Improving our Agile ‘Way of Working’

A description of a real team's experience, their principles, practices and antipatterns, using ESSENCE cards.

Version 3.0 2 Dec 2020

Kim Horn

Summary

Building software is hard and complex #.

Building software products is even harder.

Building these as an efficient team requires thoughtful continual improvement and reflection.

Building software is a unique endeavour, more like making a movie or writing a book, than engineering a bridge. However, like making movies there are essential engineering aspects*, that when mastered, open the craft to higher levels of creativity and purposeful control.

We can improve the “Way We Work”, the maturity in this endeavour, with simple continual engineered change, focusing on improving the team's quality of life.

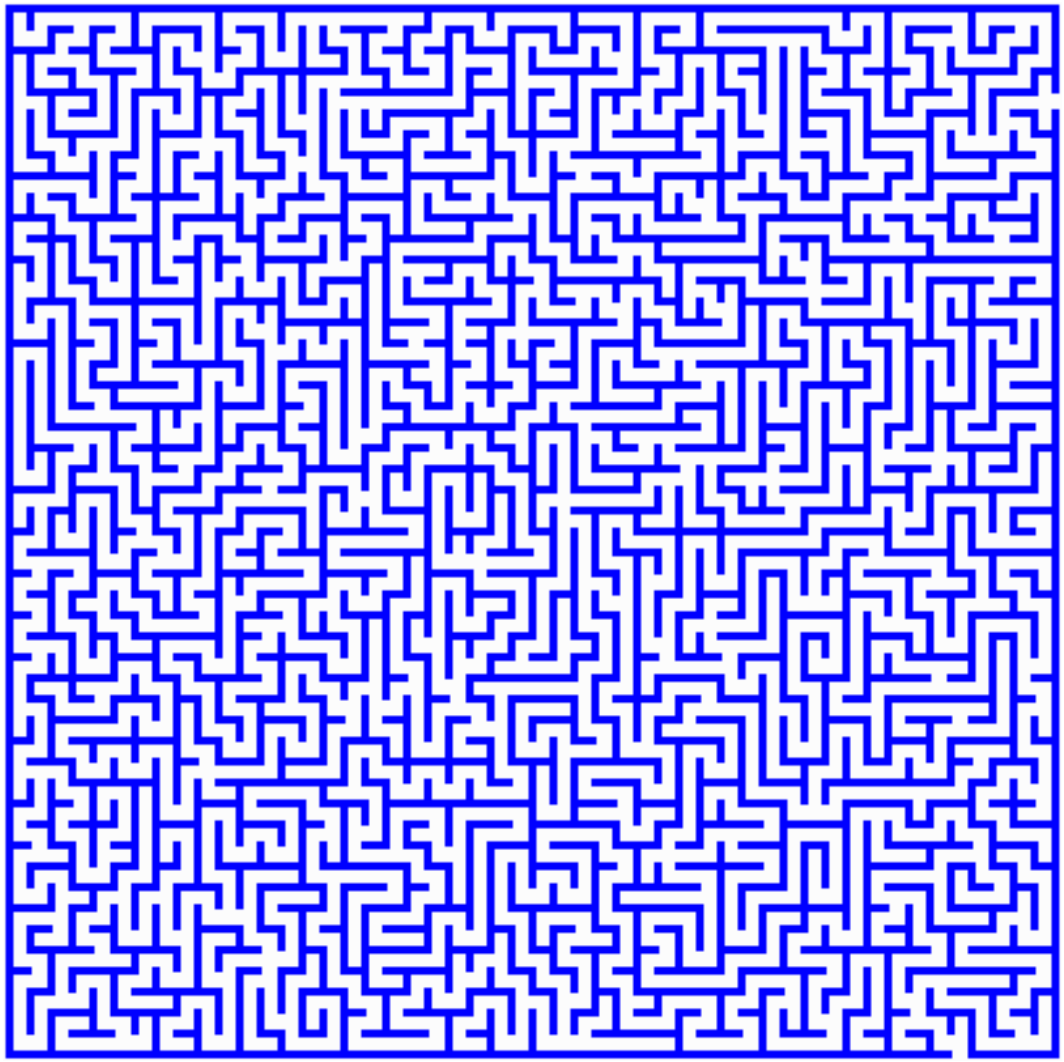
see issues and anti-patterns in Appendix.

* e.g. knowledge of photography, optics, mastering the use of camera equipment, lighting, set construction, editing, special effects, etc

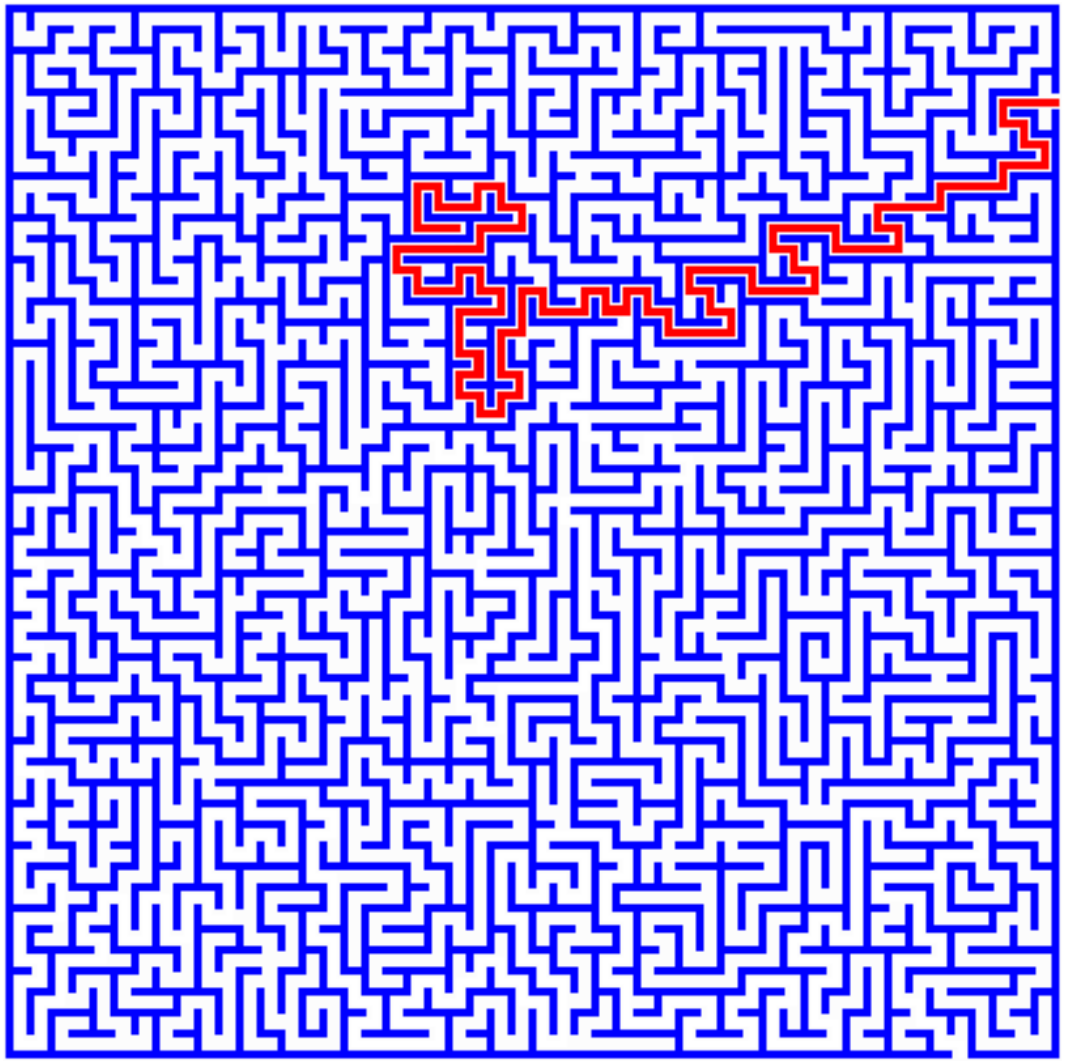
Agenda

- How do we start a project ?
- Introduction
- Principles, Practices, and Values
- Iterations and Increments
- Planning and Scrum of Scrums
- Task Board
- Daily Scrum or Stand-up
- Artefacts
- Agile Architecture
- Improvement (retrospective)
- Finale
- Appendices:
  - Essence Cards : Core Kernel and Scrum Essentials
  - Scrum Anti-patterns
  - Process Flow
  - Common Project Issues
  - Agile issues

Where Do We Start or End ?



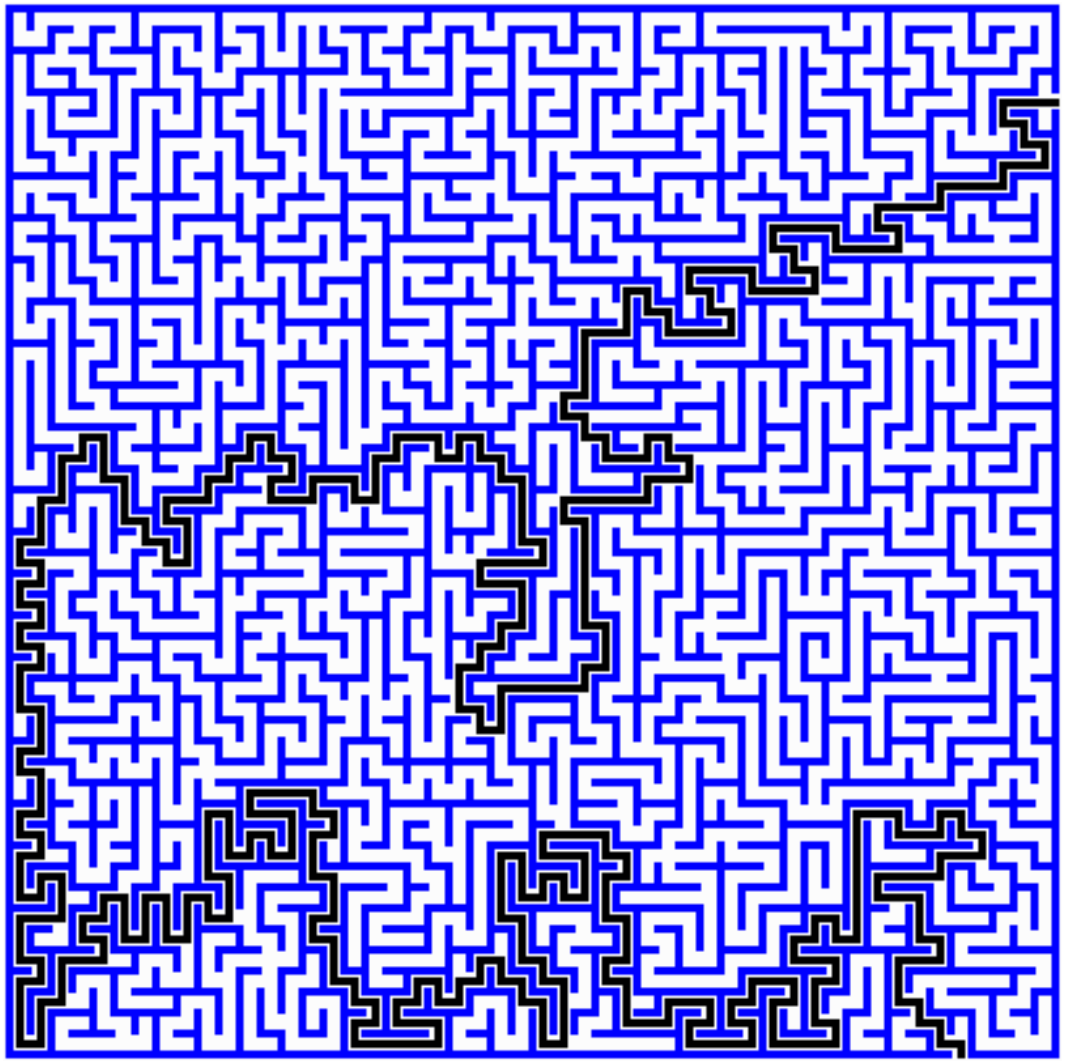
Lets try any path ?



Dead End. No way forward:

- Too many bugs
- Risks > Value
- Team burnt out
- Requirements keep changing
- Project Stopped

We always did it this way, its what we know ?



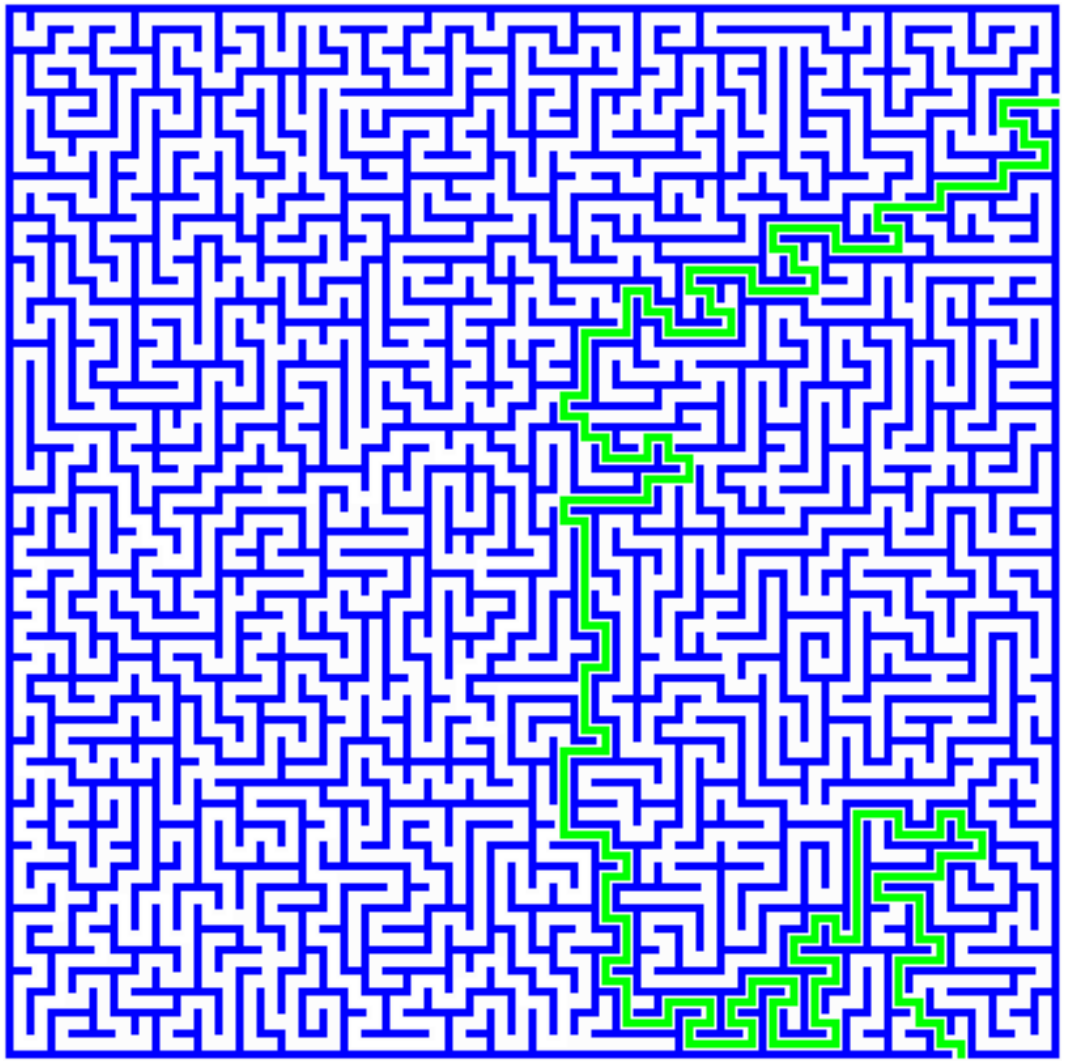
Project completed:

- Late
- Team confused
- Significant rework
- Cost > Budget
- Poor Quality
- Team burnt out

There are many real issues a project can face. I have collected many at the end of this talk.

Having an add hoc approach, with no shared or transparent ‘method’ can lead the project astray.

There are Better Paths ?



Project completed

- On Time
- On Budget
- Value > Risks
- Team is happy

The point of a good “Way of Working” is to find a most efficient and happy path.

The one-off cost of changing the way of working is far cheaper than continually repeating a poor method that we know.

Introduction

Is 'Process' the right thing to fix ?

Improving the Way of Working - Today

Overwhelmingly, the world is going Agile :

- 71% of organizations have adopted Agile methodologies,
- 90% of Agile projects have faster time to market than the average for traditional project management.
- A key contributor to the success of Agile is a different approach to planning.
- The Open Group now see Agile as mature, and the old ways (e.g. waterfall) as immature.

There are many versions of "Agile", and how well they fit, depends on the unique team; the individuals that make it up. However, what is key is the maturity of the organisation to improve, and change, and not be fixed in old 'safe' ways of doing things. I**t can be very challenging and disruptive for some companies to change, to innovate their roles, processes and structure.**

Instead of 'process' this presentation focuses on simple principles, practices, anti-patterns and check lists that establish a "way of working". These were derived from a real team's work to improve, and then tested and refined across several other teams.

The goal of the team was to continually improve its maturity, in moving away from a negative autocratic command and control, micromanagement styles.

What Level of Maturity is your Way of Working ?

The Team got here

Maturity Level 5	Innovative at Scale	Think out of the box. The enterprise leverages rapid learning cycles not only to improve but also to innovate. The enterprise invents and experiments with new digital offerings, business, and operating models.
Maturity Level 4	Continuous at Scale	Optimized. The enterprise masters continuous improvement, DevOps, Cloud Native Computing, and the safety and security disciplines. The organization adapts as it learns.
Maturity Level 3	Agile at Scale	Autonomous yet aligned. Agility scales without creating chaos. Digital technology usage grows rapidly. Leadership and organization shifts away from a command and control style.
Maturity Level 2	Agile Teams	New ways of working. Agile methodologies such as SCRUM are generalized in the IT department, and digital technology is experimented with. The enterprise culture evolves toward a dual model.
Maturity Level 1	Initial	Agile experiments. Some IT teams start using Agile methods. The command and control culture remains dominant. Little if any digital technology is experimented with.
Maturity Level 0	Old School	Waterfall-dominant. The enterprise is siloed, uses waterfall methods, and lags behind on the technology side.

A new maturity scale to replace CMM from the OpenGroup.org

Changes to ‘Agile’, Scrum and XP.

