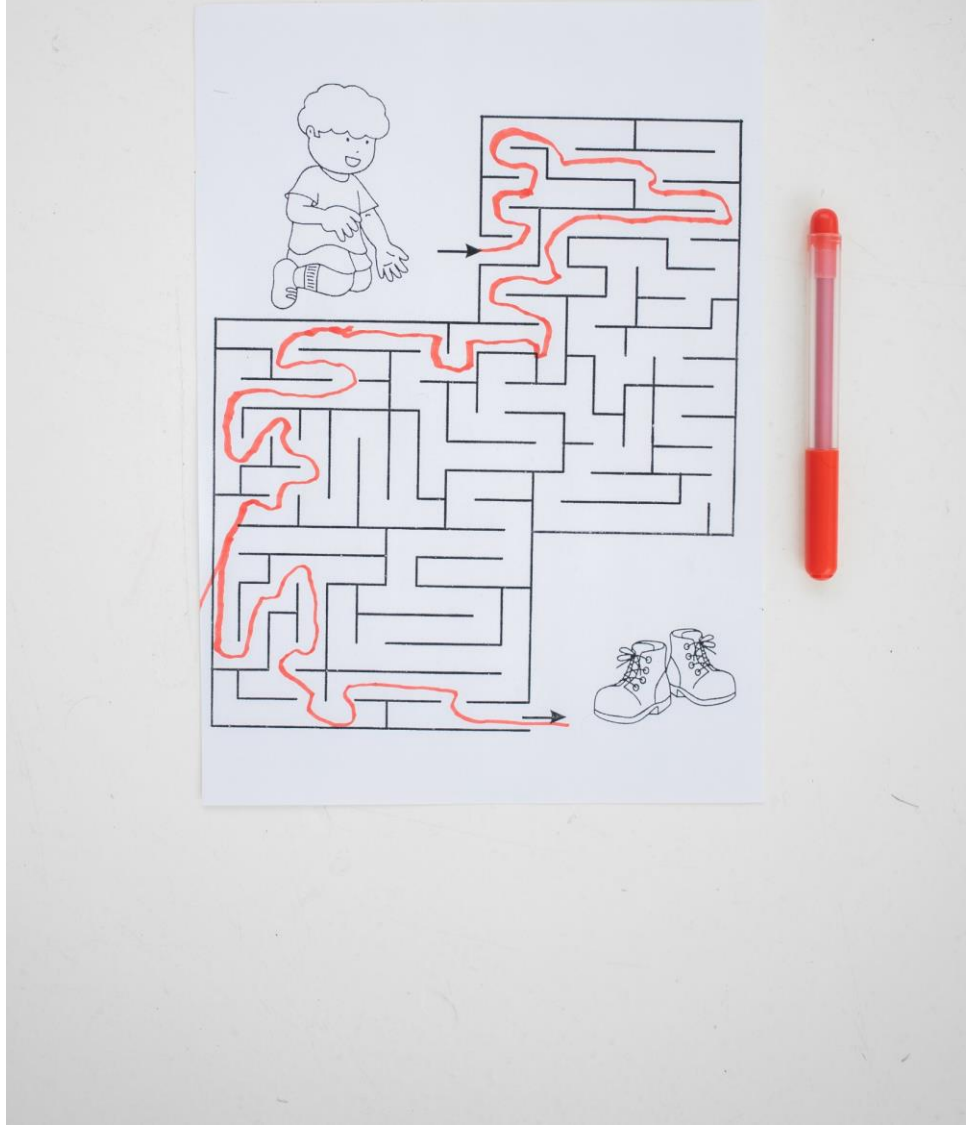
Certified **Professional Scrum Master III**

Agenda

- Incremental and iterative delivery
  - Output Outcome Impact
  - Minimum Viable Product
  - Product discovery
  - The Founder
- How to break down work
- Scaled Scrum
- Ending

The goal for today

Learn how agile teams build products



OUTPUT, OUTCOME, IMPACT

Very similar words, easy to misunderstand

Output, outcome and impact

Output is what you are producing e.g. code, documents...

May lead to

Outcome is the **EFFECT** of your output, something that happens *because of it*

May lead to

Impact are long-term or indirect effects that you **WISH TO REACH**

You want to **MINIMIZE OUTPUT**, but **MAXIMIZE OUTCOME AND IMPACT**

Output, outcome and impact

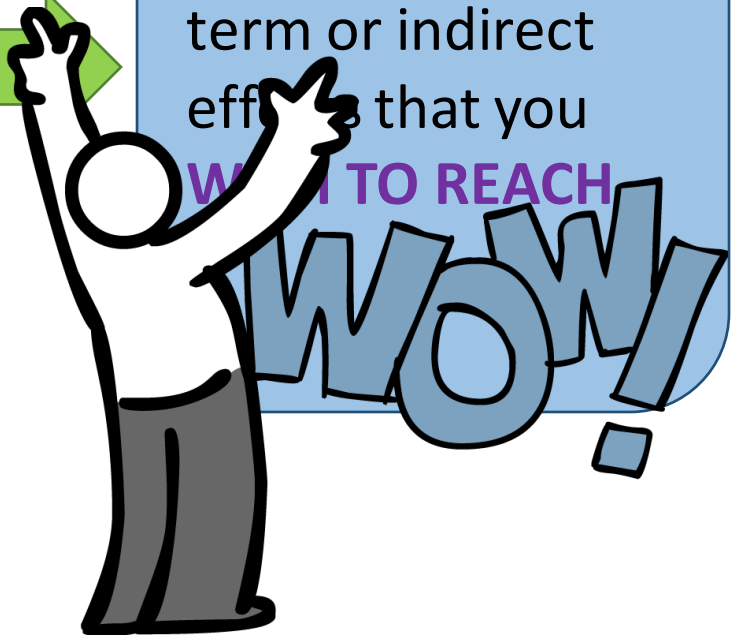
Output is the cost i.e. the price you pay to build your product

May lead to

Outcome is the **EFFECT** of your output, something that happens *just after*

May lead to

Impact are long-term or indirect effects that you **WANT TO REACH**



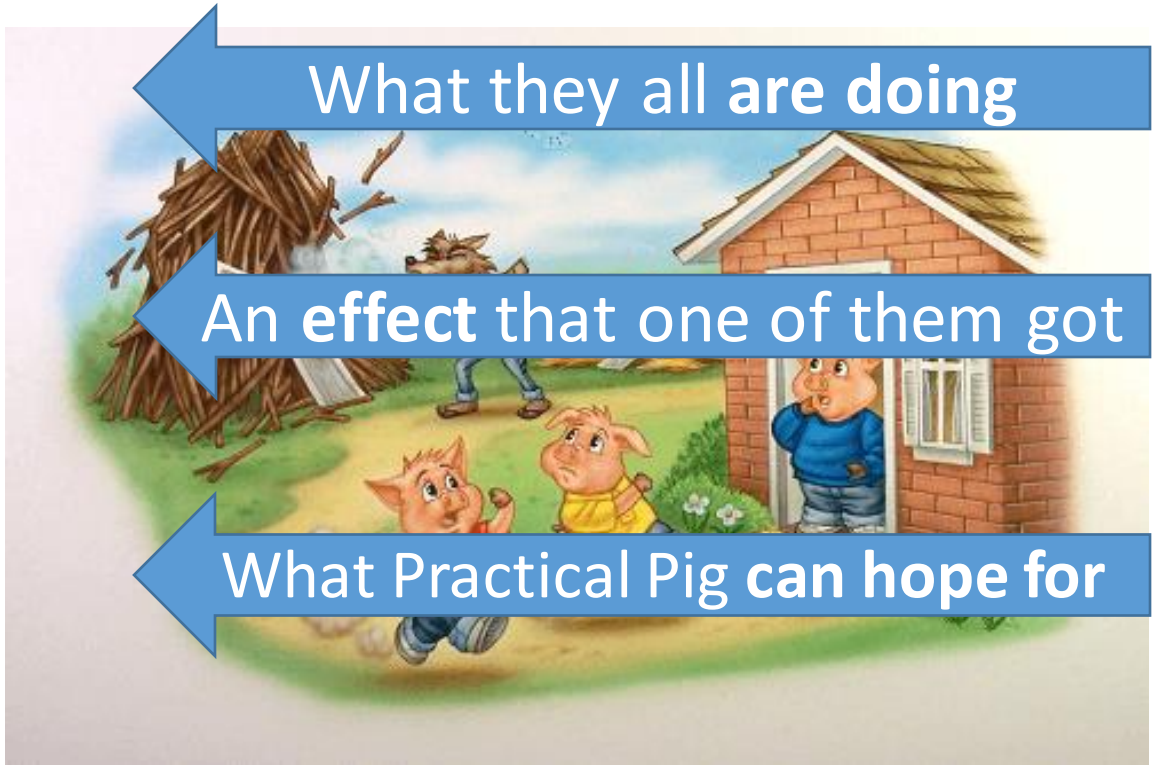
You want to **MINIMIZE OUTPUT**, but **MAXIMIZE OUTCOME AND IMPACT**