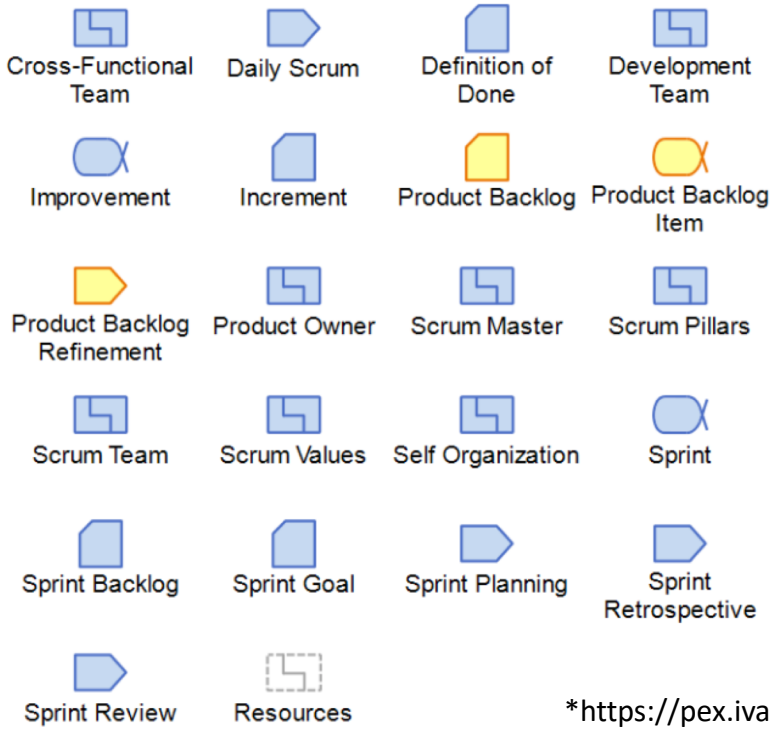
The Agile Manifesto is very old and needed an update. The team found that Agile processes like SCRUM and XP could by improved with recent new ways. The main changes:

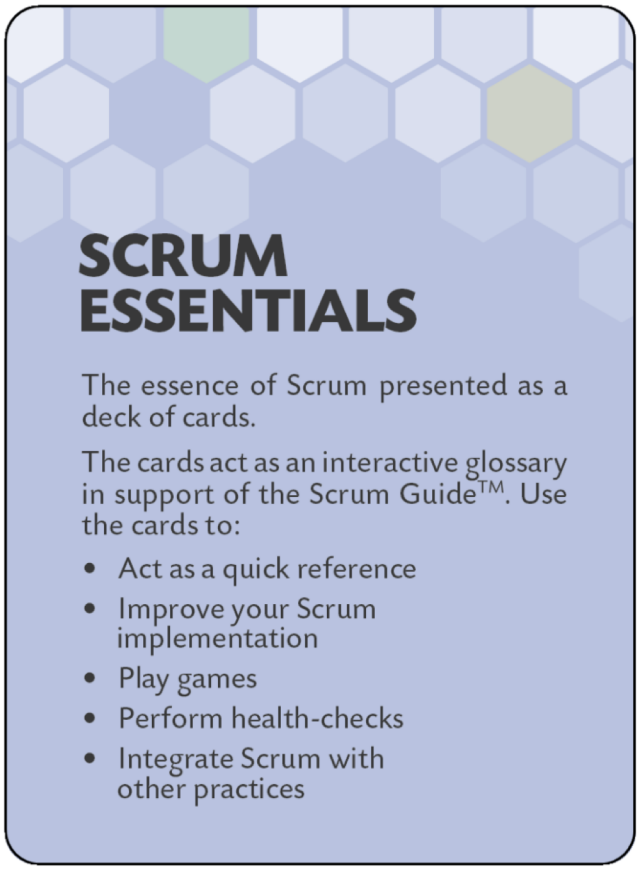
- **The ‘Manifesto’** although claiming 12 principles, does not meet the definition of a ‘principle’ for many. We used Bertrand Meyers improved principles as a basis.
- **Use Case 2 Approach** - Use Cases provide an E2E context for Stories (Jacobsen), and define well, how stories are packaged up. Whereas Epics to Stories is frequently a confusing relationship, negatively impacting work. We often get poor stories from Epics. Jira has a lot to answer for as Epics are used for reporting (a bad practice). Unfortunately, tools can impact the way of working negatively.
- **ESSENCE Check list cards -** are an Open Standard used to improve planning and state monitoring (SEMAT.org):
  - These provide checklists, like a pilot performing flight checks. These are done at all stages of the flight. Pilots are very experienced experts but still use simple checklists.
  - These can be tailored for the team, as they go.
  - Provides sets of well-established base practices.
- We also enhanced SCRUMS basis on “Empiricism” with ‘Explanation’ as the key;
- This pack aligns, in intent, with the official SCRUM Guide (scrumguides.org)

Essence Check List Cards

Two sets of Essence cards were employed:

1. **Essence Kernal Cards** – generic base (bare minimum) cards for all ‘ways of working’;
2. **SCRUM Essentials** – cards that capture the ‘essence’ of SCRUM, based on the official guide.
   - These can be found in full in the appendix or online*;
   - The slides that relate to these cards, will have the relevant icons at the bottom:





*https://pex.ivarjacobson.com/sites/default/files/practice/scrum_at_scale_cards.html

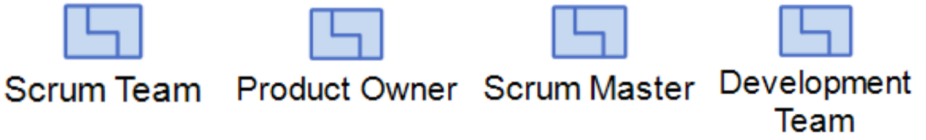
Simple Set of SCRUM Roles

The team roles:

- **Developers** – build the highest value product by working together in the most effective way possible. **Architect** is a specific skill set focused on quality and big picture solution;
- **Scrum Master** – steer the whole team to becoming increasingly effective; a facilitator and Coach, team assistant, but not a manager.
- **Product Owner** - brings the problem to team to solve, work with development team to maximise the value of the product.

"Scrum recognizes no sub-teams in the Development Team, regardless of domains that need to be addressed like testing, architecture, operations, or business analysis. " Scrum Guide

- The team was completely flat, no hierarchy; there were no 'bosses' or managers.
- The team is cross functional and self organizing, "everybody all together".
- Leadership is a responsibility of all members of the team;
- Project management, business analysis, and quality assurance are done by everybody, the whole team. **These are not roles but *practices*.** Quality is pervasive, testing is done early and automated.



Way of Working = Transparent Shared Beliefs and Values

Everyone is different and has different backgrounds, and experiences, we all have different:

- **Beliefs**
- **Assumptions**
- **Values**

- **Making a 'way of working' explicit**, transparent, by describing it, by agreeing on principles, practices, and reviewing this frequently makes our beliefs, assumptions and values open for sharing and review. We can pin-point where we have differences and work to resolve these.
- An important aspect of describing a way of working is that it defines a ***shared language*****.** The team can talk about the way they are working in an understandable manner. Why should we just magically come together and understand terms like 'scrum master', 'standups', 'stories', 'use cases', and 'slices'.
- Often people report that team X works well, with no process, how is this so ? They do, in fact, have a process; however it is just not explicit or transparent. It is not described. They are lucky that the members share beliefs, values and assumptions, out of the box. In most cases, this luck does not happen, and work is required to get to a shared state.

Autonomy, Mastery, Purpose (David Pink)

Successful teams and individuals are driven by:

- ***Autonomy***, or the desire to be self-directed;
- ***Mastery***, or the itch to keep improving at something that's important to us;
- ***Purpose***, the sense that what we do produces something transcendent or serves something meaningful beyond than ourselves.

"This is what it means to serve: improving another's life and, in turn, improving the world."

D.P

"Control leads to compliance; autonomy leads to engagement."

D. P.

"Great teams do not hold back with one another. They are unafraid to air their dirty laundry. They admit their mistakes, their weaknesses, and their concerns without fear of reprisal."

Patrick Lencioni

"Successful agile teams are the ones that have the freedom to determine how they work. Do you believe this freedom is something worth fighting for ?

Ivar Jacobsen

Autonomy, Mastery, Purpose

Small, autonomous, cross functional teams that self organise, that are the masters of their own fate, with a shared purpose aligned with their business, and the customer outcomes:

- Leadership by the team itself is core to Agile;
- Team members should see themselves as running their own business, they own their quality, they needs to be entrepreneurial (this requires autonomy);
- Leaders don't give orders; they share a purpose. The 'why' is shared.
- Innovation is primary to purpose. The team innovates to achieve the outcome.
- Autocratic leadership, Micromanagement and Command-and-Control kill autonomy, mastery and remove purpose, and ultimately eliminates collaboration;
- Rather than project manage, do what US Marines do in the uncertainty of warfare: ***"define the goal but not the means";***
- Team does not deliver features but meets goals / outcomes whatever way they believe is best, they make decisions constantly on how. If a manager or product owner must be involved in this then inefficiency will result;
- Manager that control, create complex systems that require more control, and get more complex. They themselves need more information and start doing the work of the team. The manager in the end cannot maintain control, loose respect of the team, cannot serve, and quickly becomes a bottle neck resulting in a late delivery.

Empowering Autonomous Teams

"Organizational leaders hold the greatest responsibility in fostering environments conducive to the successful implementation of autonomous team outcomes. Even when considering the impact of various team dynamics on team-member interactions and the development of team processes, they must recognize that organizational dynamics overwhelmingly control the degree of success in team outcome implementation.

And the most crucial organizational dynamics moving autonomous teams toward successful outcome implementations include organization-wide commitment to using autonomous teams, upfront and unwavering commitment to allocating resources for team use, and frequent, face-to-face feedback."

Empowering Autonomous Teams
by: James Hess

Principles
Practices
Values

Organisational Principles

1. Put the customer at the center – solve for their outcomes.
2. Let the team self-organize: desire autonomy, mastery and purpose.
3. Work at a sustainable pace
4. Develop minimal (lean) software:
   a) Produce minimal functionality
   b) Product only the product required
   c) Develop only code and tests

Minimal Functionality Minimal Product (MVP) Minimal Artefacts

5. Accept change - Manage Queues rather than Timelines
6. Team manages its “Way or Working”; it is continually improving, described, open, transparent
7. Use simple checklists to manage project state
8. Use well established practices
9. Everybody all together from early on

This is partially adapted from a ‘re-write’ of the Agile manifesto by Bertrand Meyer, and from Coplien, Lean Architecture.

Technical Principles

1. Develop Iteratively:
   a) Produce frequent working releases
   b) Freeze requirements during the iteration
   c) Treat tests as key resources
   d) Evolve the Agile Architecture and Requirements, just in time
   e) Agile Architecture is both Intentional and Emergent
2. Do not start any new development until all tests pass:
   a) Test first – test driven development (TDD)
   b) Automate testing into build and release; regression test
3. Express requirements as scenarios (Use Cases, Stories and NFRs):
   a) Use case driven development
   b) Slice use cases into stories to define tasks, and tests
   c) All requirements have acceptance criteria

Practices and Methods

Practice: “is a repeatable approach to doing something with a specific objective in mind”

Method: a description of what is done. Practices are assembled to make a method - what we rather call a “way of working”.

Good Practices are:

- Lightweight – focus on the most important things to do and produce
- Adaptable – can be tuned on the fly
- Extensible – can grow as needed
- Transparent – frequent demonstration of results
- Ensure quality – verification and testing are built in
- Focus on working software – help with the team’s main task
- Focused – capture one aspect of software development
- Pragmatic – provide useful advice to individual and team
- Play well in itself as well as together with other essential practices
- Repeatable and predictable

Practices adopted by the team

Management

- Shared Vision
- Risk-Value Lifecycle
- Iterative Planning
- Whole Team
- Shared leadership
- Team Change Management
- Just In Time (JIT)
- State based checklists
- *Andon* - stop the line
- *Kaizin* - continuous improvement

Technical

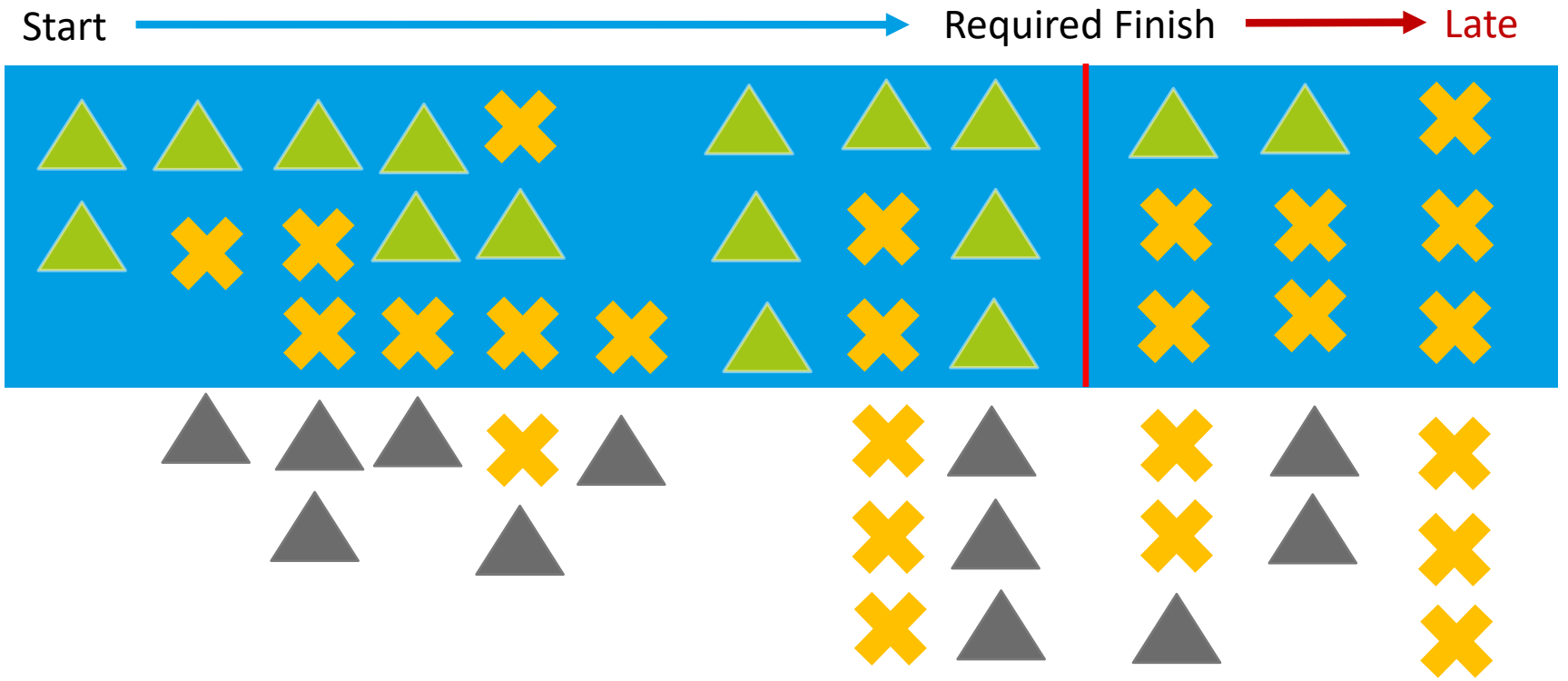
- Iterative & Incremental Development
- Evolutionary Agile Architecture
- Evolutionary Design
- Continuous Integration/Deployment
- Refactoring
- Test Driven Development
- Use Case Driven Development
- Concurrent Testing
- Automated Testing
- Acceptance Testing

Attitudes and Values

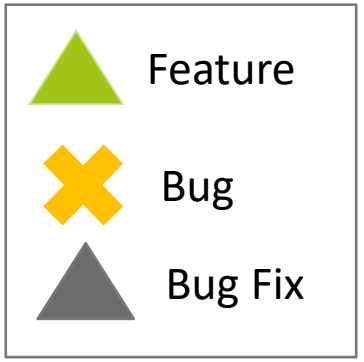
- **Commitment** - the whole team needs to be committed to success of project. The stakeholders need to commit to the teams objectives and way of working.
- **Focus** - keep focus on the iterations objectives, don't get side tracked on other activities. This is particularly true of management team that needs to stick to teams objectives and resourcing, and not change them during an iteration.
- **Honesty** - be open and honest about what you can achieve. Share problems, issues and risks. Openly discuss solutions.
- **Respect** - the team members need to respect one another, stakeholders and customers
- **Agility** - be prepared to adapt and change the way of working to facilitate success of the project. Strive for continual improvement.

Iterations (Sprints)

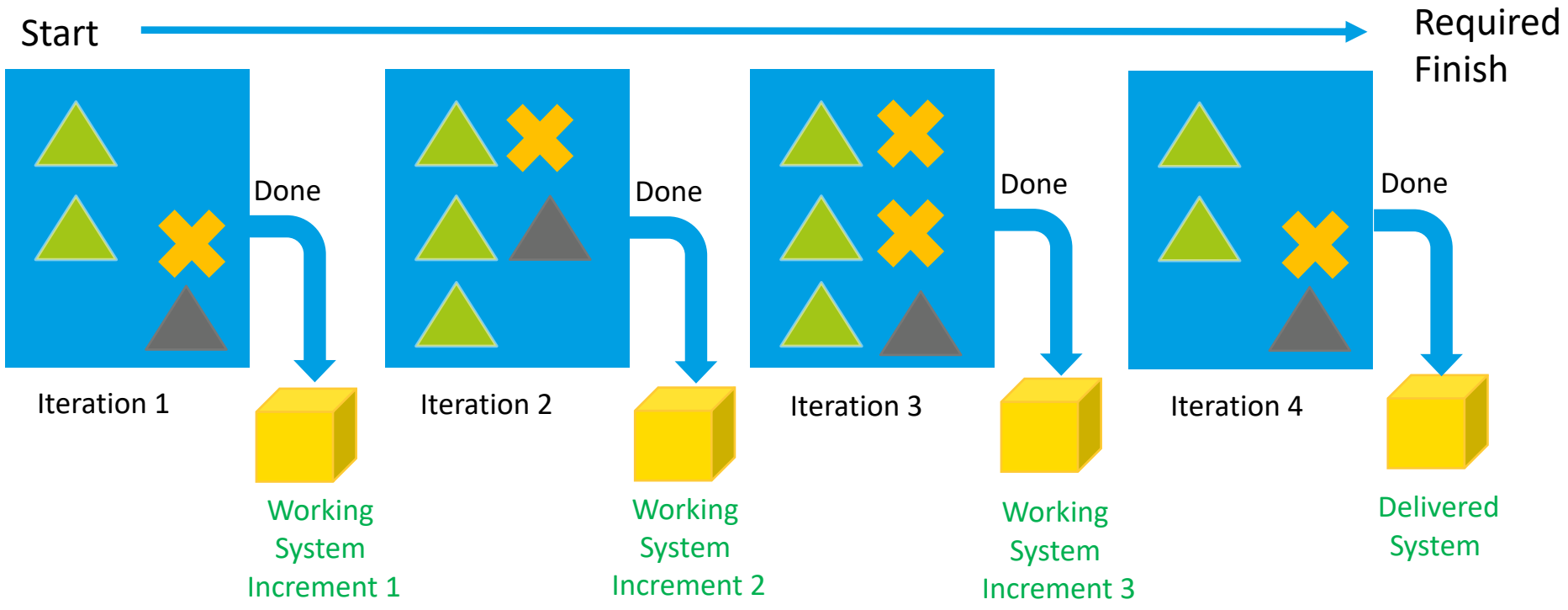
Anti-Pattern : One Long Build – 'the death march'



- A year of continuous work and it never seems to end;
- Nothing to show, work is so complex, bugs rule, team burnt out;
- Integrate and nothing works together;
- 'Feature Factory' Focus;
- Features are vague, work stops while that is sorted out;
- More developers and QAs added in to help fix bugs and keep testing;
- Stakeholders loose faith, as finish date is missed.



Solution : Build Iteratively – Release Increments

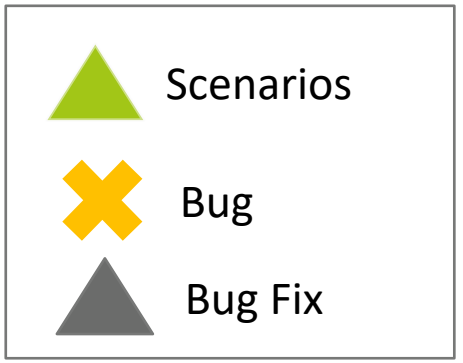


Iteratively add new scenarios, build and release a new small working system; an increment.

- Demonstrates 'it works' quickly to stakeholders, and allows immediate feedback to be sought on the iterations goal.

Employ a simple measure of progress:

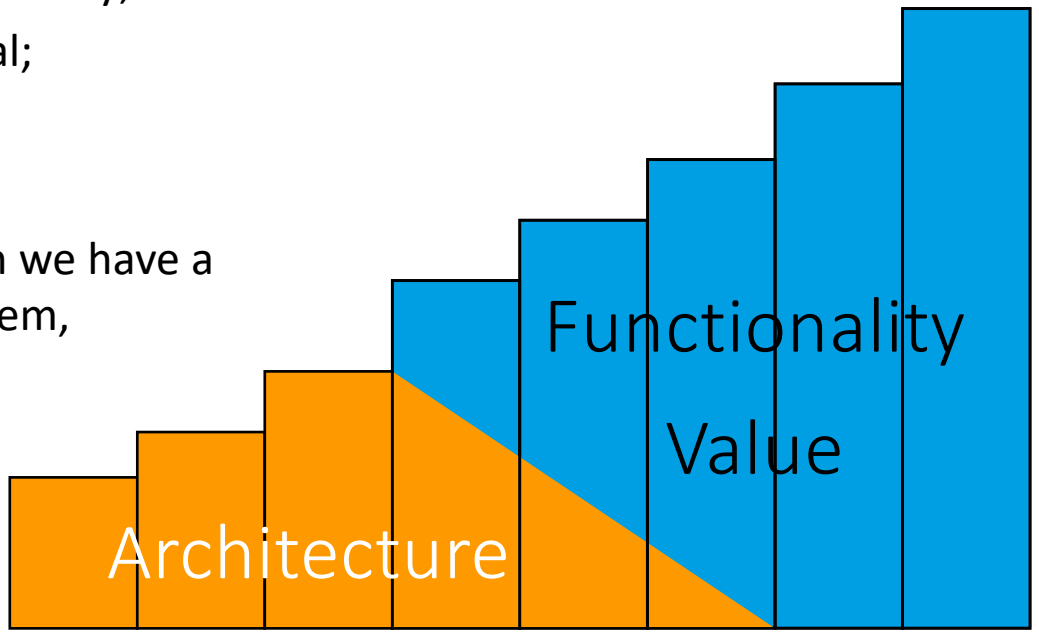
- Running systems in which a number of scenarios have been implemented, tested and verified.



Solution: Iterations and Increments

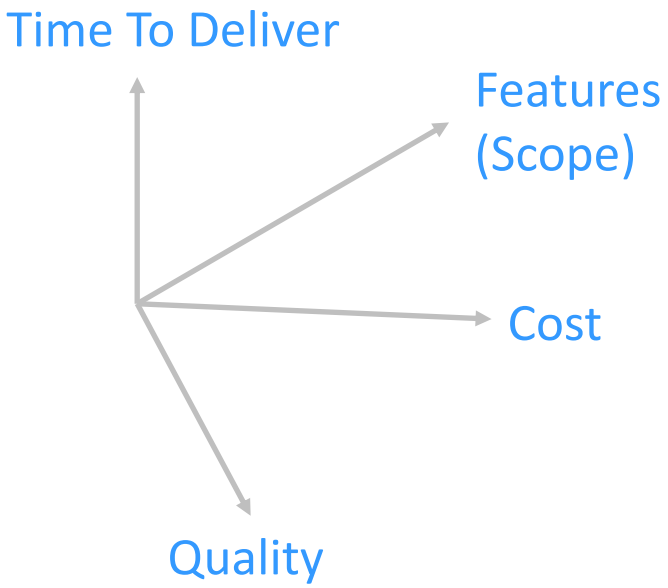
The bigger system is evolved from a series of smaller systems, each of which extends the other:

- Initial iterations focus on a stable architecture;
- Later iterations deliver functional value to stakeholders;
- The system grows Incrementally;
- Each increments has a goal;
- Quality increases;
- Test early, as we go;
- At the end of any iteration we have a releasable product or system, of known quality.



Increment

Balance Trade-offs: 4 Dimensions



Iteration length is decided by Team:

- **Fixed** – time boxed. 4-6 weeks allows enough value to be delivered, and team to run autonomously.
  - 1-2 weeks is too short = production line mindset (micro-management).
- **Variable** – based on the tasks prioritised.

Project behind schedule, choices are:

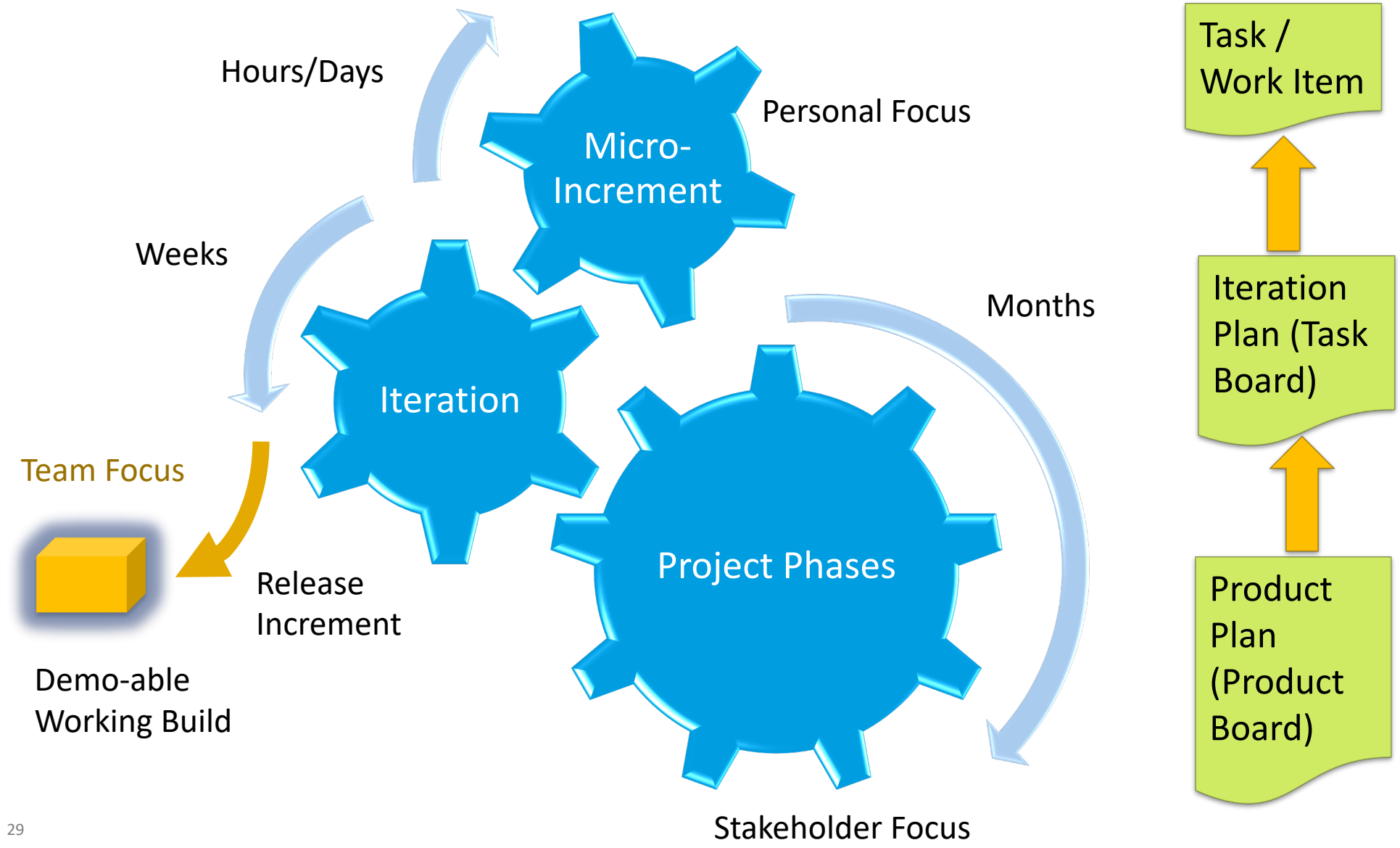
Waterfall:

- Don't deliver anything but documents, e.g. requirements spec;
- Deliver untested system with bugs;
- Trade-off Quality;
- Fix later in Maintenance;
- Deliver Late.

Iterative:

- Requirements prioritised.
- Always deliver a working tested quality system with subset of requirements that give high value;
- Deliver other features in next release;
- Manage the queue rather than timeline;

Lifecycle and Focus - Team, Personal and Stakeholder



Benefits of Iterations

- Iterations give the team and stakeholders control over schedule, cost and scope.
- Produce demonstrable working value to stakeholders early;
- Software development is a knowledge discovery process, we can improve iteratively and learn as we go. Architecture is Emergent. Mistakes may be made but fixed quickly;
- Get feedback quickly – fail fast approach;
- Reduce the impact of business change, e.g. only one iteration, not whole project affected;
- Focus the team on a small set of manageable objectives and tasks at a time;
- Allows a focus on estimation – a specific meeting to just estimate a small set of tasks;
- Allows a focus on planning – a specific meeting to just plan the iteration;
- Allow the team to self organize around who is doing what;
- Development can stop at any time, with something of value delivered;
- Reduces the reliance on falsely accurate estimates, accurate estimation is very difficult. Most new products are new experiences/technology and estimates are not really knowable;
- A feeling of completion and accomplishment is achieved with each working release;
- Provides a point to improve the process, the “Retro”. Team takes a breather and reflect;
- A new fresh iteration is a new fresh start, in many ways;
- All up, it alleviates the doom of the “death match”.

Planning

How to start the project

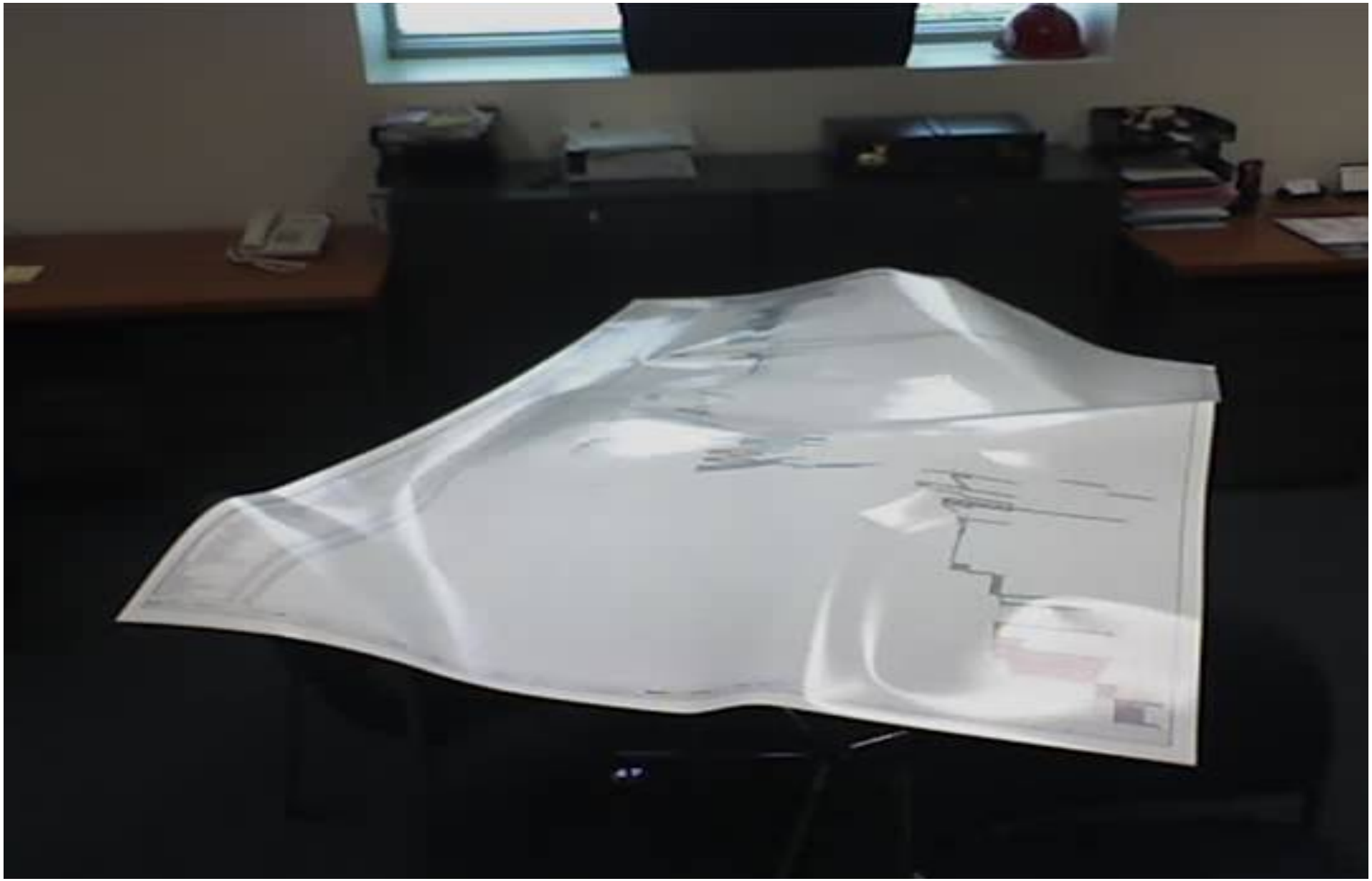
The start of the project is a well-defined ceremony, the aim is to get **ALL** stakeholders and **ALL** the team on the same page. **Principle 9: Everybody all together from early on.**

Agree on:

- **Vision** – establish the goals clearly for the project;
- **High Level requirements**, key epics, the important functional and non-functional ;
- **Solution Architecture**, very high level, on a page;
- **A clear definition of ”Done**” for product and iterations;
- **Team charter, roles and values** defined, who does what, working agreements defined;
- **Agreement on the “way of working’** – this is our method, how the project will operate, and be delivered. An approach to timelines is critical;
- **Demo and Review agreement** - Stakeholder review, who attends, and why.
- **Shared commitment** to allocate effort for improvement;

The above may seem obvious but many projects start without “starting”, they just drift onto operation in a state of confusion and chaos; no one knows what they are meant to do or why. (See Issues at end of talk). Different members have different knowledge, which sends the project off in the wrong direction from day 1.

The BIG Plan



Iterative (Sprint) JIT Planning

“No plan survives contact with the enemy” Helmuth von Moltke

“Everyone has a plan until you get punched in the mouth” Mike Tyson

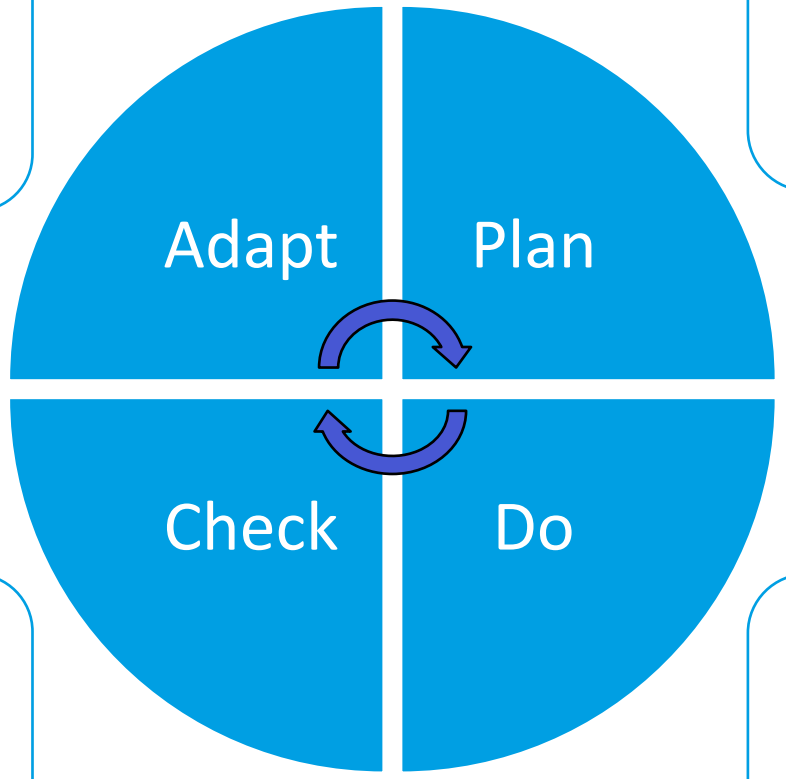
A detailed up-front plan on day one will most likely be wrong on day two:

- Project planning is inaccurate when you plan beyond the initial phases of work;
- Plan the current iteration only in detail, while planning the whole product (project) only at a very high level, the potential milestones;
- Plan the next iteration at a lower level of detail. Future iterations are not planned;
- A narrow focus on a small set of **goals** and tasks:
  - Better estimated – face the fact that estimation is a black art, it will be wrong, and is not a forecast. New technology and new problem domains will be poorly estimated. Building software is knowledge discovery, often there is no basis to estimate;
  - Reduces task switching and delay (waste);
  - Their state can be monitored, and ‘fixed’ more efficiently if they are blocked;
  - Examine the cost of delay, rather than cost.
- Planning is JIT, iterative, active and ongoing.

Plan-Do-Check-Adapt Cycle

- Reflect on what happened
- Improve:
  - the way of working
  - the quality of work
- Reduce waste

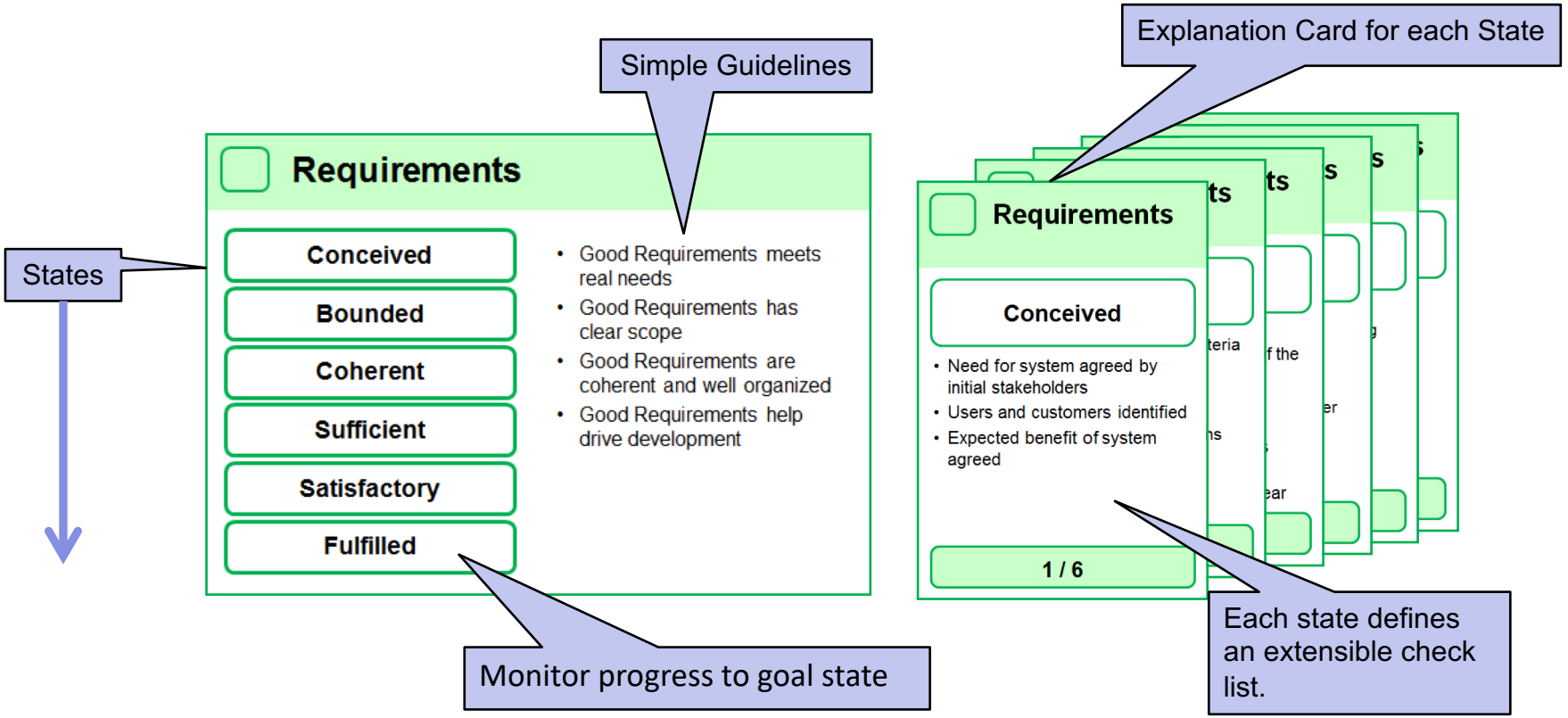
Determine:
- Current state
- Next state, the goal
- How to achieve next state, what is ‘Done’



- Track the Work in the Iteration.
- What’s the state ?
- Check the work is ‘Done’

- Work to achieve the next state, the increment
- Remove blockers as they occur

Simple Checklist Cards are used to manage state



Work moves items through states. For example, the goal of building a system is to get "*Conceived*" **Requirements** to a "*Fulfilled*" state, by executing **Work**, by the **Team**, to deliver an "*Operational*" **Software System**. We are interested in monitoring these items to plan work. The next page shows how some of these items move from state to state.

Note: The cards used are detailed in remainder of pack. These are a basic set of cards and more specific ones are described for SCRUM. These cards should be owned and tailored by the team.

Monitor our Current State with Essence Cards

States Change – progress from left to right

Requirements

Requirements — **Conceived** — 1 / 6
- Need for system agreed by initial stakeholders
- Users and customers identified
- Expected benefit of system agreed

Requirements — **Bounded** — 2 / 6
- Theme, scope, success criteria of system is clear
- Mechanisms for managing requirements in place
- Constraints and assumptions considered

Requirements — **Coherent** — 3 / 6
- Described requirements provide coherent picture of the system
- Conflicting requirements separated
- Important usage scenarios explained
- Priority of requirements clear

Requirements — **Sufficient** — 4 / 6
- Requirements adequately describe solution and acceptable to stakeholders
- Rate of change to agreed requirements is low and under control

Requirements — **Satisfactory** — 5 / 6
- System implementing requirements is worth making operational
- Enough requirements are implemented

Requirements — **Fulfilled** — 6 / 6
- System implementing requirements is accepted as fully satisfying the need
- No outstanding requirement items prevent system from being accepted
- Stakeholders accept requirements as accurate

Software System

Software System — **Architecture Selected** — 1 / 6
- Architecture selected that address key technical risks
- Criteria for selecting architecture agreed
- Platforms, technologies, languages selected
- Buy, build, reuse decisions made

Software System — **Demonstrable** — 2 / 6
- Executable version of system demonstrates architecture is fit for purpose
- Supports functional and non-functional testing
- Critical interface and system configurations exercised

Software System — **Usable** — 3 / 6
- System is usable and has desired quality characteristics
- System can be operated by users
- Functionality and performance have been tested and accepted
- Defect levels acceptable
- Release content known

Software System — **Ready** — 4 / 6
- System (as a whole) has been accepted for deployment in operational environment
- Sponsors, users, stakeholders accept system as fit for purpose
- Installation and other documents available
- Operational support in place

Software System — **Operational** — 5 / 6
- System in use in operational environment
- System available to intended users
- At least one example of system is fully operational
- System supported to agreed service levels

Software System — **Retired** — 6 / 6
- System no longer supported
- Updates to system will no longer be produced
- System has been replaced or discontinued.