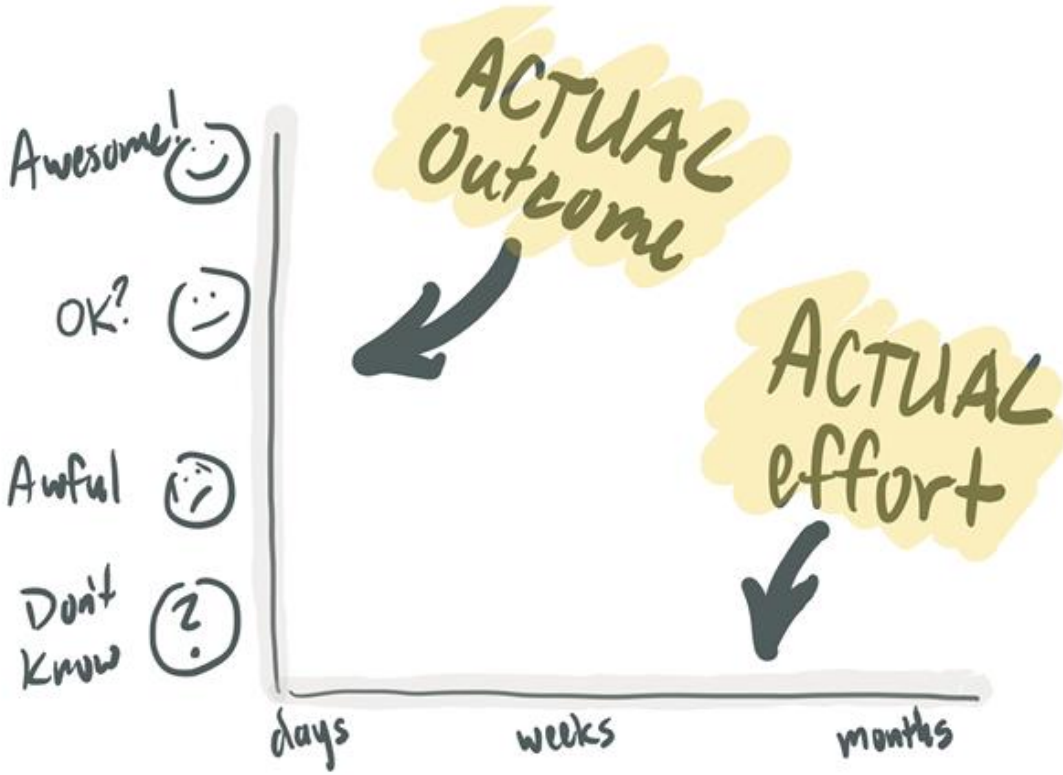
Output, outcome och impact

OUTPUT – All pigs have built a house

OUTCOME – Practical Pig is safe from the wolf because he built a solid house

IMPACT – Practical Pig gets popular among the female pigs and can plan for a life with wife and children