Work

Work — **Initiated** — 1 / 6
- Work initiator and client known
- Work goal and constraints clear
- Sponsorship and funding model clear
- Priority of work clear

Work — **Prepared** — 2 / 6
- Cost & effort understood
- Funding in place
- Resource availability and risk exposure understood
- Governance model is clear
- Integration and delivery points defined

Work — **Started** — 3 / 6
- Development work has started
- Work progress is monitored
- Work broken down into actionable items with clear definition of done
- Team members are accepting and progressing work items

Work — **Under Control** — 4 / 6
- Work going well, risks being managed, productivity levels acceptable
- Unplanned work & re-work under control
- Work items completed within estimates
- Measures tracked

Work — **Concluded** — 5 / 6
- Work to produce results have been finished
- Work results are being achieved
- The client has accepted the resulting software system

Work — **Closed** — 6 / 6
- All remaining housekeeping tasks completed, and work officially closed
- Everything has been archived
- Lessons learned and metrics made available

Team

Team — **Seeded** — 1 / 5
- Team's mission is clear
- Team knows how to grow to achieve mission
- Required competencies are identified
- Team size is determined

Team — **Formed** — 2 / 5
- Team has enough resources to start the mission
- Team organization & individual responsibilities understood
- Members know how to perform work

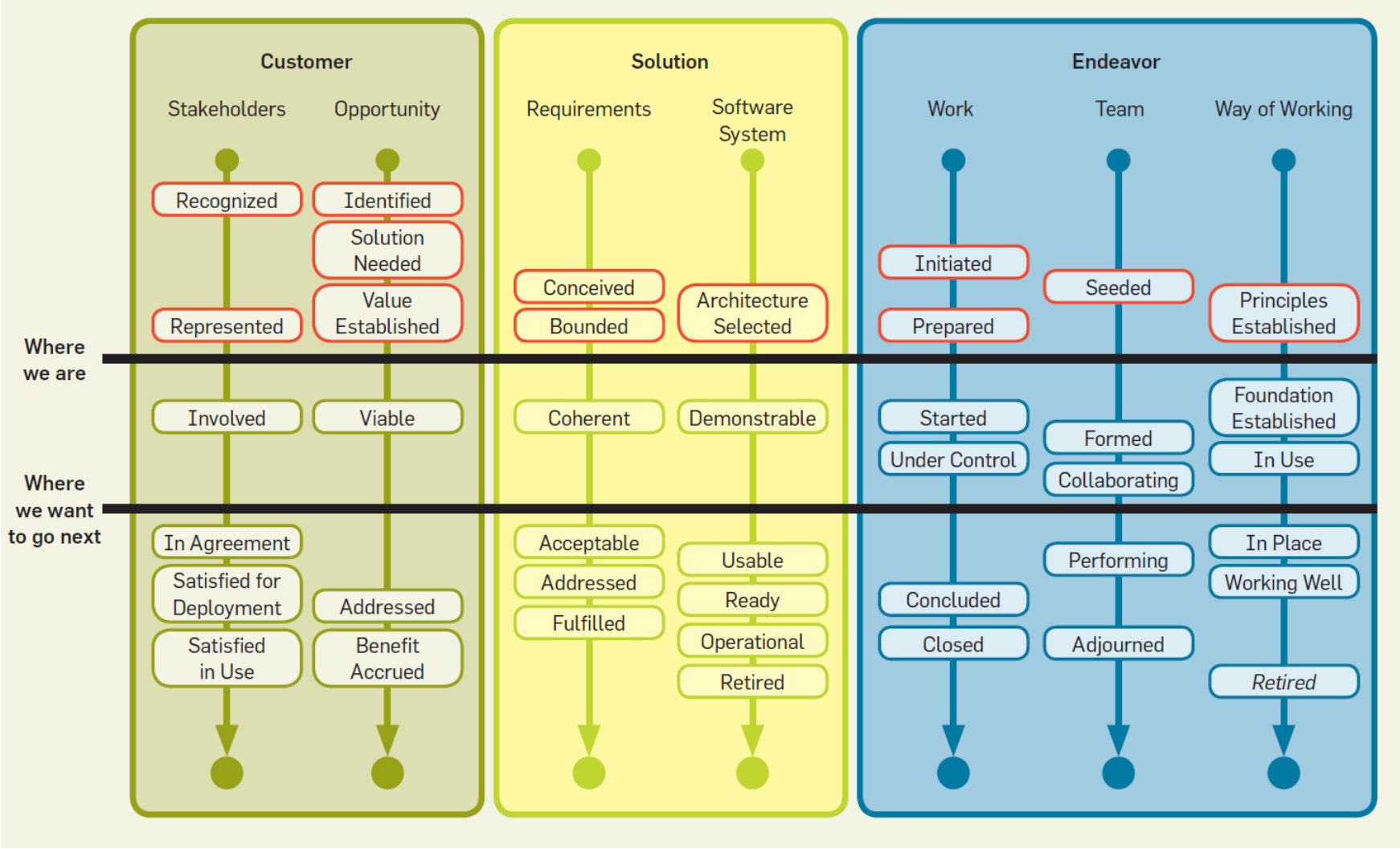
Team — **Collaborating** — 3 / 5
- Members working as one unit
- Communication is open and honest
- Members focused on team mission
- Success of team ahead of personal objectives

Team — **Performing** — 4 / 5
- Team working efficiently and effectively
- Adapts to changing context
- Produce high quality output
- Minimal backtracking and re-work
- Waste continually eliminated

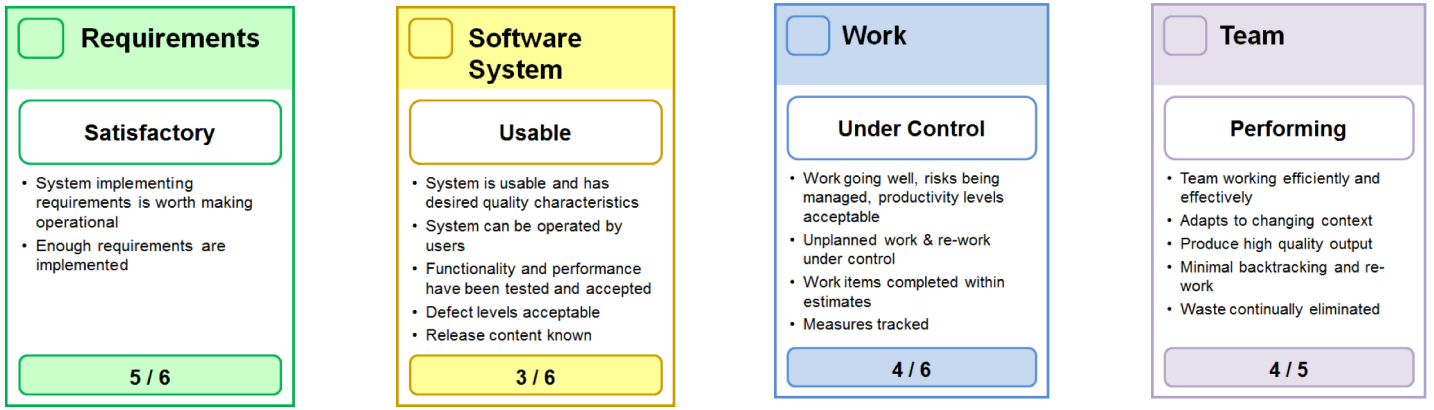
Team — **Adjourned** — 5 / 5
- Team no longer accountable
- Responsibilities handed over
- Members available for other assignment

Visualise : What have we done – Where next ?

The 7 basic monitored aspects, and their states, as sliders.



Example - How to achieve the Next State ?



For example, we may decide that these are our objectives for our next iteration:

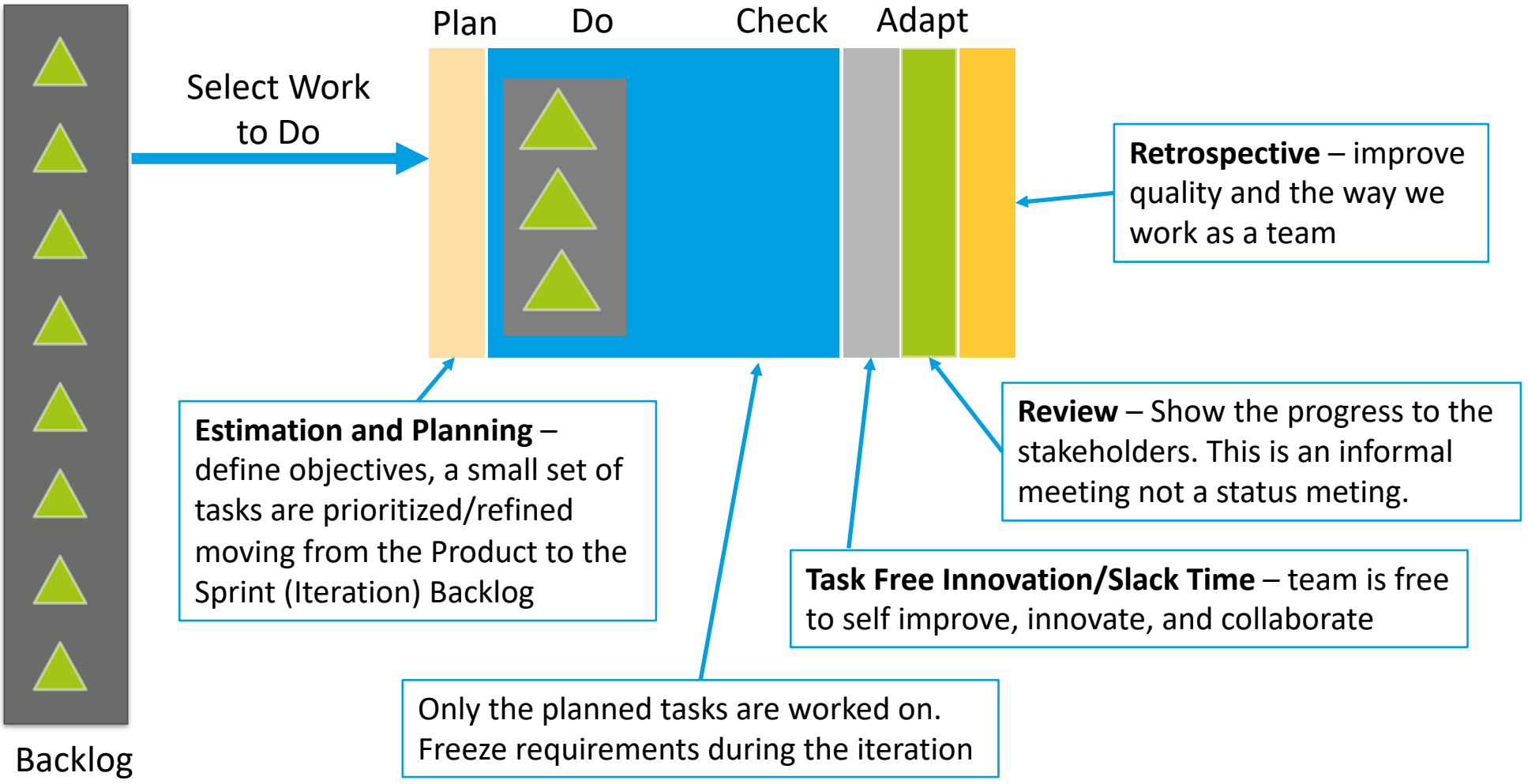
- Focus on getting a subset of *satisfactory* requirements
- Release a *usable* Software System, e.g. an MVP version
- Ensure the right practises are in place to get work *under control*
- Ensure the team is *performing*

To reach our objectives, we define a small set of tasks, for each of these.

The product backlog has has all the work required for the whole product, and is where new work goes until prioritised. A selection is taken for the next iteration.

Anatomy of a Sprint (Iteration)

An Iteration is broken into 5 steps, based on the PDCA cycle:



Product Backlog | Product Backlog Refinement | Sprint Backlog | Sprint Planning | Sprint Retrospective | Sprint Review

Planning Example – Move cards to new states

The team gets together, all in one room, to estimate and plan the next iteration:

- Estimates are not made individually but by the team as a group;
- Estimates are not forecasts but a best guess based on experience at hand;
- The product owner suggests the priorities, from the product backlog. The use cases and stories, are selected, and go into the Iteration backlog.

Planning in a nutshell:

- The team checks the requirements are in the state “Satisfactory”, otherwise a task is created to complete this work.
- Backlog Items have dependencies, and some must be done before others;
- A parallel endeavor is to get the Architecture (Software System) states to “Useable” and “Ready”. Tasks are created for these;
- The team slices (breaks) the “Satisfactory” Use Cases down to “Satisfactory” Stories, then development tasks are created and estimated;
- Objective measurable criteria for “Done” is agreed (see later slide);
- The team checks the work fits into the iteration, if not, some go back into product backlog.
- *Only the Development Team can change its Sprint Backlog during a Sprint.*

Sprint Planning

Scrum of Scrums

Program – Scrum of Scrums

Program Level – Program Increment (PI) 1

Milestone

PI 2

Release Plan 1

Release Plan 2

Product/Program Backlog

Team Level

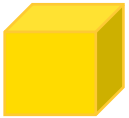
Iteration 1

Iteration 2

Iteration 3

Iteration 1

Spring Backlog



Internal Release

Internal Release

External Release

Internal Release

Release Plans - Release Cadence (3-4 Iterations)

Program Increment 1

Iteration	Items	Dependency	Date
1.1 *	MVP of new Order UI	UX team designs	20 Aug 2017
1.2	Backend for UI	Team 2 Order DB Changes	20 Sep 2017
1.3	Harden: Release New Order		20 Oct 2017
1.2-3	Architect Job (next release)	Lays our runway for PI 2	20 Oct 2017

Program Increment 2

Iteration	Items	Dependency	Date
2.1	New Job feature		20 Nov 2017
2.2	Backend Job Service		20 Dec 2017
2.3	Integrate with Order		20 Feb 2018
2.4	Harden: Release New Job	Marketing team	20 Mar 2018

* Only the next Iteration is planned in detail, further out less detail. Detailed plan for 1, less detail for 2, not for 3 (until complete 2). Only next PI is planned in detail.

Release Plan – Long Term View

Release	Items	Date
1	Order	20 Oct 2017
2	New Job feature	20 Mar 2018
3	Supplier Integration	20 June 2018
4	Port Backend to microservice	20 Sep 2018
4	Mobile MVP	20 Dec 2018

The Product Owner may manage the plan as a Product Roadmap.
This plan is highly intentional, emergent issues, changes in direction, and strategy will mean plan the will definitely be adapted.

Planning Estimation

"Worldly affairs do not always go according to a plan and orders have to change rapidly in response to a change in circumstances. If one sticks to the idea that once set, a plan should not be changed, a business cannot exist for long." Taiichi Ohno

- Estimates are not forecasts, they will be wrong. No one can predict the future.
- Program level estimates will have high level of error, That's OK, they are our current best bet.
- Work that cannot be completed is placed back into Program/Product backlog, for next next Program Planning step.

- Iteration planning is estimated one Iteration ahead.
- Iteration Plans are also a best bet, they will be wrong.
- Work that is an issue is placed back into Sprint Backlog, for next next Iteration Planning step.

- Changes in Iteration Plans will change the Program Plans. Feedback Up.
- Estimates are more an input to these plans than an output.

Task Board

Task Board or "Wall"

"Use visual control so no problems are hidden."

From: 'The Toyota Way' a management philosophy of the Toyota corporation that includes the Toyota Production System.

A visual Kanban device is used to make the Iteration tangible and irrefutable:

- The objectives;
- Where tasks are up to;
- Who is doing what;

- The Task Board can be placed on a Physical wall, so that everyone / anyone can see, at any time where the project is at.
- **Nothing is hidden**. Transparency is key. Use camera to share off-site.
- The Overall Product/Project Plan, can also be visualized with a Task Board, in the same way, to provide a shared vison of where we are heading;

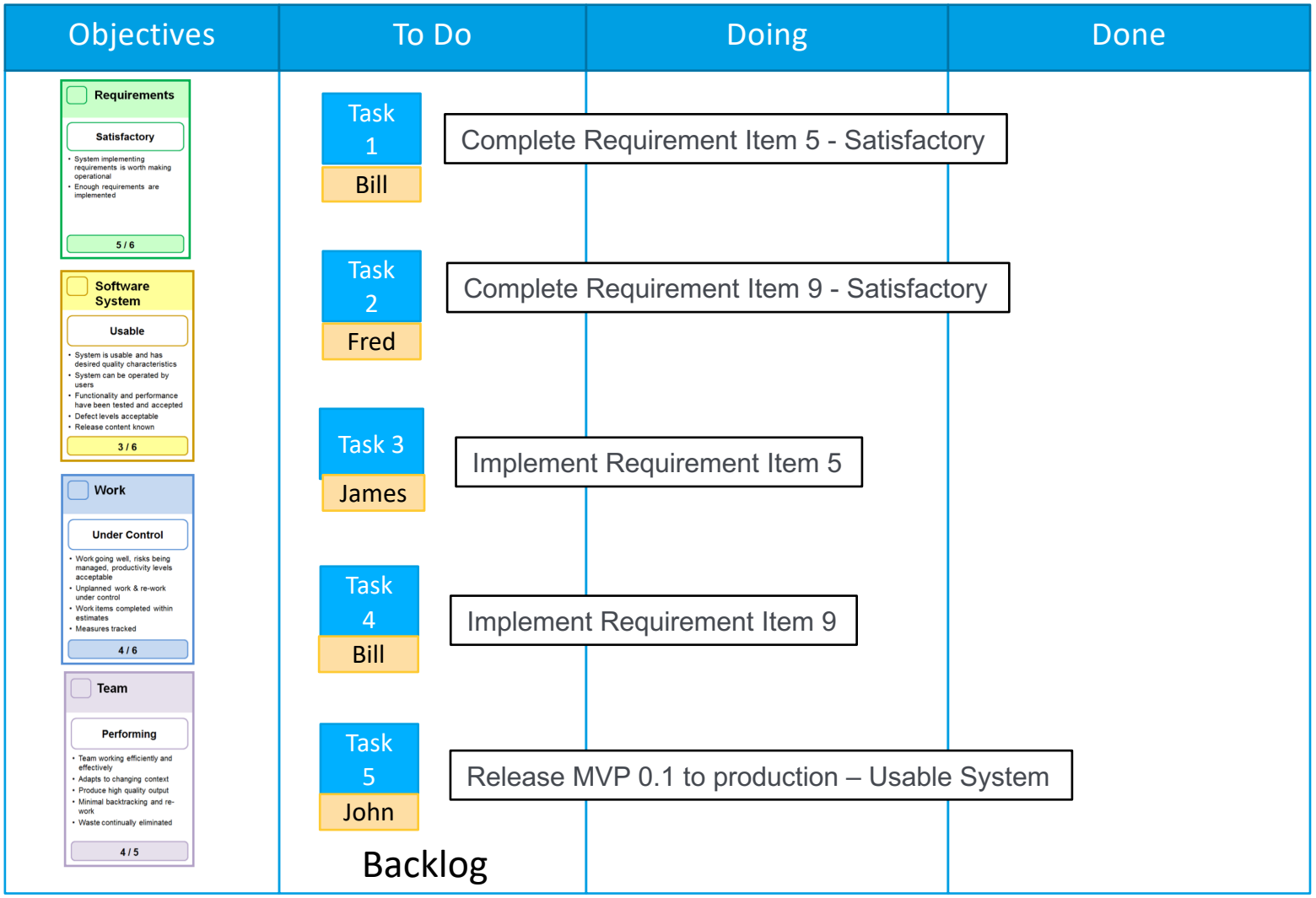
Iteration (Sprint) Task Board

Visually Shows:

- Clear Objectives and Goals for the Iteration – What do we want to achieve for who;
- Can use the Essence cards, where appropriate;
- How we will achieve those goals:
  - Iteration backlog explicitly shown
  - Tasks to do;
  - Who is doing what work – use name or avitar for the team members;
- What's 'done', according to our definition;
- WIP bottlenecks, too much work stuck at one point (Kanban)
- Where a team member may be overloaded, and holding up the iteration;
- Some add the column 'blocked', a simple set of columns is better than many;
- Used to drive the daily stand-up. Speeds it up as the information is clear;

Rather than manage timeline milestones, as in traditional project management, instead managing the queues of *work in progress* of the software developers.