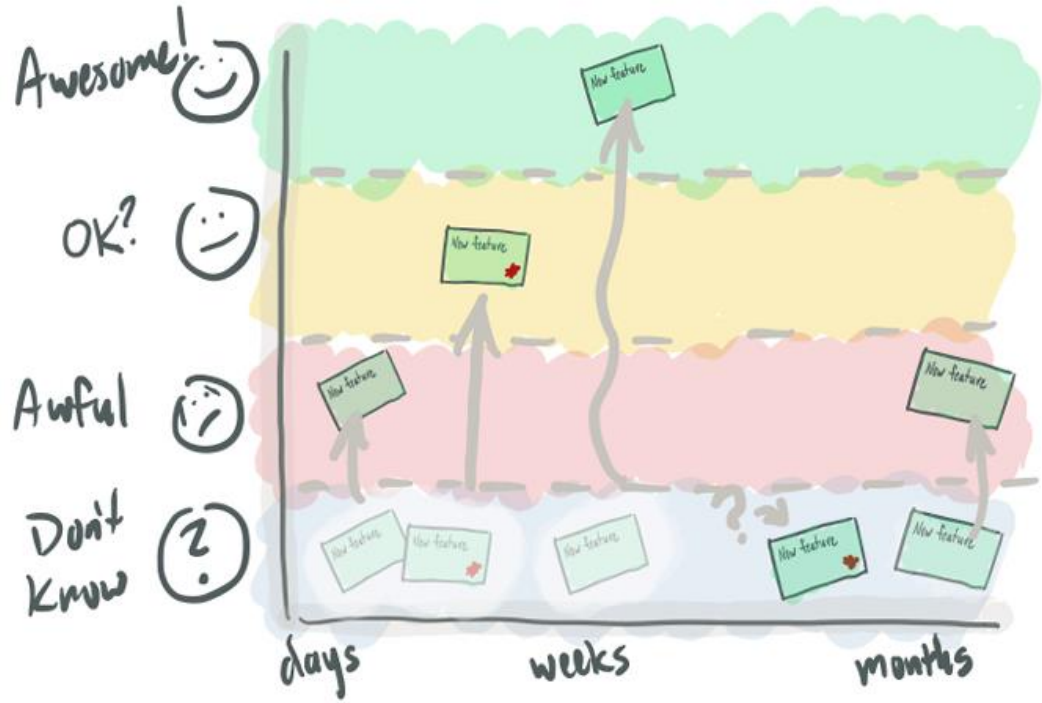


Image: Jeff Patton

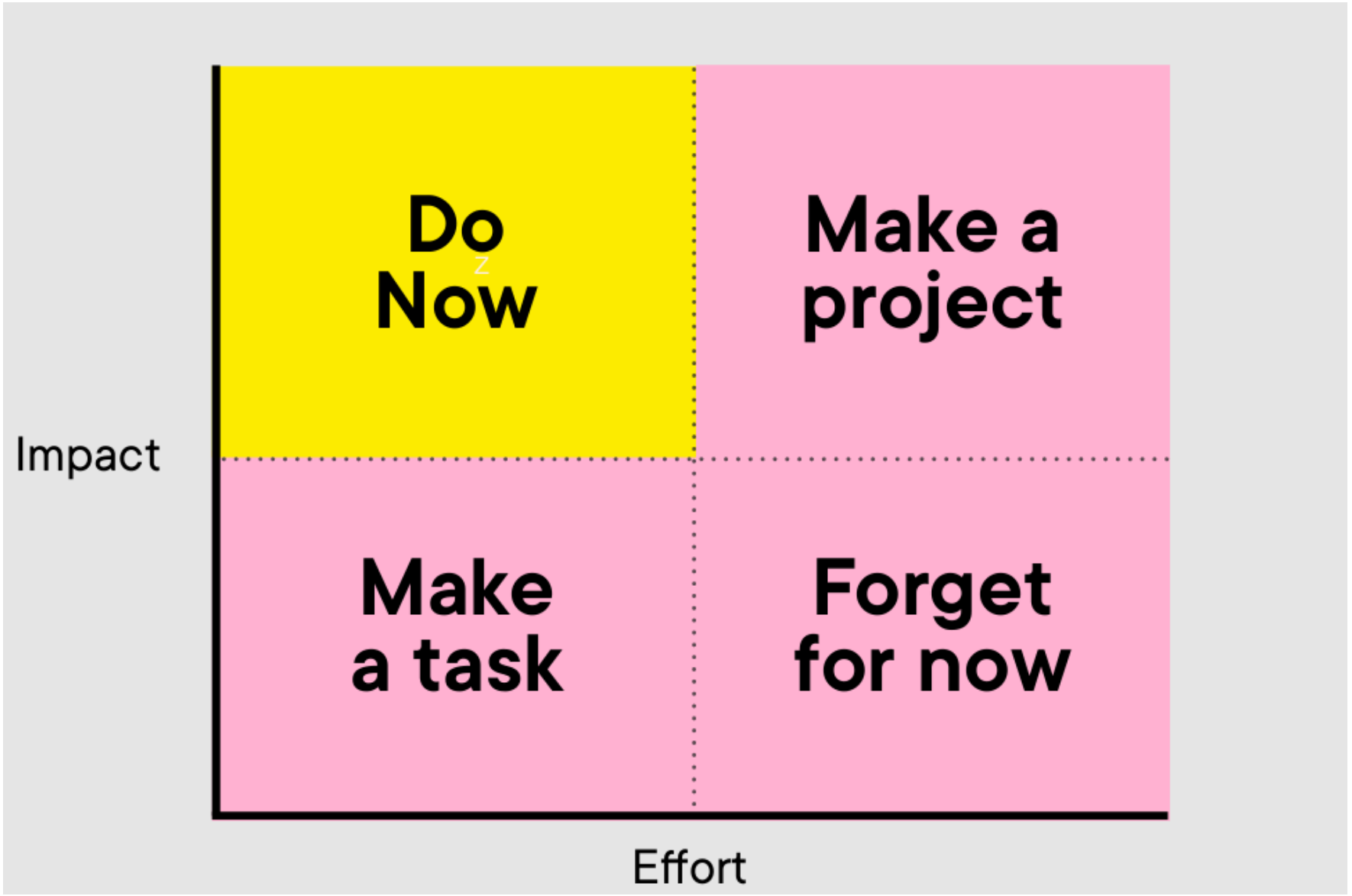


Image: Christer Hedberg

Going to work, day in and day out:

- **Output**: coaching Scrum, helping agile teams...
- **Outcome**: making something meaningful for others
- **Impact**: by doing a good job I hope to be getting more and more exciting assignments

Evening class in French

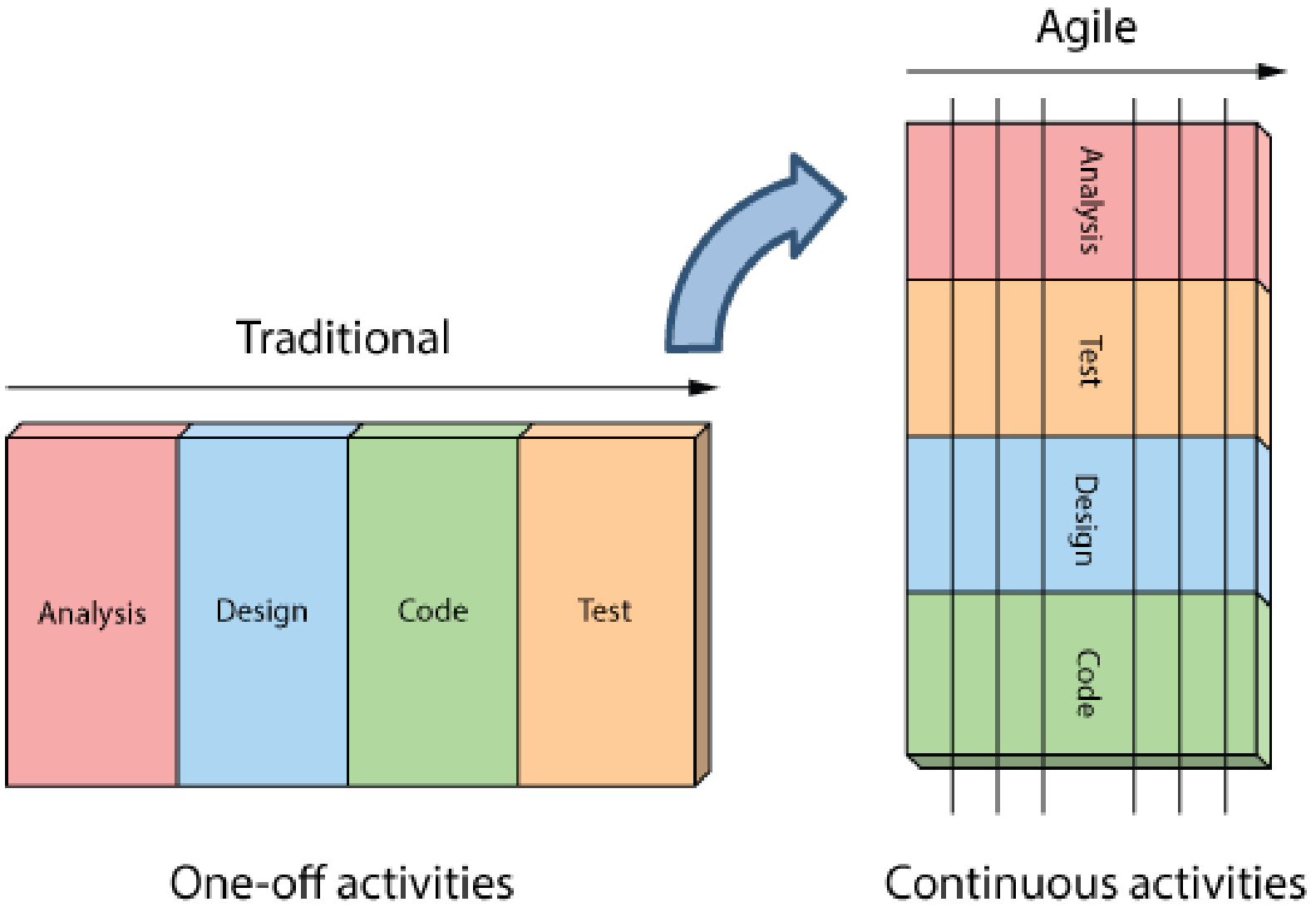
- **Output**: learning words and phrases
- **Outcome**: being able to travel in France by speaking only French

Teaching this course

- **Output**: slides
- **Outcome**: that you learn agile methodology
- **Impact**: when we meet in a future workplace you remember me as someone that made a good job

INCREMENTAL AND ITERATIVE

Very similar words, easy to misunderstand


Agile
Traditional
Analysis
Design
Code
Test
Analysis
Test
Design
Code
One-off activities
Continuous activities

Incremental and iterative

Incremental Development is when each successive version of a product is usable, and each builds upon the previous version by adding user-visible functionality.