Task Backlog – Start (what we are achieving and how)

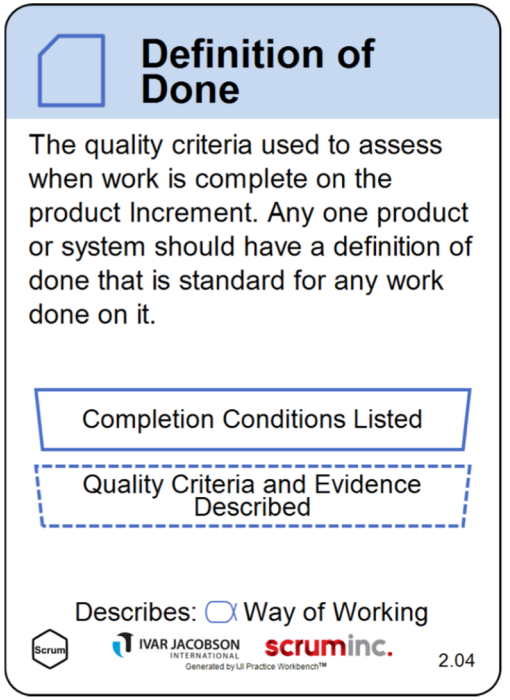


Task Backlog – A week later - In Progress

Objectives	To Do	Doing	Done
Software System **Usable** • System is usable and has desired quality characteristics • System can be operated by users • Functionality and performance have been tested and accepted • Defect levels acceptable • Release content known 3 / 6 **Work** **Under Control** • Work going well, risks being managed, productivity levels acceptable • Unplanned work & re-work under control • Work items completed within estimates • Measures tracked 4 / 6 **Team** **Performing** • Team working efficiently and effectively • Adapts to changing context • Produce high quality output • Minimal backtracking and re-work • Waste continually eliminated 4 / 5	Task 5 John	Task 3 James Task 4 Bill	**Requirements** **Satisfactory** • System implementing requirements is worth making operational • Enough requirements are implemented 5 / 6 Task 1 Bill Task 2 Fred

Example – List of Completion Conditions

- ✓ Story is functionally and non-functionally tested
- ✓ Story is reviewed by a peer
- ✓ Acceptance criteria are met
- ✓ It is deployed to appropriate environment
- ✓ Documentation is updated
- ✓ It is demonstrated and handed over in sprint review



Note: many of the cards define done for you.

Definition of Done

Standups (Daily Scrum)

Developers Meet Daily – The Daily SCRUM

The daily stand-up (scrum) is simple to describe:

The whole team meets every day for a very quick status update. We stand up to keep the meeting short.

That's it. But this short definition does not really tell you the subtle details that distinguish an effective stand-up from a waste of time. So how can you tell?

- **For experienced practitioners**, when things go wrong, they will instinctively know what to adjust to fix the situation. Most commonly the stand up becomes a meeting. Traditional project managers see the stand up as a way to control, manage and get reports; and find it difficult to let go of this approach. This is not a stand-up.
- F**or novices**, when things go wrong, it is much less likely that they'll figure out what to do and it's much more likely that, given no assistance, they will simply abandon the practice altogether. This is specially hard for traditional project managers that can't give up trying to run the show; to try to plan.

The stand-up is an internal meeting for the development team. The structure of the meeting is set by the development team, and they are responsible for conducting the meeting.
No one else manages or leads the meeting.

Yesterday, Today, Obstacles - The Three Questions

Some people are talkative and tend to wander off into **Story Telling**. Some people want to engage in **Problem Solving** immediately after hearing a problem. Meetings that take too long tend to have low energy and participants not directly related to a long discussion will tend to be distracted.

Therefore, one way to structure the contributions, is by using the following format:

1. **What did I accomplish yesterday to meet the sprint goal? - Inspect**
2. **What will I do today to meet the sprint goal? - Adapt**
3. **What obstacles are impeding my progress?**

The Task Board is used to focus the meeting and direct the team to the sprint goal and project objectives. The identifies and so everyone can see what they are working on.

- Other topics of discussion (e.g., design discussions, solutions, gossip, etc.) should be deferred until after the meeting. Planning and estimation are never done.
- What is quick or short ? **Generally each member should have 1-2 minutes, and the meeting go no longer than 15 minutes.** The stand-up is for the Development Team only. If others are present, they should not disrupt the stand-up and only act as spectators.
- The stand-up is used to reduce delays in resolving issues and obstacles so we meet the iterations objectives. It allows the whole team to act quickly.

Problems - when people attempt to work together

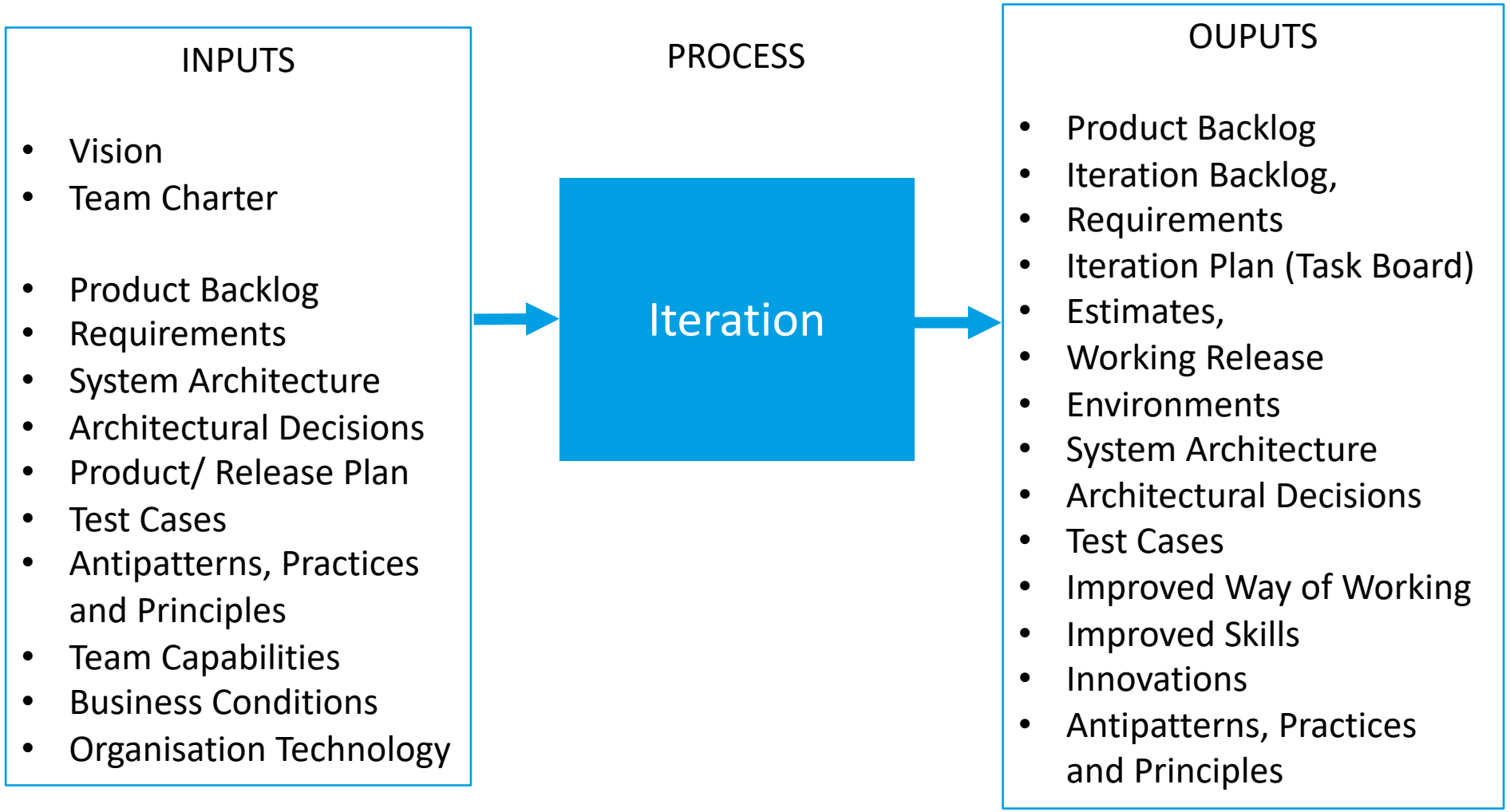
Stand-ups are a recurring solution to a particular set of problems that occur when a group of people attempt to work together as a team. They are a synchronisation mechanism, so that teams:

- **Share understanding of goals.** Even if we thought we understood each other at the start (which we probably didn't), our understanding drifts, as does the context within which we're operating. A "team" that is not working toward the same goals tends to be ineffective.
- **Coordinate efforts.** If you have a team, the work requires coordination. Poor coordination amongst team members tends to lead to poor outcomes. Importantly the team takes responsibility to coordinate, it is not led or directed by a 'manager' or 'boss'.
- **Share problems and improvements.** One of the primary benefits of a team is that the members can help each other out and discover better ways of working. A "team" where team members are not comfortable sharing problems tends to be ineffective.
- **Identify as a team.** It is very difficult to psychologically identify with a group if you don't regularly engage with the group and have no idea what people are doing. You will not develop a strong sense of relatedness even if you believe them to be capable and pursuing the same goals.

Artefacts

Iteration SIPOC

The Artefacts that are INPUTs (from Suppliers) to the Iteration and the Outputs (to Customers)



Inputs may come from Program Level (above)

Backlogs and Goals

The Sprint Backlog is composed of the **Sprint Goal (why),** the set of Product Backlog items selected for the **Sprint (what**), as well as an actionable plan for delivering the **Increment (how**).

- The Sprint Backlog is a plan by and for the Developers.

The Sprint Goal is the single objective for the Sprint. Although the Sprint Goal is a commitment by the Developers, it provides flexibility in terms of the exact work needed to achieve it.

The **Product Backlog** is an emergent, ordered list of what is needed to improve the product.

- It is the single source of work undertaken by the Scrum Team.

The **Product Goal** describes a future state of the product which can serve as a target for the Scrum Team to plan against.

- The Product Goal is in the Product Backlog.
- The rest of the Product Backlog emerges to define “what” will fulfil the Product Goal.

Lean 'Muda' or Waste – from Toyota Production System

One of the most important Lean principles is to reduce waste.

There are 8 main wastes, (adapted for IT):

- **Partially done work** – unfixed defects, incomplete documentation, code not tested, technical debt;
- **Extra features** – features included but not required, scope creep, suggests there is no MVP defined;
- **Re-learning** – knowledge not captured, poor or no documentation;
- **Hand offs** – knowledge is lost on each handoff;
- **Task switching** – multitasking is not a human capability; increases when team has too much Work in Progress (WIP), and reduces productivity;
- **Delays** – testing late, blocked work, waiting for approval;
- **Defects and re-work** – test early as possible, TDD, automated testing.
- **Skills** - waste of unused human talent and ingenuity. Allow team to innovate and self organise.

Are documents waste ?

Traditionally in Agile, documents are waste, as they are not the 'product' delivered to the customer. However, they have a place:

- **Knowledge Transfer** – speed up and facilitate learning, provide transparency;
- **Corporate memory** – maintain the knowledge gained, reduce cost of re-learning;
- **Software maintenance** – known documented test cases, requirements, architecture;
- **Remembering important decisions** - so they are not re-investigated;
- **Thinking before doing - c**reating documents does facilitate understanding;

The goal is to produce artefacts that are as **lean as possible**. So for example, no lower-level design specs, unless critical. These get out of date quickly and are too hard to keep up to date.

What is important is artefacts that are:

- Well defined, agreed and understood by the team; transparency.
- Their state is managed. The definition of "Done" is well understood;
- Knowledge rich;
- Traceable, for example, test cases trace back to scenarios;
- Developed JIT, to reduce the impact of change, and waste.

The principle: Express requirements as scenarios.

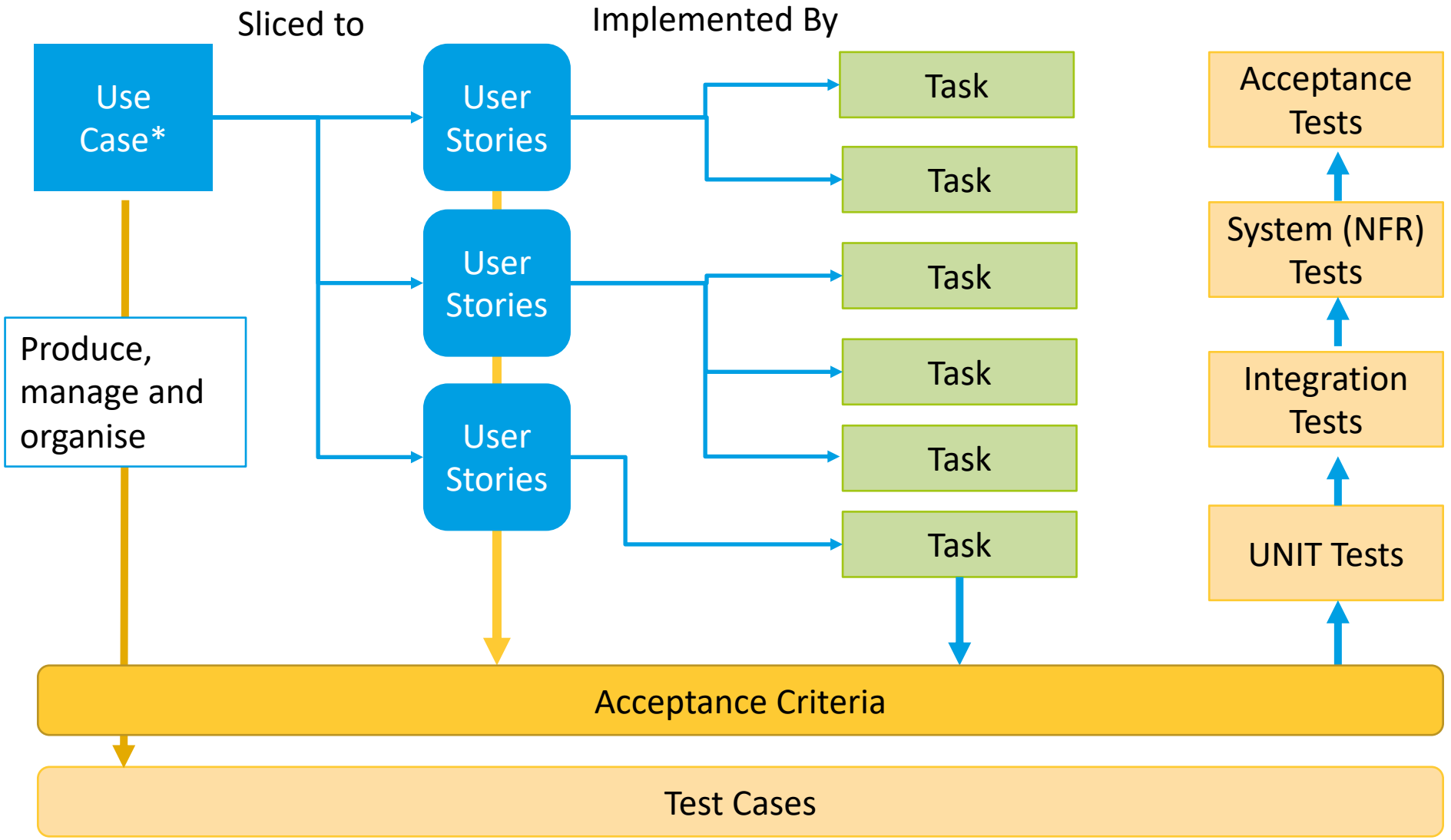
The 3 main types of requirements as scenarios are:

1. **Use Cases** – capture E2E User flows that focus on the value provided to customer - Replace Epics. Identify effective end to end test-cases. Eliminate the Feature Factory.
2. **Stories** – descriptions of a feature told from the perspective of the person that desires that new capability (The use case Actor)
3. **Non-Functional Requirements (**NFRs) - Use cases and stories can have specific NFRs.

All of these :

- have acceptance criteria, the conditions that a software product must satisfy to be accepted by a user, product owner, customer;
- drive testing and test cases, e.g. a use case may have 10 alternate flows, we would expect at least 11 test cases that trace to this requirement; Use cases effectively manage your test assets.
- are traceable - use cases (e.g. 12-30 days work) are broken down (sliced) into multiple stories. If a use case is sliced to just one story (2-3 days) then the use case is too simple;

Requirements drive Testing and Traceability



* Note: in traditional XP/SCRUM EPICs, are not developed into Use Cases.

An Example Use Case Template

Attribute	Description (use discretion in selecting attributes to use)
ID	A unique Identifier
Name	A short, descriptive name usually verb-noun
Description	A short summary description of the use case, 1-2 sentences
Outcome	The outcome for the user/customer.
Trigger	A business or operational event, condition or stimulus that starts the use case.
Actors	The source of the trigger. Only the ones initiating the use case (active)
Goal	A description of the goal of the Use case; what the actor is trying to achieve
Actions: Main Flow	The normal sequence of actions; the “happy path”
Actions: Alternate Flows	Exception and variation paths, alternate actions
Pre/Post-conditions	What must be true prior to starting / after the use case
User Stories	Links to the stories sliced from this use case
NFR	Link to generic NFRs or detail specific NFRs for this Use Cases
Time Frame	Required now or a future state
Priority	Business priority (high, medium or low)

An Example User Story Template

1) Title (one line describing the story)

2) Narrative:

- As a ……
- I want ……
- So that …….

As a	I want to	So that
Type of User (Role Actor)	Some Goal (Scenario)	Some Reason / Outcome / Benefit / Justification
Who	What	Why

3) Acceptance Criteria: (A List of Scenarios; conditions that must be met)

<u>Scenario 1:</u>

- Title
- Given [context]
  - And [some more context]...
- When [event]
- Then [outcome]
  - And [another outcome]...

<u>Scenario 2:</u>

……

……

Without Acceptance Criteria we never know if a Story is 'Done', it can't be tested, and quality is unknown.

It is the most important aspect of a story.

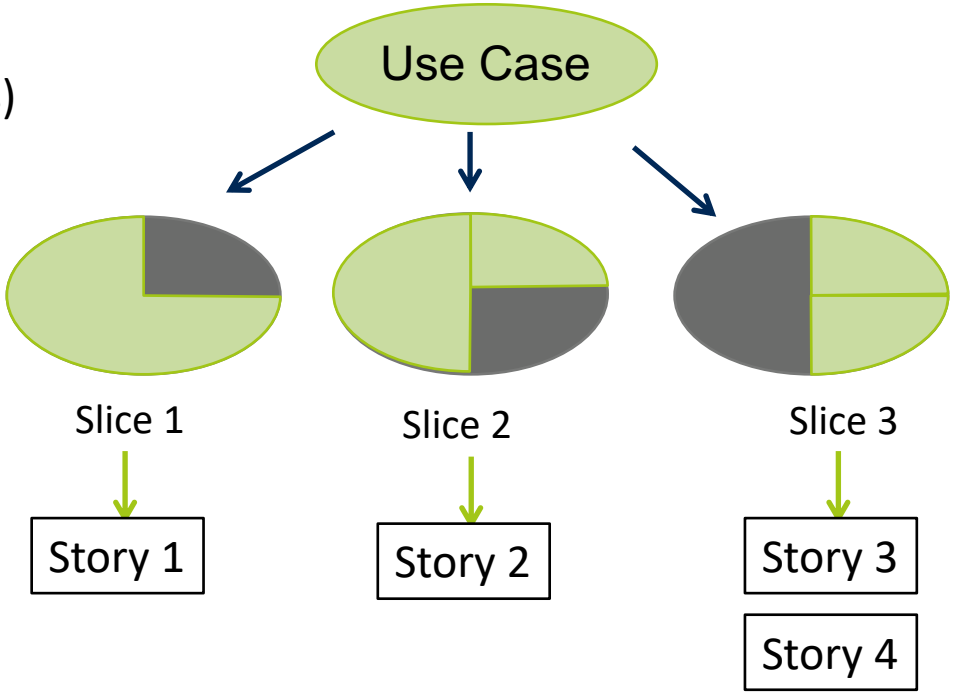
Slicing a Use Case

Slice by:

- Actors, roles, personas
- Each scenario (Main flow, alternate flows)
- Collections of actions
- Special requirements
- NFRs (e.g. scale from 10 users to 100)
- Etc

A Slice Includes:

- a story or use case
- its analysis counterpart,
- its design,
- its code,
- and its test.



Each slice forms one or more stories

Starting from a use-case model makes it significantly easier, compared to the traditional user story approach, to identify the right user stories to work with and to understand how the ones selected make up the whole system.

Agile Architecture

Agile Architecture – Emergent Vs Intentional

Agile Architecture is a set of values and practices that support the active evolution of the design and architecture of a system, *concurrent with* the implementation of new business functionality.

Enables incremental value delivery by balancing between two end points:

- ***Emergent Design*** provides the technical basis for a fully evolutionary, incremental implementation approach, and it helps the design respond to immediate user needs. The design *emerges* as the system is built and deployed.

- ***Intentional Architecture*** provides guidance and technical governance to Agile programs and teams for certain overarching technical constructs. These provide for commonality of approach, elimination of redundancy, and higher robustness of the full system.
  - Organizations must respond to new business challenges with larger-scale architectural initiatives that require some intentionality and planning.

Emergent Design is not enough

As Agile practices mature and are adopted by larger teams and teams of teams, there comes a point at which ***emergent design*** is an insufficient response to the complexity of large-scale system development, and these problems start to occur:

- Excessive redesign and delays; flow bottlenecks
- Different architecture constructs in support of the same capabilities, increasing maintenance costs
- Reduced collaboration and synchronization among teams
- Low velocity
- Systems that are difficult to integrate and validate
- Deterioration of system qualities (Nonfunctional Requirements, or NFRs)
- Low reuse of components; implementation redundancies

Both are needed:

- ***Emergent design*** enables fast, local control so that teams react appropriately to changing requirements without excessive attempts to future-proof the system.
- ***Intentional architecture*** provides the guidance needed to ensure that the system as a whole has conceptual integrity and efficacy.

Architecture Runway

Together, ***Emergent Design*** and ***Intentional Architecture*** create the ***Architectural Runway*** needed to enable teams to deliver business value faster and more reliably.

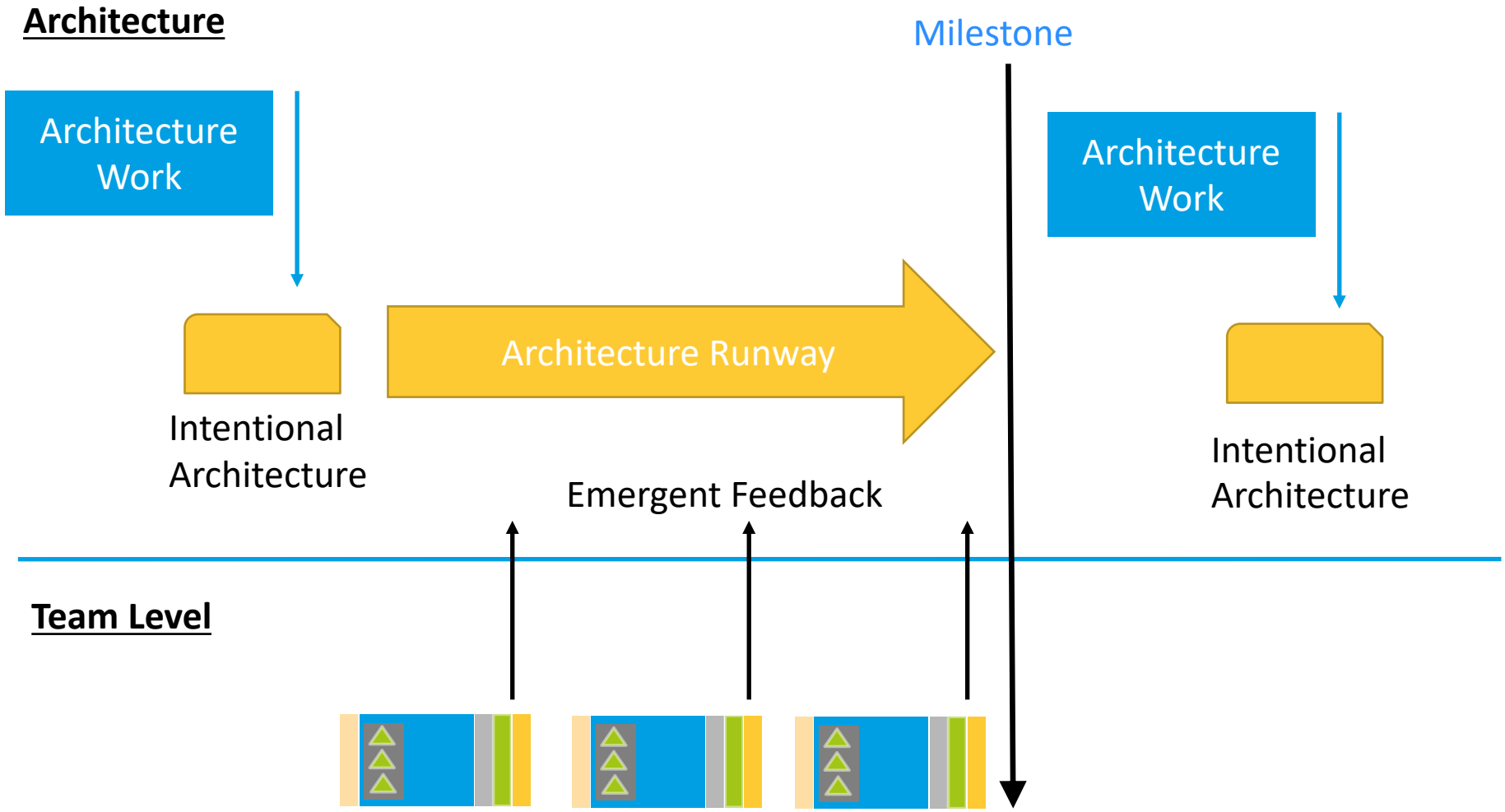
- An architectural runway exists when the enterprise's **platforms** have sufficient technological infrastructure to support the implementation of the highest-priority Features and Capabilities in the backlog without excessive, delay-inducing redesign.

- Enable programs to create and maintain large-scale solutions.
- To mitigate the risk, programs must take care to ensure that the necessary architectural underpinnings for the most innovative new features are *already* in the system when planning for the next Iterations.

- The ***Runway*** is just long enough to 'predict' the needs of the next Iteration. It removes the issue of big Upfront Architecture. It
- The ***Runway*** paves the way for the team to build effectively.

- *Build* some runway, *use* it, and *extend* it

Emergent Design Feedback to Intentional Architecture

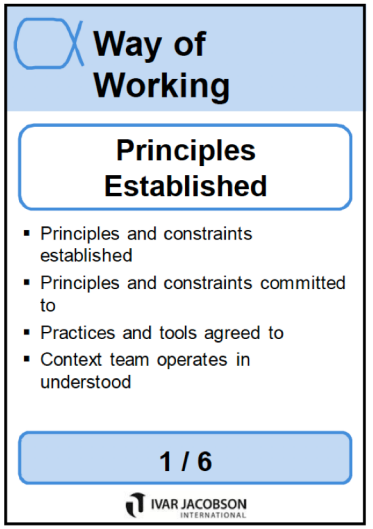


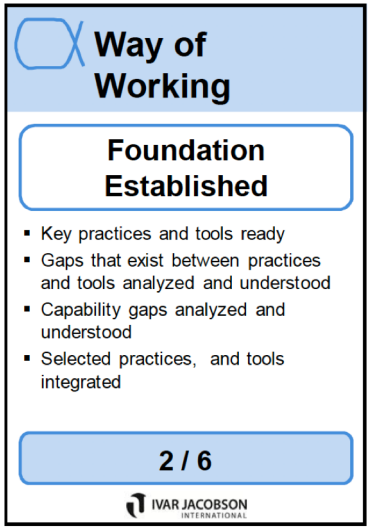
Improvement

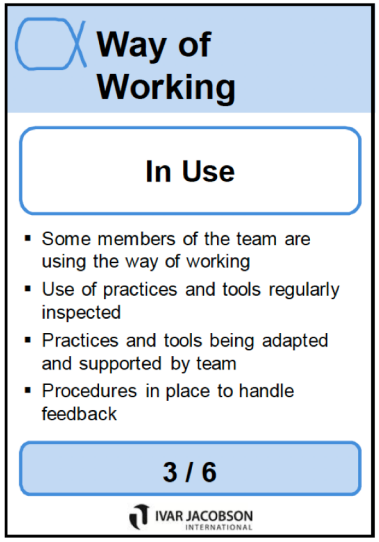
The States of the "Way of Working"

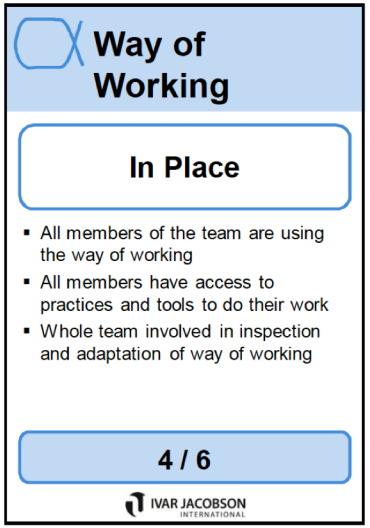
The aim of this pack is to outline how a team improved its way of working from non-existent/add-hoc to the goal of "Working Well".

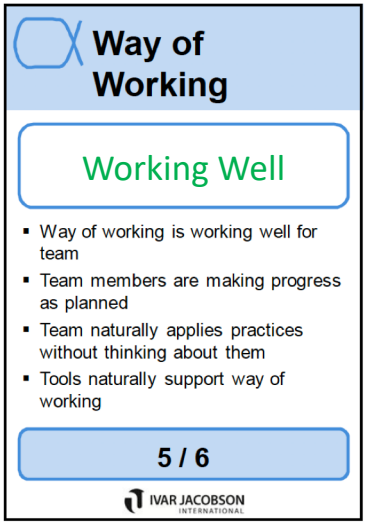
This is done via continual improvement, inspection and adaption, using the retrospective.







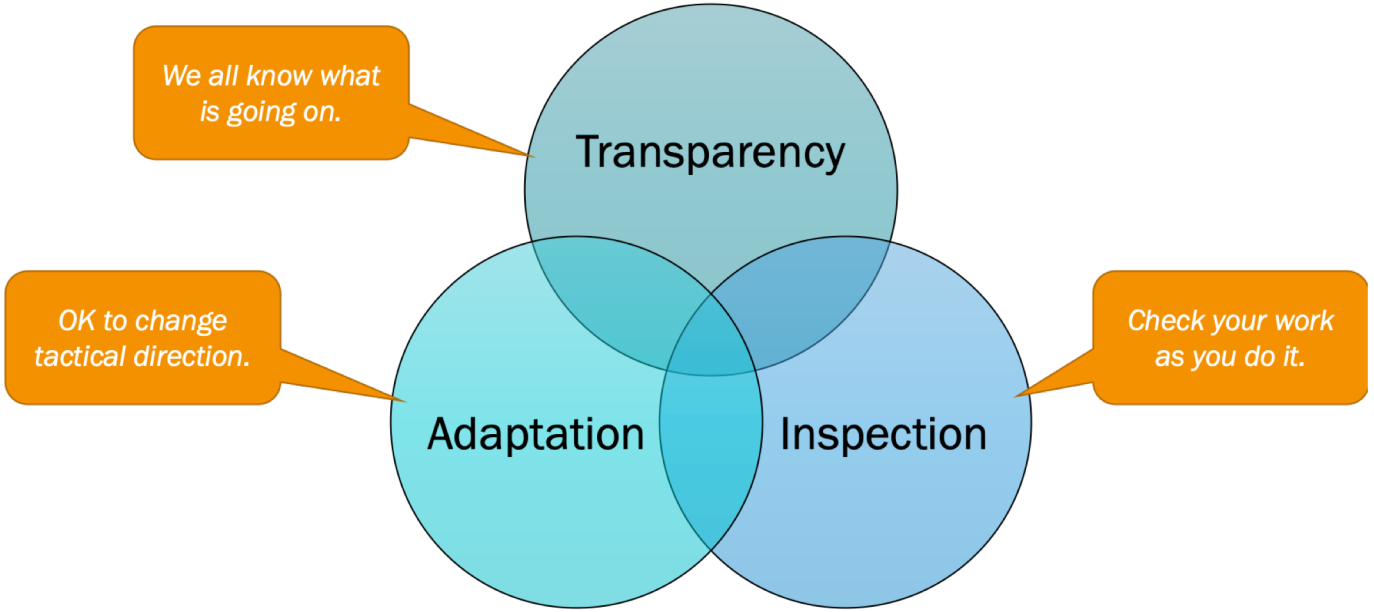




Improvement

SCRUM's Empiricism

"Empiricism means working in a fact-based, experience-based, and evidence-based manner. Scrum implements an empirical process where progress is based on observations of reality, not fictitious plans. Scrum also places great emphasis on mind-set and cultural shift to achieve business and organizational Agility. " Scrum.org



Rather than just "Empiricism", we used the Scientific Method (Karl Popper). New improvements are conjectures tested empirically, by experiment. Empiricism is not enough to provide new improvements, this a creative process. All conjectures are attempts to improve our explanations of why we do things.

Retrospective – Inspect and Adapt

The Toyota Way defines 2 important practices:

- **'Kaizen'** or continuous relentless process improvement. Become a learning organisation.
- **Thinking before doing**; planning up front (just enough) but responding to change. A little thought can avoid a lot of doing and re-doing.

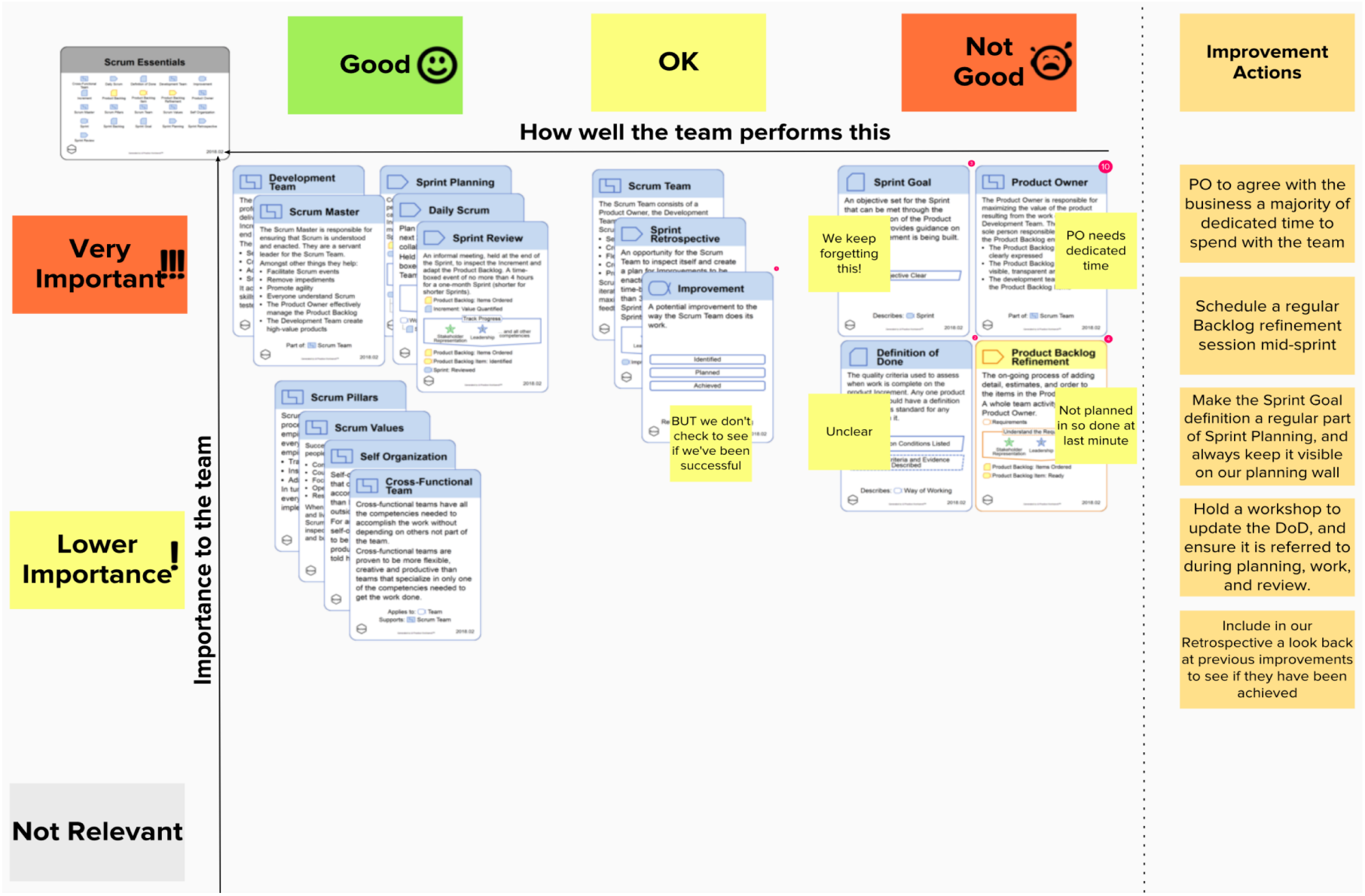
When we finish an iteration we inspect, reflect and improve, while the experiences are fresh in the team's mind, before starting the next iteration. If we must fix an issue, we may stop the line, '*Andon*', until resolved. **This is a time to review our beliefs, assumptions and values.**

What is discussed:

- What worked well, what we liked ?
- What worked just OK ?
- What didn't work ?
- Based on tangible cases and examples. Evidence and real data-driven, not opinions. Team may need to collect data before deciding on actions.

Sprint Retrospective

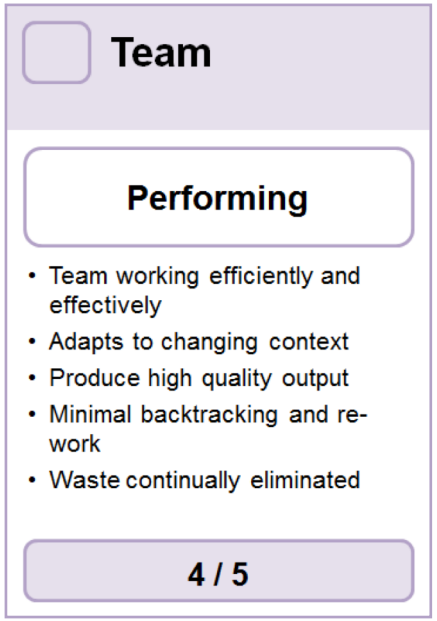
Using the SCRUM Cards to Explain

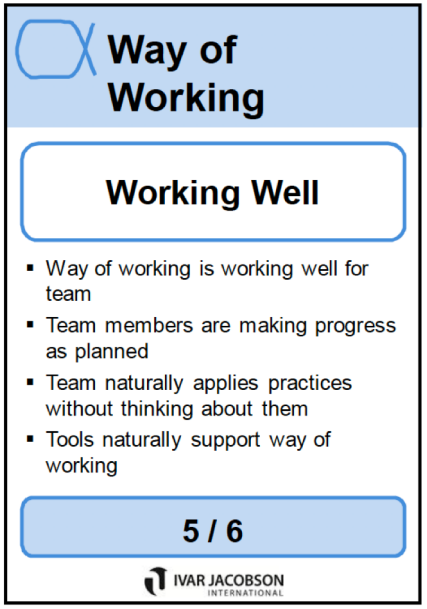


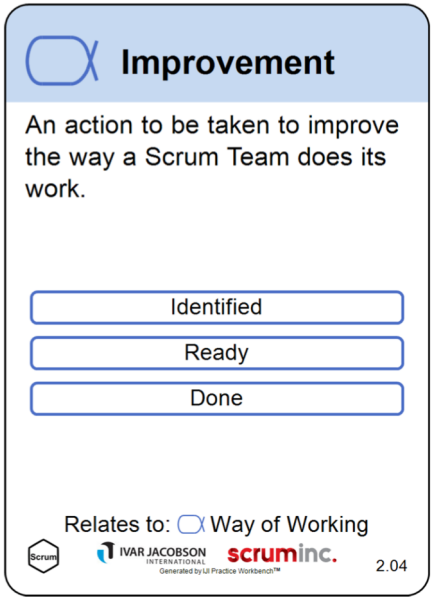
Retrospective – Use the Cards

Use the cards to aid the retrospective, **review the 7 core aspects: stakeholders, requirements, software system, the opportunity, the team, the way of working, and the work.**

- The goal is to get the Team to the "Collaborating" and "Performing" states by sharing beliefs and values. We want to improve our way of working to "Working Well".
- Improvement is 'Identified', planned, 'Ready' to be executed, then completed 'Done'