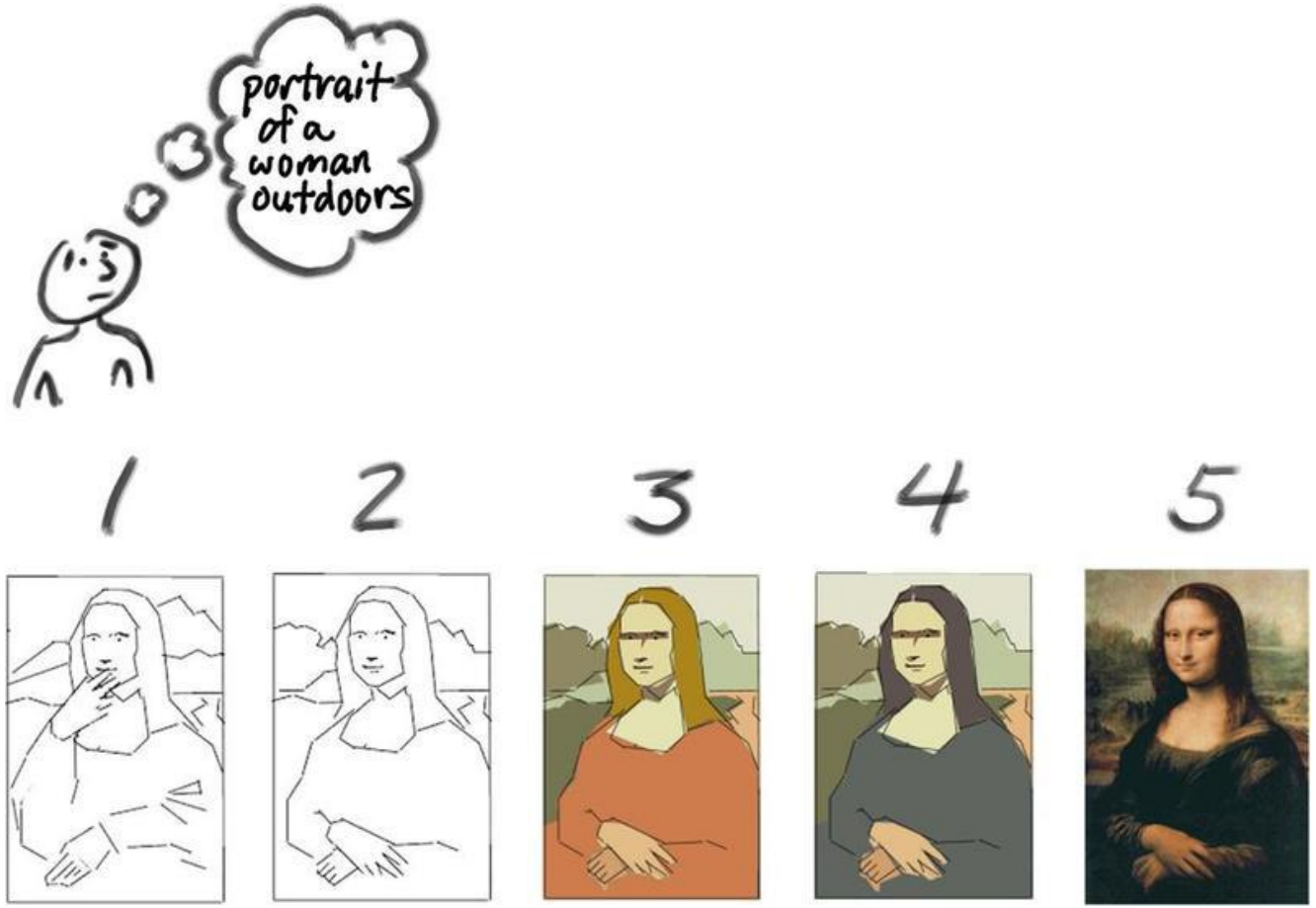
Agile projects are **iterative** insofar as they intentionally allow for “repeating” software development activities, and for potentially “revisiting” the same work products

Agile Glossary and Terminology | Agile Alliance

Don’t do this

This is a trap, because it says *“nothing is done before everything is done”*!

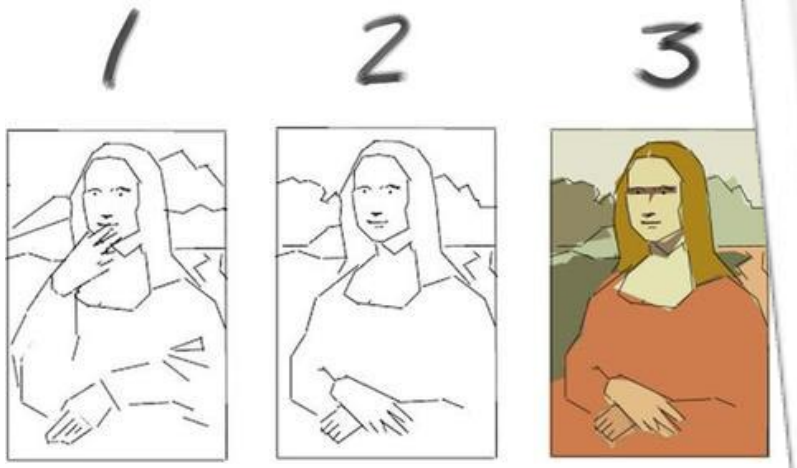
Do this!

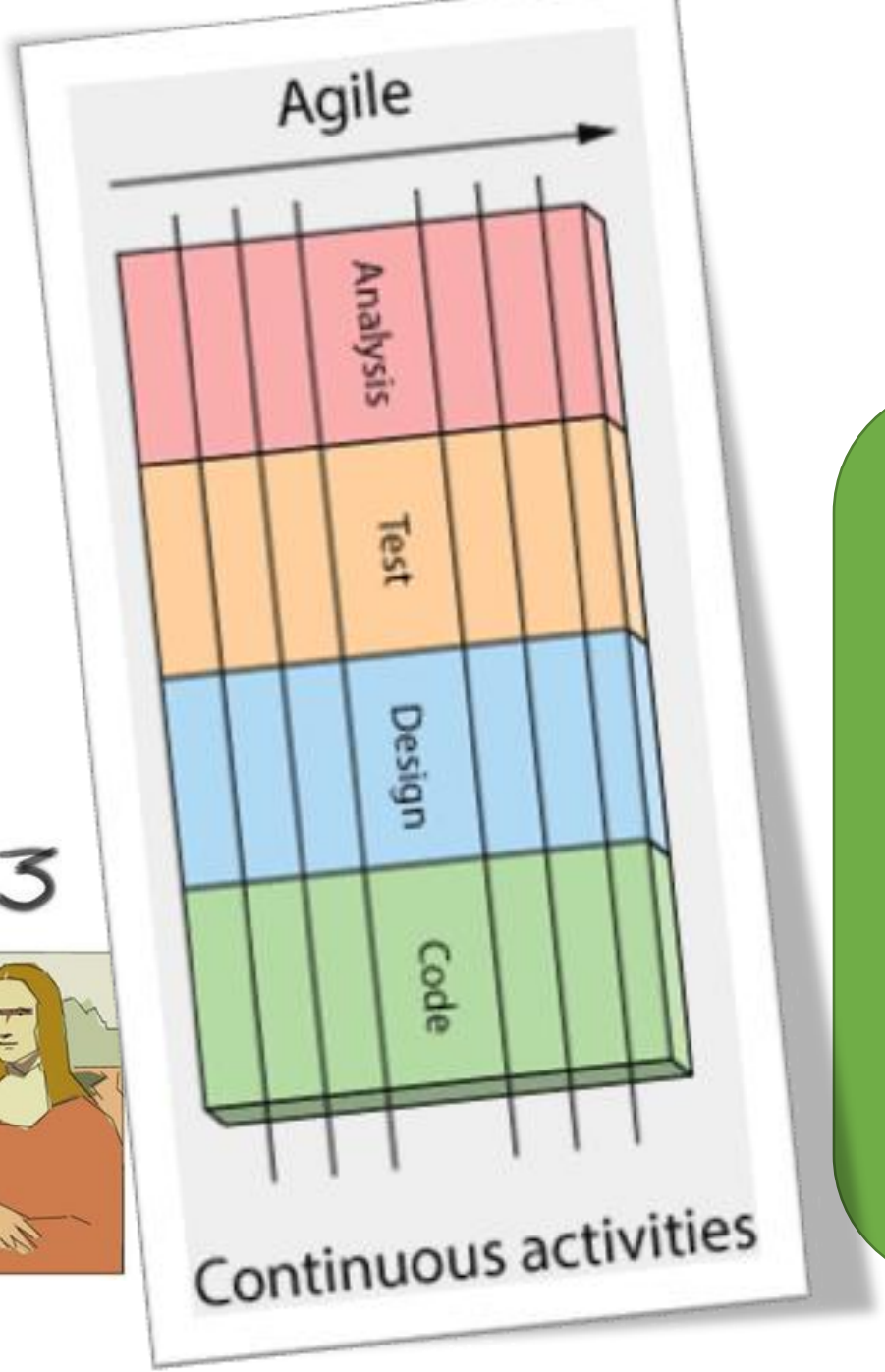


The customer may quit at any point and still have something valuable.

The customer decides if he/she would like to spend more money for a new layer

Do this!





The customer may quit at any point and still have something valuable.

The customer decides if he/she would like to spend more money for a new layer

Work like da Vinci to finish on time

When managing a release budget, split larger stories into "opening game," "mid game" and "ending game" stories.

Try to get the "big picture" as soon as possible. Early versions that are fully formed but immature allow early functional and performance testing. They allow earlier validation that your concept is right.

Development Strategy

Try using a story map to slice a feature into opening, mid, and end-game parts.

Image: Jeff Patton

Discussion 2 minutes

If you have decided on a project. How can you get "the big picture" early. What's your first prototype?