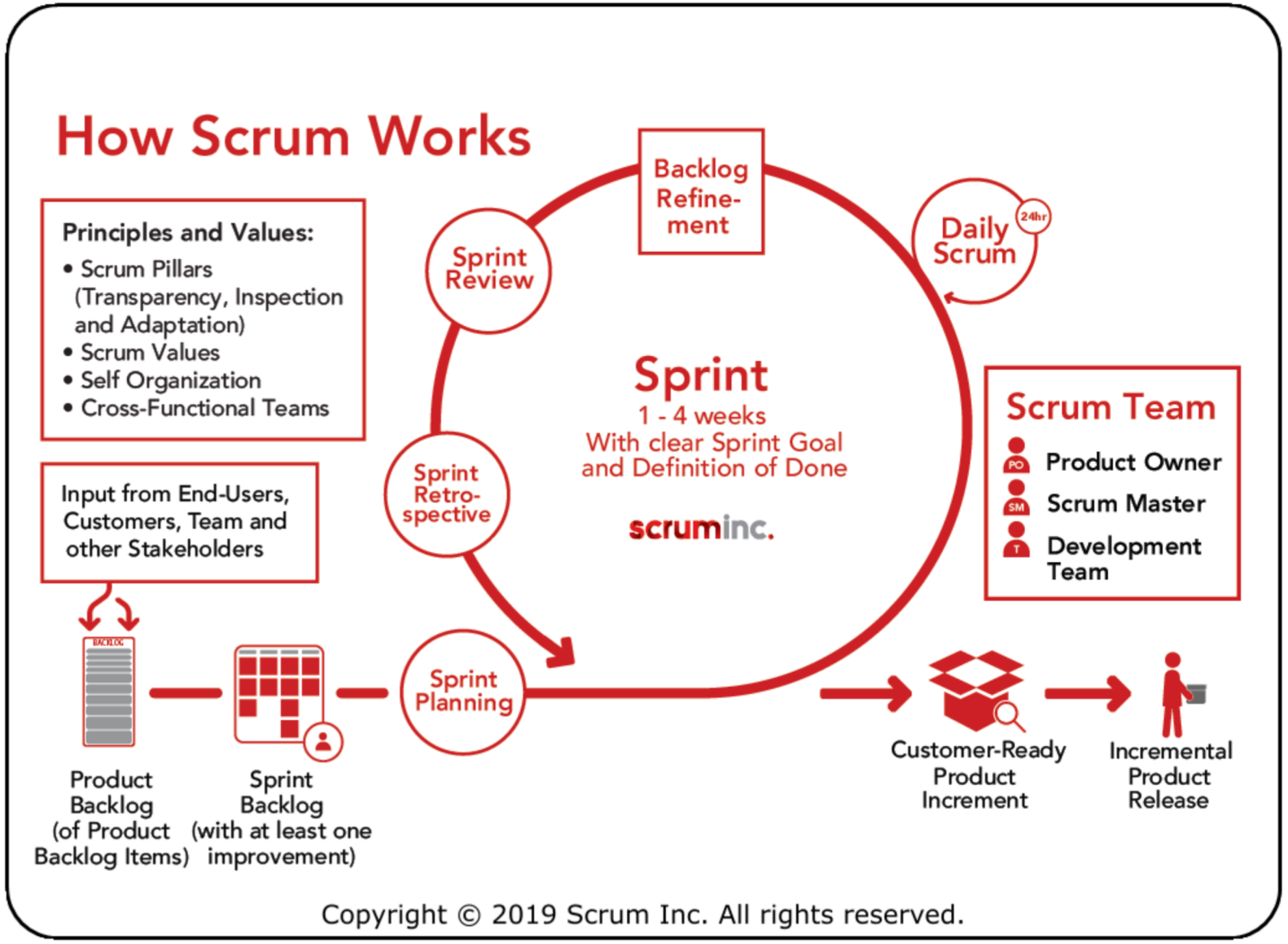




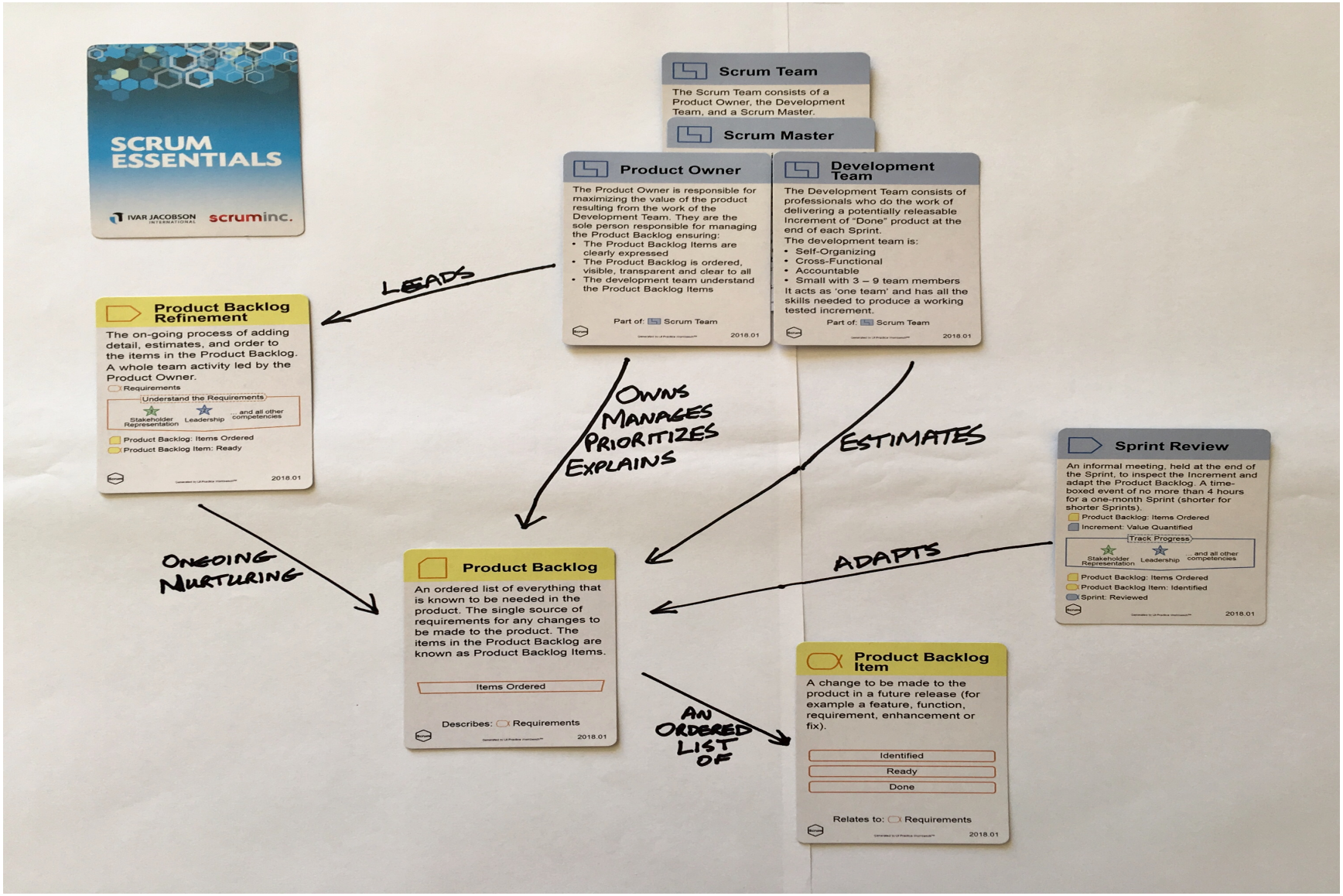
The retrospective updates the 'way of working'; new practices, principles, antipatterns and updated cards.

Finale

How SCRUM Works - Overview



The Cards help Explain Relationships; define the langauge



Why are we doing this ? Explanation is key

"A key driver of human progress is the search for better explanations. 'There is only one way of making progress: conjecture and criticism.' "

David Deutsch "The Beginning of Infinity"

If we assume that we want to 'improve' our business, not stick to the status quo, then improving how we build products is core. Reducing costs, improving quality, time to market, and quality of life is all things we should strive for. To make this progress we need 'ways of working' that can be explained and are open to criticism. We need to be able to make conjectures as to how we can improve, and explain why.

The starting point principle is having a 'way' that is described, open, transparent and can be objectively explained. If it is not described and must be learnt over the years by trial and error, then progress will be impossible There also needs to be a forum for criticism; the retrospective, and a way of running experiments. There needs to be describable explainable language elements or components that are the focus of criticism to provide improvement, these are provided by Essence cards.

The value of a described, written, ‘Way if Working’

“Writing organizes and clarifies our thoughts … writing enables us to find out what we know — and what we don’t know.”

William Zinsser

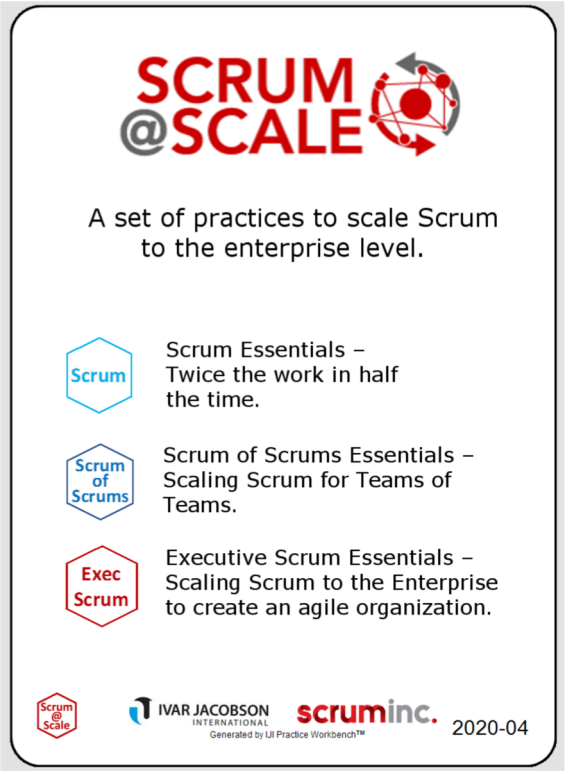
Conclusion

Having a 'described' and 'understood' Agile way of working:

- Improves the quality of life for the team and stakeholders;
- Improves the maturity and transparency of how we work;
- Improves team efficiency, the quality of the product and reduces cost;
- Provides a defined language, a way to communicate and explain the 'way or working', to improve and repeat success;
- Reduces stress and confusion;
- Reduces bugs and rework;
- Allows projects to adapt to change – provide business agility.

The next stages in evolution/improvement are:

- Moving from the concept of a project to a product stream or Flow*. The team is solidified and continually working on improving the product, and the way the product is made.
- Scaling agile teams, see https://pex.ivarjacobson.com/sites/default/files/practice/scrum_at_scale_cards.html
- Managing programs and the IT enterprise in an agile manner.



*see Flow in Appendix

The End

APPENDIX

Extra Material

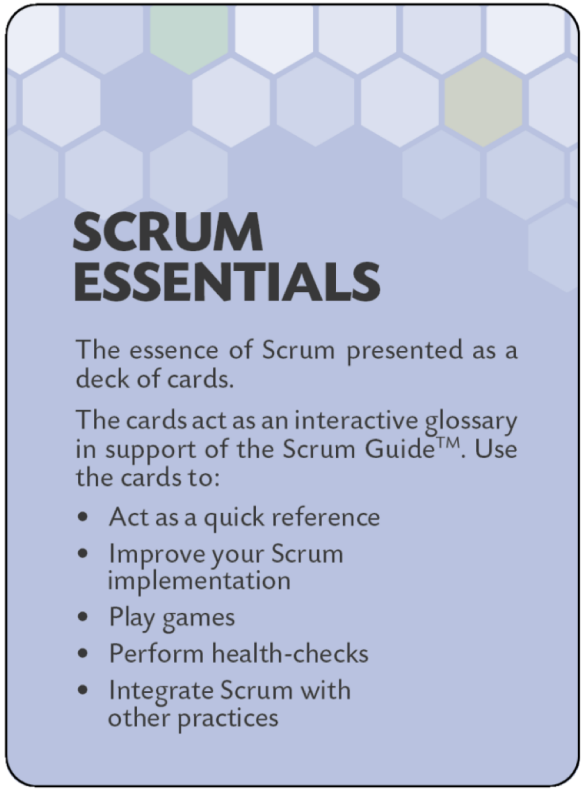
The appendix sections contain:

- SCRUM Essential CARDS
- ESSENCE Cards (not provided in pack earlier):
  - High Level cards
  - Stakeholder cards – other 6 are in slides previously.
- Anti-patterns – the team created a list of antipatterns out of retrospectives;
- Flow: “The Principles of Product Development Flow” by Don Reinertsen
- Collection of project Issue – a list compiled over the years;
- Agile project Issues.

SCRUM Essentials

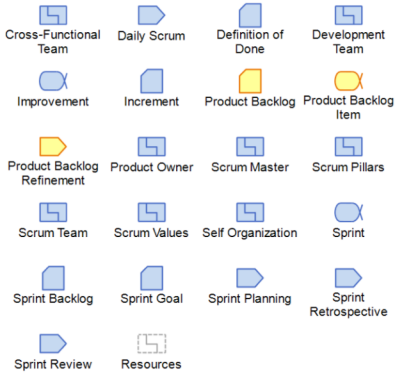
The SCRUM Essential Cards

Dr Jeff Sutherland – the co-creator of Scrum is quoted as saying that 58% of Scrum implementations fail! However, he also says: “Essence is the key to driving success” and “…the [Essence] cards have been used to have teams ‘Build Their Own Scrum’ ... One participant in this exercise using the Scrum cards said he learned more about Scrum in one hour with the Essence cards than he did in the previous six years of being on a Scrum team.

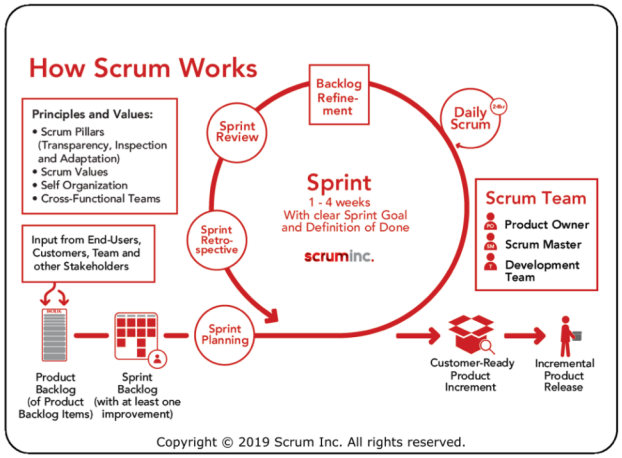


Scrum Essentials

Scrum is a framework for developing, delivering, and sustaining complex products.



Scrum IVAR JACOBSON INTERNATIONAL scruminc. 2.04



Cross-Functional Team

Cross-functional teams have all the competencies needed to accomplish the work without depending on others not part of the team.

Cross-functional teams are proven to be more flexible, creative and productive than teams that specialize in only one of the competencies needed to get the work done.

Applies to: Team
Supports: Scrum Team

Scrum IVAR JACOBSON INTERNATIONAL scruminc. 2.04

Daily Scrum

Plan and replan the work for the next 24 hours to optimize team collaboration and performance.

Held daily, this is 15-minute time-boxed event for the Development Team.

Sprint: Planned

Work: Under Control (contributes to)
Sprint Backlog: Forecast or beyond

Scrum IVAR JACOBSON INTERNATIONAL scruminc. 2.04

Definition of Done

The quality criteria used to assess when work is complete on the product Increment. Any one product or system should have a definition of done that is standard for any work done on it.

Completion Conditions Listed

Quality Criteria and Evidence Described

Describes: Way of Working

Scrum IVAR JACOBSON INTERNATIONAL scruminc. 2.04

Development Team

The Development Team consists of professionals who do the work of delivering a potentially releasable Increment of “Done” product at the end of each Sprint.

The development team is:

- Self-Organizing
- Cross-Functional
- Accountable
- Small with 3 – 9 team members

It acts as ‘one team’ and has all the skills needed to produce a working tested increment.

Part of: Scrum Team

Scrum IVAR JACOBSON INTERNATIONAL scruminc. 2.04

Improvement

An action to be taken to improve the way a Scrum Team does its work.

Identified

Ready

Done

Relates to: Way of Working

Scrum IVAR JACOBSON INTERNATIONAL scruminc. 2.04

Increment

The sum of all the Product Backlog Items completed during a Sprint and the value of the Increments of all previous Sprints. The Increment must be “Done” which means it must be in a usable condition and meet the Definition of Done.

Product Backlog Items Listed

Value Quantified

Describes: Sprint

Product Backlog

An ordered list of everything that is known to be needed in the product. The single source of requirements for any changes to be made to the product. The items in the Product Backlog are known as Product Backlog Items.

Items Ordered

Describes: Requirements

Product Backlog Item

A change to be made to the product in a future release (for example a feature, function, requirement, enhancement or fix).

Identified

Ready

Done

Relates to: Requirements

Product Backlog Refinement

The on-going process of adding detail, estimates, and order to the items in the Product Backlog.

A whole team activity led by the Product Owner.

Requirements

Understand the Requirements

3 Stakeholder Representation

2 Leadership

... and all other competencies

Product Backlog: Items Ordered

Product Backlog Item: Ready

Product Owner

The Product Owner is responsible for maximizing the value of the product resulting from the work of the Development Team. They are the sole person responsible for managing the Product Backlog ensuring:

- The Product Backlog Items are clearly expressed
- The Product Backlog is ordered, visible, transparent and clear to all
- The development team understand the Product Backlog Items.

Part of: Scrum Team

Scrum Master

The Scrum Master is responsible for ensuring that Scrum is understood and enacted. They are a servant leader for the Scrum Team.

Amongst other things they help:

- Facilitate Scrum events
- Remove impediments
- Promote agility
- Everyone understand Scrum
- The Product Owner effectively manage the Product Backlog
- The Development Team create high-value products

Part of: Scrum Team

Scrum Pillars

Scrum is founded on empirical process control theory, or empiricism. Three pillars uphold every implementation of empirical process control:

- Transparency
- Inspection
- Adaptation

In turn they are the foundation of every successful Scrum implementation.

Applies to: Team and Way of Working

Scrum Team

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master.

Scrum Teams are:

- Self organizing
- Cross-functional
- Flexible
- Creative
- Productive

Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback.

Applies to: Team

Scrum Values

Successful use of Scrum depends on people living the five Scrum Values:

- Commitment
- Courage
- Focus
- Openness
- Respect

When these values are embodied and lived by the Scrum Team the Scrum Pillars of transparency, inspection and adaptation come to life and build trust for everyone.

Applies to: Team

Self Organization

Self-organizing teams are teams that choose how to best accomplish their work, rather than being directed by others outside the team.

For any form of complex work, self-organizing teams are proven to be more flexible, creative and productive than teams that are told how to do their work.

Applies to: Team
Supports: Scrum Team

Sprint

A time-box of one month or less during which a “Done”, useable and potentially shippable Increment is created. A new Sprint starts immediately after the conclusion of the previous Sprint.

Scheduled
Planned
Reviewed

Relates to: Work

Sprint Backlog

The set of Product Backlog Items selected for the Sprint, plus a plan for delivering the Increment and realizing the Sprint Goal. It includes at least one Improvement identified at the last Sprint Retrospective.

It makes visible all of the work the Development Team identifies as necessary to meet the Sprint Goal.

Forecast
Other Detail Captured

Describes: Sprint

Sprint Goal

An objective set for the Sprint that can be met through the implementation of the Product Backlog. It provides guidance on why the Increment is being built.

Objective Clear

Describes: Sprint

Sprint Planning

Collaboratively plan the work to be performed in the Sprint and agree what can be delivered in the Sprint’s Increment. A time-boxed event of no more than 8 hours for a one-month Sprint (shorter for shorter Sprints).

Product Backlog: Items Ordered
Sprint: Scheduled

Coordinate Activity
3 Leadership
2 Management
... and all other competencies

Sprint: Planned
- Sprint Backlog: Forecast or beyond
- Sprint Goal: Objective Clear

Sprint Retrospective

An opportunity for the Scrum Team to inspect itself and create a plan for Improvements to be enacted in the next Sprint. A time-boxed event of no more than 3 hours for a one-month Sprint (shorter for shorter Sprints).

Support the Team
3 Leadership
2 Management
... and all other competencies

Improvement: Ready or beyond

Sprint Review

An informal meeting, held at the end of the Sprint, to inspect the Increment and adapt the Product Backlog. A time-boxed event of no more than 4 hours for a one-month Sprint (shorter for shorter Sprints).