Two storey
Store
School
One storey house
Bus
Bus
Office building
Hospital

PRODUCT DISCOVERY

How do we know what to build?

Old days

- An idea was put in a roadmap. Planning how long it would take. Allocating time and resources
- The idea would get funding
- A team would start working on it
- Still working...
- Still working...
- Releasing to users... The team starts working on next initiative
- Perhaps getting feedback

Modern days – Product discovery

1. What **problems** are we solving, and for **who**?
2. What will customers and users **value**?
3. What can users **use** to reach their goals?
4. What's **feasible** to build given the tools and time we have?

- Develop a small solution, an early prototype
- Try it we beta testers. Observe them and interview them to get their feedback
- Refine your product. Improve it a little. Repeat

minutes ago. Each new story we create software to support is something new.

One of the luminaries in the Agile development community is my aforementioned friend, Alistair Cockburn, who once told me, "For every story you write, you need to put three into your backlog of stories."

I asked him why, and he said, "You just do."

I asked, "What should I write on the other two?"

"It doesn't matter what you write."

"What do you mean?" I asked, "I have to write *something* on them!"

Alistair replied, "Well, if you have to write something on them, then write what you want on the first card, and on the second card write 'Fix the first card.' Then on the third card, write 'Fix the second one.' If you aren't going around this cycle three times for each story, you're not learning."

In a traditional process, learning gets referred to as *scope creep* or *bad requirements*. In an Agile process, learning is the purpose. You'll need to plan on learning from everything you build. And you'll need to plan on being wrong a fair bit of the time.

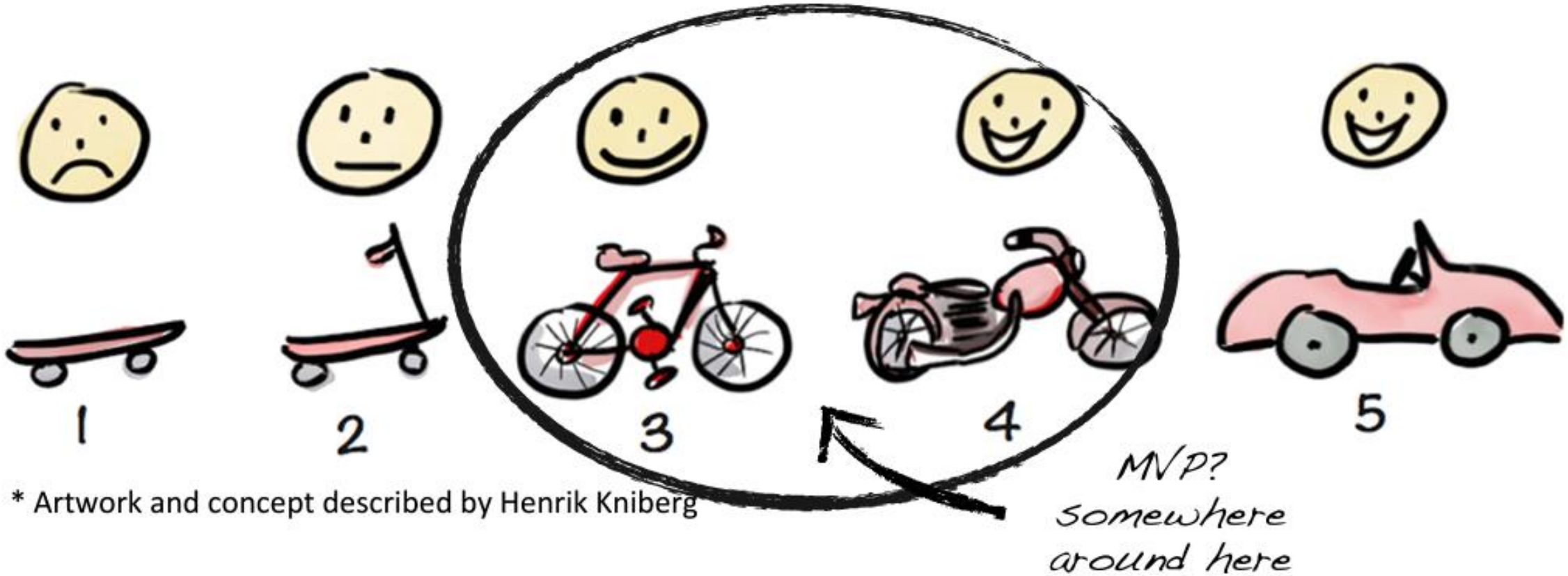
Jeff Patton – User story mapping (book)

Discovery team

A product owner leads the discovery team that includes individuals that have the knowledge and skills to identify a valuable, usable, and feasible product.

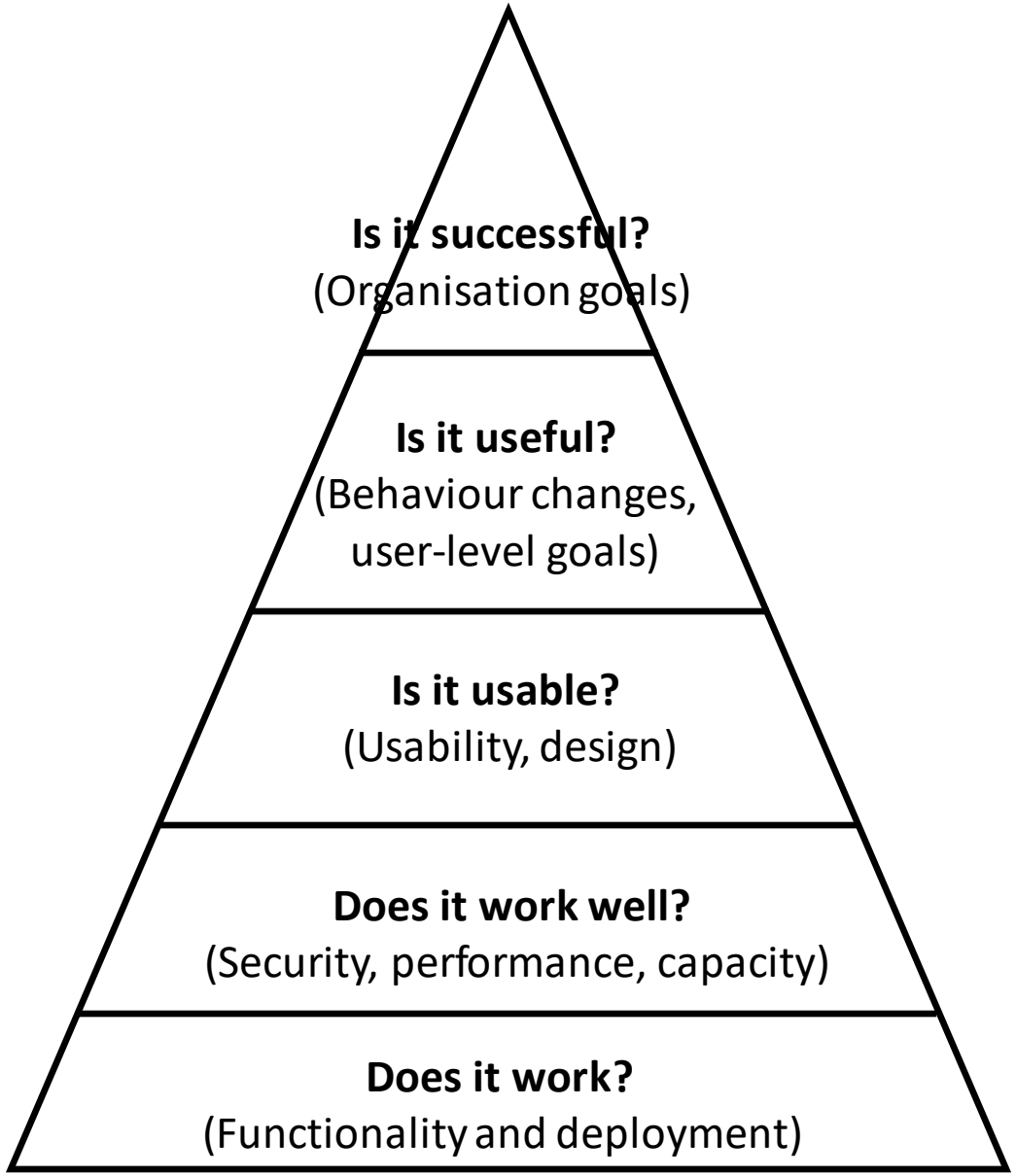
The Founder (movie):

- Working with a strong vision: *"Orders ready in thirty seconds, not thirty minutes"*
- Customer focus
- Prototyping
- Feedback
- Enabling flow
- Continuous improvements



* Artwork and concept described by Henrik Kniberg

Image: Jeff Patton and Henrik Kniberg


Is it successful?
(Organisation goals)
Is it useful?
(Behaviour changes, user-level goals)
Is it usable?
(Usability, design)
Does it work well?
(Security, performance, capacity)
Does it work?
(Functionality and deployment)

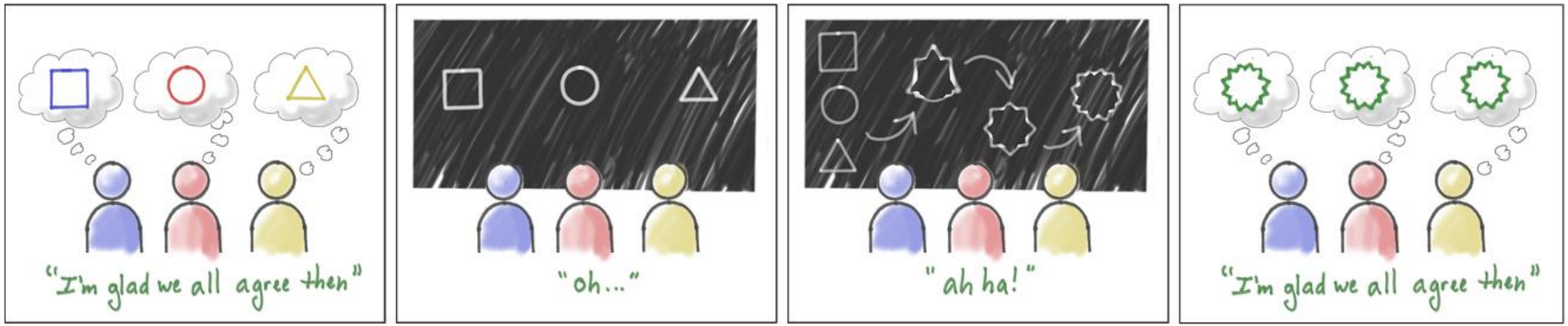


Image: Jeff Patton – User story mapping (book)

To break down work

Explained in 15 examples

How to think when breaking down work

What is the smallest piece of working functionality that we can give to a user? This is the first user story.

Gather feedback and identify the next stories

Example – roles

<u>Situation</u>

A feature should be added
(admin, basic user, su

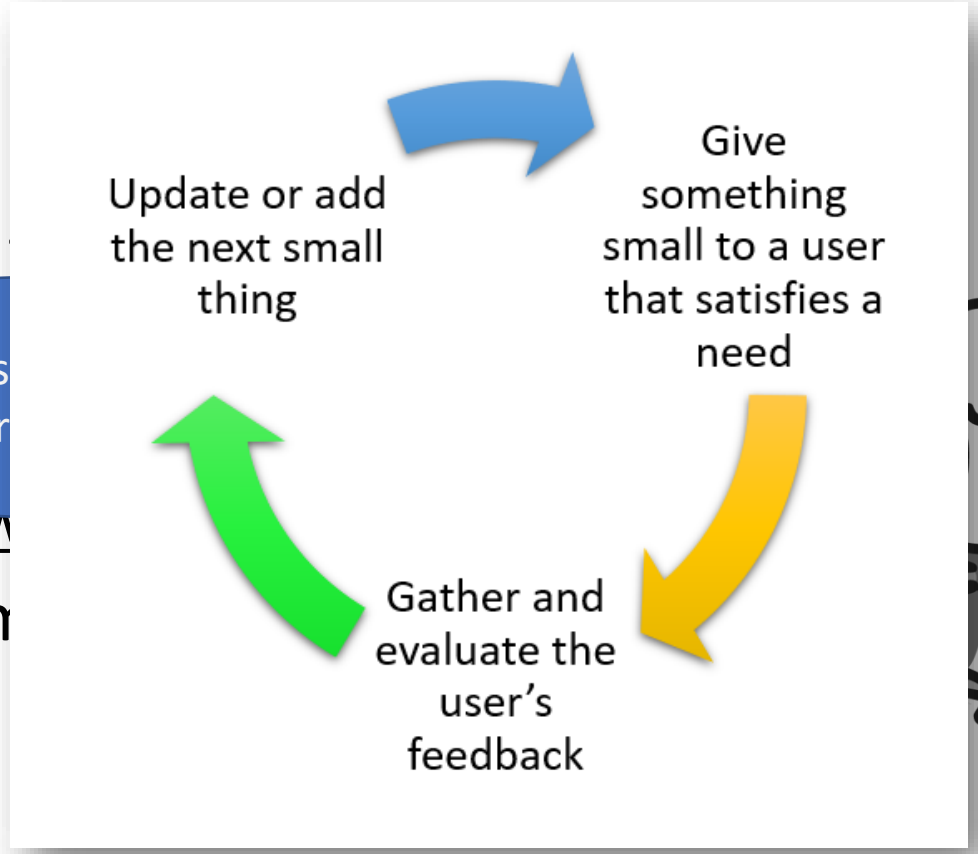
This is
user

<u>What's the smallest thin</u>

Implement one role at a tim

<u>Then what?</u>

Implement the other roles, one at a time



These are the next user stories

Example – user interface

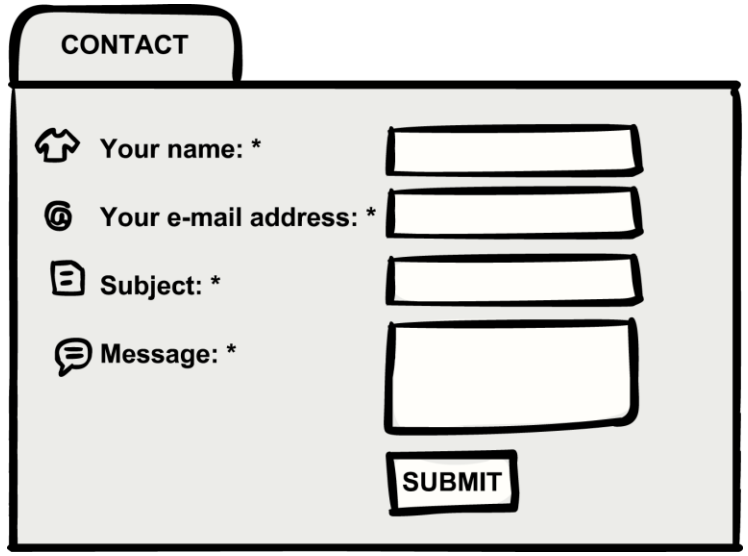
Situation

We need a user interface

One of these is the first user story!

What's the smallest thi… to a user?

Start with a text based UI (terminal window) *or* do a prototype on paper



Then what?

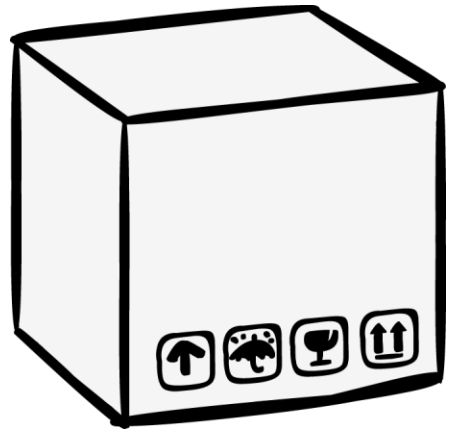
Then create a graphical UI

This is the next user story!

Example – new feature including error handling

Situation

A new feature should be created, including error handling



What's the smallest thing we can give to a user?

Make a "happy path" implementation first i.e. where everything goes right

Then what?

Add error handling, stepwise

Example - automation

Situation

A feature should work in an automated flow/sequence

What's the smallest thing we can give to a user?

Make it work manually

Then what?

Make a step-by-step instruction list on how to perform the manual sequence

Make it work automatically

Integrate it to a build pipeline, if desired

Example – multi-user scenario

Situation

A feature should work in a multi-user scenario

What's the smallest thing we can give to a user?

Make it work in single-user scenario

Then what?

Implement support for multi-user

Example – operating systems

Situation

A new feature should work on several operating systems

What's the smallest thing we can give to a user?

Make it work on one operating system (this could/should probably be broken down to tasks)

Then what?

Then make it work on the others, one at a time

Example – new technology domain

Situation

We are facing a new technology domain (i.e. going from monolith to microservices, changing from one vendor to another e.g. from Azure to Amazon...)

What's the smallest thing we can give to a user?

Learn as much as needed to provide a decision material to hand to PO/manager/architect/stakeholder *and/or* create a proof-of-concept

Then what?

Create one or more solution proposals, broken down to well-defined steps

Implement the first step

Implement the rest

Example – generic cases

Situation

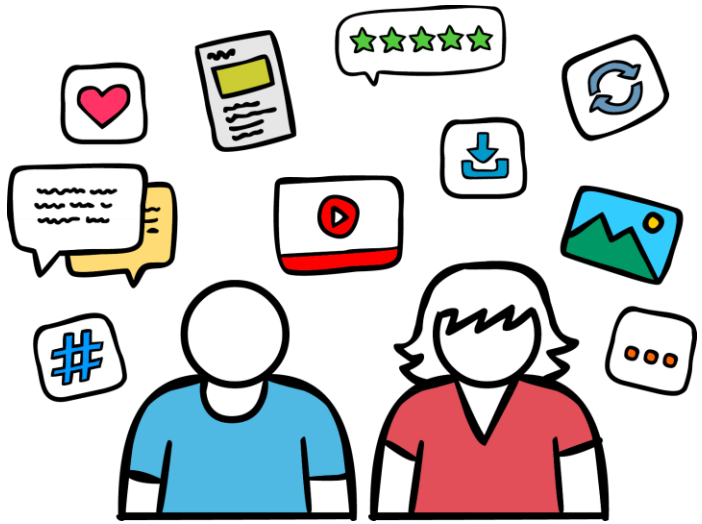
A feature should work in generic cases

What's the smallest thing we can give to a user?

Make it work in a basic case first

Then what?

Add support for generic cases



Examples - scalability

Situation

Something should work in big scale e.g. large data volumes, many concurrent users or total number of users in the system...

What's the smallest thing we can give to a user?

Make it work in small scale

Then what?

Scale it, gradually

Examples - bugs

Situation

Troubleshooting a bug

What's the smallest thing we can give to a user?

Can we reproduce the behaviour? Then ask, is it a bug or works as designed? This is a good way to catch misunderstandings.

Then what?

1. Propose a workaround, this could well be enough
2. Implement a solution

Examples - documentation

Situation

We should create documentation/a specification

What's the smallest thing we can give to a user?

Make a draft (outline important chapters or write the abstract first)

Then what?

Take each chapter as separate tasks

Example - dynamic

Situation

Something should work dynamically

What's the smallest thing we can give to a user?

Make it work hardcoded

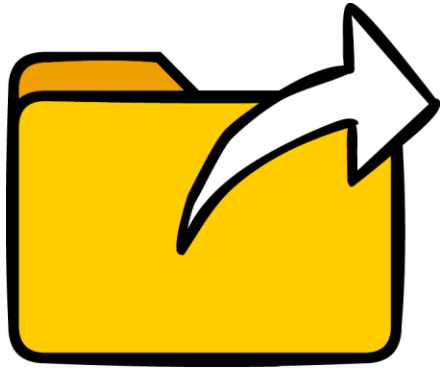
Then what?

Then make it dynamically, and don't forget to remove hardcoded values

Example - export

Situation

To be able to export data to several formats (Excel, pdf, ...)



What's the smallest thing we can give to a user?

The first user story is to process the data and create the simplest output, e.g. a text file

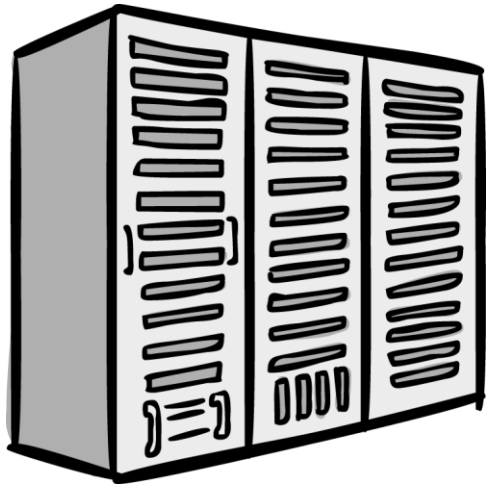
Then what?

Add support for more formats

Example – store data persistently

Situation

Data should be stored persistently



What's the smallest thing we can give to a user?

Make it work with transient information (i.e. That isn't saved when the application is closed)

Then what?

Store the data persistently

Example - size

Situation

A task feels too big

What's the smallest thing we can give to a user?

Find the biggest risk or uncertainty

Then what?

When the biggest risk is under control, handle the next risk and so on. Iterate like this until you have an overview of the total scope

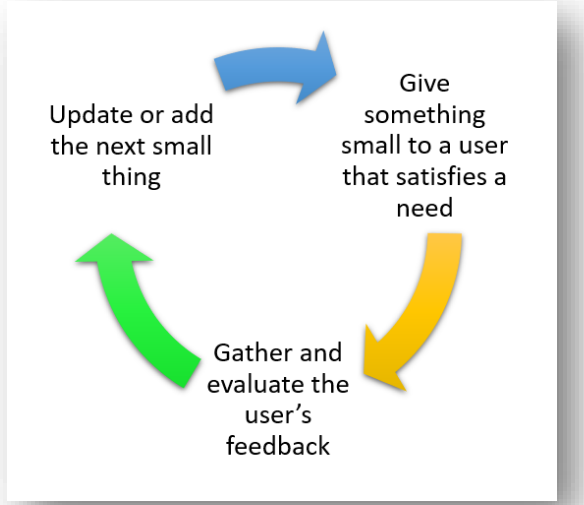
How to find the small pieces to do first

START SMALL

- Make a solution on paper
- Make a hardcoded solution
- Create a draft for a document
- **Split by functionality:** Make a solution for one role, file format, operating system… at a time
- **Split by capacity:** Make a solution work for e.g. a single user, small data volumes, only for transient information… before scaling capacity

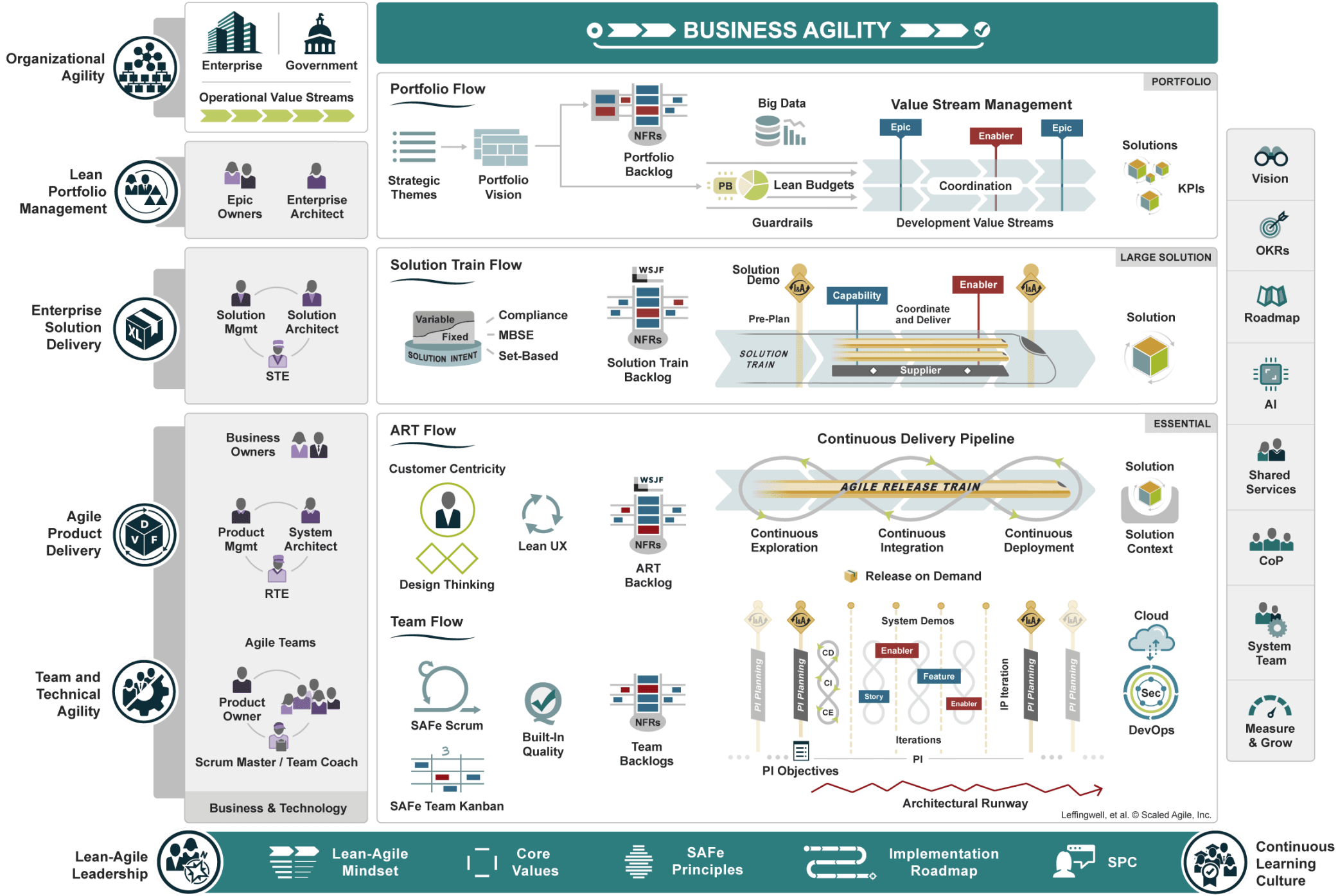
START HAPPY AND RISK-FOCUSED

- Implement the happy path first
- Focus on the biggest risk first
- Reproduce bugs and propose workaround before implementing a solution
- Learn about new technologies, then do a proof-of-concept before making a full transition from one thing to another



Scaling Scrum

When you have more than one team working on the same product


BUSINESS AGILITY
Organizational Agility
Enterprise
Government
Operational Value Streams
Lean Portfolio Management
Epic Owners
Enterprise Architect
Enterprise Solution Delivery
Solution Mgmt
Solution Architect
STE
Agile Product Delivery
Business Owners
Product Mgmt
System Architect
RTE
Team and Technical Agility
Agile Teams
Product Owner
Scrum Master / Team Coach
Business & Technology
PORTFOLIO
Portfolio Flow
Strategic Themes
Portfolio Vision
NFRs
Portfolio Backlog
Big Data
PB
Lean Budgets
Guardrails
Value Stream Management
Epic
Enabler
Epic
Coordination
Development Value Streams
Solutions
KPIs
LARGE SOLUTION
Solution Train Flow
Variable
Fixed
SOLUTION INTENT
Compliance
MBSE
Set-Based
WSJF
NFRs
Solution Train Backlog
Solution Demo
Capability
Enabler
Pre-Plan
Coordinate and Deliver
SOLUTION TRAIN
Supplier
Solution
ESSENTIAL
ART Flow
Customer Centricity
Design Thinking
Lean UX
WSJF
NFRs
ART Backlog
Continuous Delivery Pipeline
AGILE RELEASE TRAIN
Continuous Exploration
Continuous Integration
Continuous Deployment
Release on Demand
Solution
Solution Context
Team Flow
SAFe Scrum
SAFe Team Kanban
Built-In Quality
NFRs
Team Backlogs
PI Planning
CD
CI
CE
System Demos
Enabler
Feature
Story
Enabler
IP Iteration
Iterations
PI
PI Objectives
Architectural Runway
Cloud
Sec
DevOps
Vision
OKRs
Roadmap
AI
Shared Services
CoP
System Team
Measure & Grow
Lean-Agile Leadership
Lean-Agile Mindset
Core Values
SAFe Principles
Implementation Roadmap
SPC
Continuous Learning Culture
Leffingwell, et al. © Scaled Agile, Inc.

Takeaway: Scaling agile adds so much more complexity that need to be taken care of

Key takeaways

Key takeaway, 1 of 3

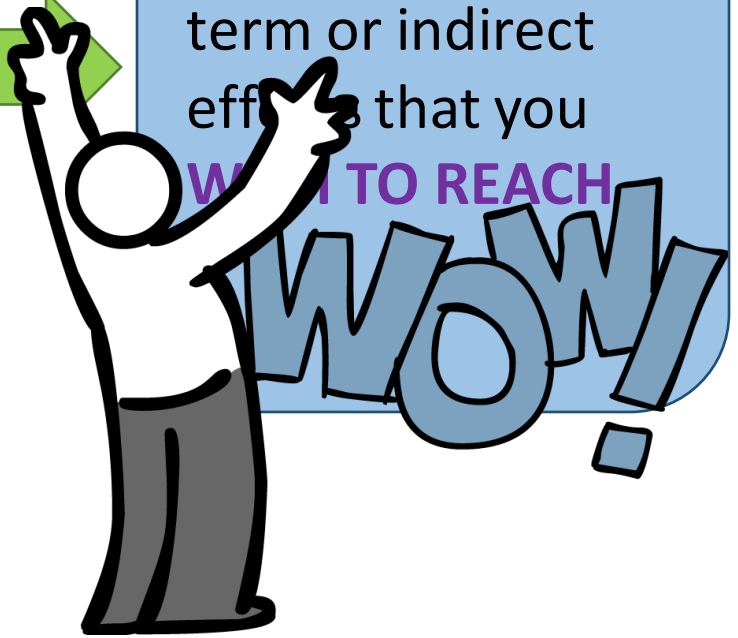
Output is the cost i.e. the price you pay to build your product

May lead to

Outcome is the **EFFECT** of your output, something that happens *just because*

May lead

Impact are long-term or indirect effects that you **WANT TO REACH**



You want to **MINIMIZE OUTPUT**, but **MAXIMIZE OUTCOME AND IMPACT**

Key takeaway, 2 of 3

Think in small pieces

Do a prototype early

Deliver often

Get feedback

Improve

Key takeaway, 3 of 3

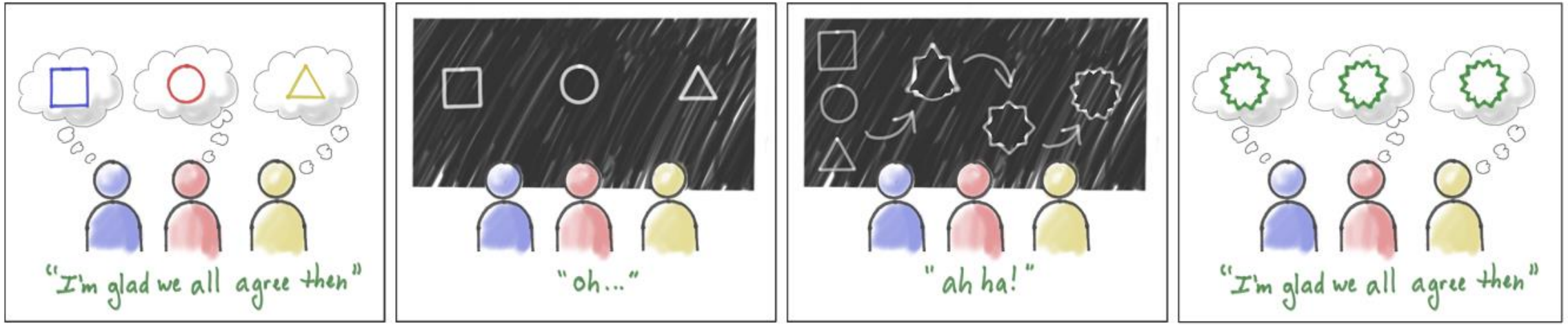


Image: Jeff Patton – User story mapping (book)

Talk and draw a lot in your teams. Make sure everyone is onboard on what you are doing


Thea Schukken
www.beeldinwerking.nl