Product Backlog: Items Ordered
Increment: Product Backlog Items Listed or beyond

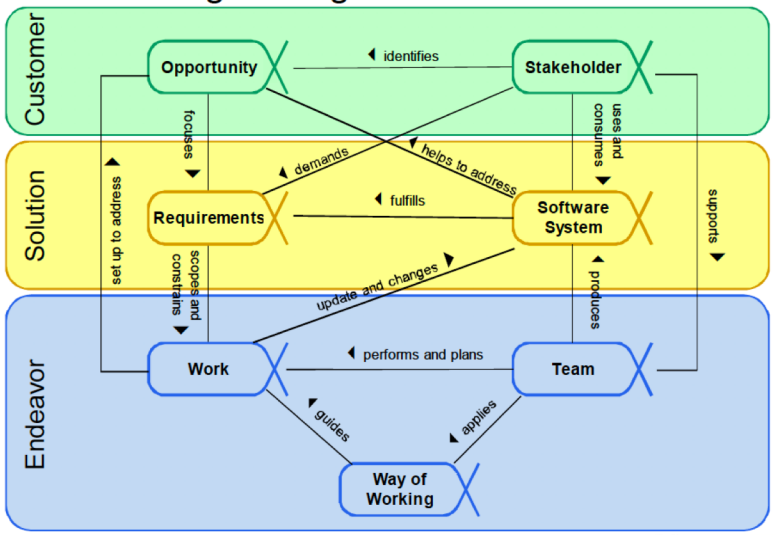
Track Progress
3 Stakeholder Representation
3 Leadership
... and all other competencies

Product Backlog: Items Ordered
Product Backlog Item: Identified
Sprint: Reviewed

Base Essence Cards

(not already in pack)

Software Engineering Kernel



IVAR JACOBSON INTERNATIONAL

Stakeholders

- Recognized
- Represented
- Involved
- In Agreement
- Satisfied for Deployment
- Satisfied in Use

The people, groups, or organizations who affect or are affected by a software system.

- Healthy stakeholders represent groups or organizations affected by the software system
- Healthy stakeholder representatives carry out their agreed to responsibilities
- Healthy stakeholder representatives cooperate to reach agreement
- Healthy stakeholders are satisfied with the use of the software system

IVAR JACOBSON INTERNATIONAL

Opportunity

- Identified
- Solution Needed
- Value Established
- Viable
- Addressed
- Benefit Accrued

The set of circumstances that makes it appropriate to develop or change a software system.

- A good opportunity is identified addressing the need for a software-based solution
- A good opportunity has established value
- A good opportunity has a software-based solution that can be produced quickly and cheaply
- A good opportunity creates a tangible benefit

IVAR JACOBSON INTERNATIONAL

Requirements

- Conceived
- Bounded
- Coherent
- Acceptable
- Addressed
- Fulfilled

What the software system must do to address the opportunity and satisfy the stakeholders.

- Good Requirements meets real needs
- Good Requirements has clear scope
- Good Requirements are coherent and well organized
- Good Requirements help drive development

IVAR JACOBSON INTERNATIONAL

Software System

- Architecture Selected
- Demonstrable
- Useable
- Ready
- Operational
- Retired

A system made up of software, hardware, and data that provides its primary value by the execution of the software.

- Good Software System meets requirements
- Good Software System has appropriate architecture
- Good Software System is maintainable, extensible and testable
- Good Software System has low support cost

IVAR JACOBSON INTERNATIONAL

Team

- Seeded
- Formed
- Collaborating
- Performing
- Adjourned

The group of people actively engaged in the development, maintenance, delivery and support of a specific software system.

- A healthy Team meets its team goals effectively
- A healthy Team has members that collaborates effectively
- A healthy Team focus on their work
- A healthy Team continually improves

IVAR JACOBSON INTERNATIONAL

Work

- Initiated
- Prepared
- Started
- Under Control
- Concluded
- Closed

Activity involving mental or physical effort done in order to achieve a result.

- Healthy Work is sizeable, estimate-able and track-able
- Healthy Work breakdown reduces dependencies between work items
- Healthy Work management keeps risks, work and re-work under control

IVAR JACOBSON INTERNATIONAL

Way-of-Working

- Principles Established
- Foundation Established
- In Use
- In Place
- Working Well
- Retired

The tailored set of practices and tools used by a team to guide and support their work.

- Good way of working is agreed by the team
- Good way of working reduces risks and technical debts
- Good way of working is effective and removes duplicate work and wastes
- Good way of working improves itself

IVAR JACOBSON INTERNATIONAL

Stakeholder State cards

Stakeholders

Recognized

- Stakeholders have been identified
- There is agreement on stakeholder groups to be represented
- Responsibilities of stakeholder representatives defined

1 / 6

IVAR JACOBSON INTERNATIONAL

Stakeholders

Involved

- Stakeholder representatives carry out responsibilities
- Stakeholder representatives provide feedback & take part in decisions in timely way
- Stakeholder representatives promptly communicate to stakeholder group

3 / 6

IVAR JACOBSON INTERNATIONAL

Stakeholders

Satisfied for Deployment

- Stakeholder representatives provide feedback on system from their stakeholder group perspective
- Stakeholder representatives confirm system ready for deployment

5 / 6

IVAR JACOBSON INTERNATIONAL

Stakeholders

Represented

- Stakeholder representatives appointed
- Stakeholder representatives agreed to take on responsibilities & authorized
- Collaboration approach agreed
- Representatives respect team way of working

2 / 6

IVAR JACOBSON INTERNATIONAL

Stakeholders

In Agreement

- Stakeholder representatives agree their input is valued and respected by the team
- Stakeholder representatives agree with priorities
- Stakeholder representatives have agreed upon minimal expectations for deployment

4 / 6

IVAR JACOBSON INTERNATIONAL

Stakeholders

Satisfied in Use

- System has met or exceed minimal stakeholder expectations
- Stakeholder needs and expectations are being met

6 / 6

IVAR JACOBSON INTERNATIONAL

Anti-Patterns

Anti-Patterns – Process

Anti: Process is a negative thing

- **Pattern:** The organisation views process as negative. It is seen to add unnecessary delays and effort; bureaucracy. It is only required when things don't work. Developers see it as cramping their 'creative' style. Managers fear that following a defined process will slow the project down, and reduce business agility, steps must be followed despite not providing value.
- **Solution**: Inappropriate processes and following them in a dogmatic fashion, does not add value or allow for variations in projects and people.

  The process needs to be built by the team based on what they need, and experience. The fundamental objective of process improvement is to reduce the cost of developing and maintaining software, improve product quality, improve the team's quality of life, and to achieve business value. It's not to:

  - generate a shelf full of processes and procedures.
  - comply with the dictates of the most fashionable process improvement model or framework.

  Rather than process focus on 'practices'. Retrospectively improve the practices by inspection and adaption. Team is responsible for its description.
- **Principle 6:** Team manages its "Way or Working"; it is continually improving, described, open, transparent

Anti-Patterns – No Goals or Objectives

Anti: The Team is Lost

- **Pattern:** The team does not see how the work, they are directed to do, fits into the big picture, and long-term direction is unclear. The senior management and product team may know the strategy and product direction, but its not communicated. The product goals are unknown or obscure.
- **Solution**: The Iteration Plans fit into a Product Plan that delivers the vision and objectives. The Product Task Board ("wall") is used to show the major milestones and how they meet the project objectives and goals, and how the Iteration work fits in. The product owner is needs to define these goals and one of their functions is to bridge this gap to align the team on the business goals, explain the 'why' behind the product.

Anti: Tasks and Objectives are not visible

- **Problem**: The tasks are in obscure documents that are emailed around. Their text is not clear and is ambiguous; they are not discussed. The objectives are discussed in higher level meetings that the team are not privy too.
- **Solution**: The objectives and tasks are made visible and tangible in a Task Board. The goals of the sprint are agreed before hand. The product owner contributes by communicating regularly the expectations of the organization and the strategy and objectives, guiding the team to produce the best product.

Anti-Patterns - Planning

Anti: Plan and estimate every day

- **Problem**: The task goals change frequently, and the team is asked to plan every day on the fly. They don't have the information at hand or understand the dependencies. Daily meetings get longer and more complicated, and confusing.
- **Solution**: The iteration has a focused scope; an agreed goal. Planning and estimation is allocated specific dedicated time, at the start of the iteration, and never in the stand-up. The tasks should not change within the iteration, there should be no need for detailed re-assessment, and on the fly last minute planning;
- **Principle**: Freeze requirements during the iteration

Anti: Agile does not need plans

- **Pattern:** The team does not plan but tackles each iteration in an add-hoc manner. The purpose of the iteration is just to 'hack' in a fixed delivery timeframe.
- **Solution:** Agile does require planning, at several levels. Importantly at the product level and then at the Iteration level. The iteration plan has dedicated team time to decide the goals and estimates for the work items. These are scheduled and dependencies worked out. The deliverables are defined. The next iteration is also planned but in a much less detailed manner; a view to the future.

Anti-Patterns - Autonomy

Anti: Autonomy will not work

- **Problem**: The management of the business do not believe the team can manage itself. Having a top-down structure where project management / waterfall predominates and the members are micromanaged is critical as its not possible to trust the team to know what to do.
- **Solution**: The most crucial organizational dynamics moving autonomous teams toward successful outcome implementations include organization-wide commitment to using autonomous teams, upfront and unwavering commitment to allocating resources for team use, and frequent, face-to-face feedback. Trust the team and give them autonomy.
- **Principle**: Let the team self-organize; desire autonomy

Anti: Command and Control

- **Pattern**: One dominant member starts taking control of the project. In some organizations a project manager is allocated to lead, rather than facilitate (scrum master).
- **Solution**: This is the opposite goal of Agile, where teams are self organising. Empower the team to self-organize; be autonomous. The team shares leadership.

Ant-Patterns – Stand-up

Anti: Team is not Leading its Standup

- **Pattern:** Someone takes lead of the standup, they decide who talks, they ask people what they have done, and to detail their tasks. They becomes a 'boss'. Worse still they sit down while the team stands.
- **Solution:** The standup is for the developers to share not report. The team members volunteer their work and stick to the 3 Questions, it is a self organizing endeavor. Use a physical object (e.g. stuffed toy) and pass on to next member, it signals who has the floor to the team. It makes a team member decide who talks next. Everyone leads and manages. This empowers the team members. There should be leaders and not bosses.
- **Principle**: Everybody all together from early on

Anti: A Member Bloviates

- **Pattern:** A member goes off into a long rambling rant about their work. They introduce complex technology descriptions and all the bugs they have in detail. What's worse is only one other team member may be directly interested. The team is not impacted by these issues, their eyes glaze over, no one can take in the technical complexity. The teams time is being wasted.
- **Solution:** the team asks them to stick to time limit, and take it offline with the relevant member(s)

Ant-Patterns : Stand-up

Anti: Non-Contributors

- **Pattern:** People attend the stand-up, but they don't contribute to the work each day, and don't ever provide the 3 Questions. Worse still they suggest issues or ask questions interrupting another members who have the floor.
- **Solution:** They should remain silent and ask questions after the meeting. Maybe only attend when they can offer value. The standup is an internal meeting for the development team. The standup is for people actively contributing to the work and collaborating.

Anti: Wrong Content

- **Pattern:** The standup becomes a long project management meeting, the meetings go beyond the allocated time, tasks are discussed, planned, estimates asked for, and issues raised, and solved. The focus has been lost; the iteration objectives are no longer being focused on. Complex tasks are discussed out of context and the team members are not prepared. No one has the detail required at hand. It wastes everyone's time and produces poor quality result.
- **Solution:** The iteration is fixed; no new scope can be added. These discussions go into the iteration planning and estimation sessions. Other discussions are held outside the standup. If the sprint has lost its purpose it should be cancelled.

Anti-Patterns : Stand-ups

Anti: Indecipherable Content

- **Pattern:** In the stand-up an endless stream of Jira task numbers are discussed, or shortened task names, with no understandable 'English' description, no visual guidance. No goals.
- **Solution:** team members discuss their task as visualized on the Task Board or Wall. Members see the work, and task descriptions, and don't get glazed over my meaningless ticket numbers. The goals are made explicit. The team members see who is assigned to a task, so this does not need to be discussed, and wasted time is reduced.

Anti: Nothing to Say but talks anyhow to fill their allocated time

- **Pattern:** A member has had no change from yesterday, but talks anyhow at length
- **Solution**: simply say 'no change'

Anti: Jira Handoffs

- **Pattern:** Jira is used to bounce work from one member to another. This is a handoff; waste. This introduces delays and confusion, as the team tries to understand what/s been noted in the jira ticket. Members may not see the ticket for days.
- **Solution**: Issue are raised in the standup and address face to face in a separate meeting. The tasks or roles are poorly defined, redefine to remove handoffs.

Anti-Patterns : Planning and Control

Anti: The Product Owner Rules

- **Pattern:** The product owner feels they are above the team and can direct it; they take on a command-and-control role. They want to define the solution and the dev team is there to deliver what they mandate. The product owner refers to the team as ‘them’ in other meetings. The product owners shows up infrequently.
- **Solution**: The product owner is a member of the team; they are not above the team but a peer. The product owner needs to refers to the team as ‘we’, and not ‘them’. The product owner's unique role is to bring the ‘problem’ to the team, and the whole team works out the ‘solution’. The product owner needs to attend all standups. If they don’t attend, then delay results as they get up to speed from other team members, wasting their time.

Anti: Top-Down planning and prediction

- **Pattern**: The system has changed from what was planned, and stakeholders can’t see how any planning was right, and question the changed solution and architecture.
- **Solution**: Emergent design is a reality; the system grows in ways we did not expect. This is balanced with Intentional design. Long term plans will be incorrect. The architecture evolves and changes, incrementally the goals are met.

Anti-Patterns : Leadership

Anti: Leadership is a given

- **Pattern**: The team is moving from a project managed approach to Agile, and they are having some issue with members not leading and still expecting to be lead. The management team does not see the team doing well at leading and expected dramatic improvements from Agile from day one.
- **Solution**: leadership takes practice and can't be switched on instantly. The team will improve over time and adapt as they learn to lead. The team needs to be able to fail safely to learn new ways. Changes are experiments and some may not work out.

Anti: Teams are Managed via Command and Control.

- **Pattern**: a manager or leader defines the work for the team, its structure, and way they work. The team is managed not managing, they are not allowed to lead. This dis-empowers the team. "Command and Control" is not compatible with Agile. When combined with micromanagement the team becomes dis-engaged.
- **Solution**: Let the team self-organize, they decide on their own tasks. A corollary of this is that empowerment of the team is a severe curtailment of managers responsibilities and in many organisations this is just not the way things are done. Servant manager addresses this issue to some degree.

Anti-Patterns : Feature Factory Mentality

Anti: Feature Factory

- **Pattern**: The product is defined as a set of features that are to be delivered. The team does not understand how the features flow, and why they are required. It is not clear how they fit together and provide value to a user.
- **Solution**: Features are a view on the requirements, and not requirements. Scenarios provide the who, what and why needed. Use cases provide the overall context that binds together the stories to provide an end-to-end flow; they refer to features in use. Stories are not necessarily defined requirements and use cases help manage this. Use cases show how the user gains real value from the system. See walk up and walk away test.

"The requirement itself is communicated from customer to programmers through conversation: an exchange of thoughts, opinions, and feelings. This conversation takes place over time, particularly when the story is estimated (usually during release planning), and again at the iteration planning meeting when the story is scheduled for implementation. " *Ron Jeffries*

"Stories are not intended to document user needs. The story is a means to having a conversation." Mike Cohen

Anti-Patterns : Epics, Stories, Tools

Anti: Epics are Used to group Stories

- **Pattern**: Epics are created either upfront before any stories are written, or by grouping existing stories.
- **Solution**: This approach will impede vertical slicing, e.g. 'Front-end', 'Back-end', 'Security'. No epics are created upfront. Stories are written uninfluenced by epics. Stories found to be too big become epics for later slicing. Alternatively group by use cases. Epics are not essential to Agile.

Anti: Jira is used to report by Epics

- **Pattern**: A big epic is not completed, although many stories it 'groups' are done. Software is not delivered. As the Epic is not completed no progress can be reported.
- **Solution**: Report and deliver increments by Story. Epics are not deliverable.

Anti: Tools influence the way you work

- **Pattern**: we are forced to report by Epics as Jira does this, so we create 'fictional' Epics to group Stories
- **Solution:** tooling should not influence the way you work. Understand the influence of software tools on the team. Should work be allowed to evolve independently of a software tool, or be constrained by it?

Anti-Patterns : Test and Acceptance Issues

Anti: No Acceptance Criteria

- **Pattern**: The user stories do not have acceptance criteria, or ones that are too trivial.
- **Solution**: The result is that is is not possible to know if the system is completed; done. It is not possible to determine if the requirements have been delivered. The acceptance criteria must be provided, well thought through and exercise all aspect of the story and be agreed with stakeholders. Acceptance criteria are essential to provide good test cases and help determine 'Done'.

Anti: Unmanaged Test Cases

- **Pattern:** The test cases are not documented, managed or known; they are add-hoc. The cases are not documented but in someone's head and do not trace back to requirements.
- **Solution**: This situation means that the state of the system is unknown and can never be known. Functionally it cannot be determined if the system is 'Done', and quality cannot be determined. Change becomes inefficient, as a change in one requirement does have specific tests, but most likely the whole set of tests, or worse still the wrong test cases. Tracing test cases to requirements is essential for iterative development. Use cases provide a core tool for managing and generating end to end test cases. Having well managed test cases means they are a re-usable asset, by other teams, other members and for maintenance. Test case management tools, such a X-RAY are essential.

Anti-Patterns : QA and Late Testing

Anti: Separate QA Team Tests later

- **Pattern**: The developers build a release and hand it off to a separate QA team. QA finds a bug and hands a task back to developers that hand it back to QA to test. This handoff cycles until there are no bugs, or it just gets confused or out of control. Each step is a delay. The tests by QA cover only the normal flows, the way we would like the ideal product to work, but alternate, error flows, don't get exercised.
- **Solution:** Hand offs are a type of waste. On each handoff knowledge is lost. Everyone does QA, there are no hand-offs. Testing at the end of the process is a delay, and must happen from early on. Test Driven Development and Unit testing are critical to catch bugs early. Non-functional testing is automated. The QA team may not be aware of complexity of the system when things don't go right, and unless their test cases are derived from use case scenarios, the alternate flows, and stories, testing will not be adequate or traceable.

Anti: Separate QA Team Questions Requirements

- **Pattern**: The QA team tests the solution, but it does not meet their concept of what's required. They question the requirements, and the delivered product.
- **Solution:** QA and testing starts at the beginning, not at the end. The product is for the customer to meet their outcome, not the QA teams, and this is guided by the product owner. Test the product with the customer, and base change on their feedback.

QA is pervasive and starts at the very beginning.

"If quality assurance is the be implemented at the very beginning of the development stage of a new product, it means all divisions of a given company and all of its employees must participate in Quality Control….

Quality assurance must ultimately reach this third phase of development, which is implementation of quality assurance at the earliest stages of product development."

"What is Total Quality Control", Kaoru Ishikawa

Anti-Patterns: Product Owner

Anti: Sprint Goal is poorly defined

- **Pattern:** The Sprint backlog resembles a random assortment of tasks, and no Sprint goal is defined. The backlog is add hoc or no where to be seen. The product owner cannot provide a Sprint goal, or the chosen Sprint goal is flawed.
- **Solution**: The Product Owner needs to determine the product direction and goal. They need to ensure the quality of the product backlog. They may be too influenced by other stakeholders determining the direction. The product owner needs to say 'no' more often to stakeholders.

Anti: Add More Work

- **Pattern**: The product owner tries to squeeze in some last-minute user stories that are not ready or satisfactory. They try to add in more tasks than the team cab handle
- **Solution**: the product owner does define the high priority work for the product. However, if this happen frequently, it indicates that the product owner needs help with prioritization and team communication. The team works together to determine what is feasible, and they determine if they can commit to the goal. The development team produce the plan to deliver on its commitment collaboratively.
- **Principle:** Work at a sustainable pace

Anti-Patterns: Product Owner

Anti: The product owner runs the Standup

- **Pattern:** The product owner creates a list of issues and then creates a list of actions, each day, for the team to carry out, and then manages the standup.
- **Solution**: The Product Owner needs to determine the product direction and goal, however they do not run the standup, or manage the teams activities. They are not above, but at the same level as the developers. They do not manage or micromanage. The standup is an internal meeting for the development team. Others including the product owner, should not disrupt it. The product manager is an optional attendee of the standup.

Anti: The product owner gets involved in standup

- **Pattern**: The product owner gets involved in the standup and disrupts developers.
- **Solution**: Product Owners who can attend the standup may add value, but they are a guest. By being available they may be able to directly answer the development team's questions to unblock their work and have important additional information about the business context. Product owners should contribute quickly or wait till after to share their thoughts. Product owners should attend the standup to reduce the us and them perception. If they don't attend, they can still reach out afterwards.

Anti-Patterns : Speed up the Work

Anti: More Developers Increases Productivity

- **Pattern**: The C level sees the team is not delivering to the deadlines. As a result they add more members to the team as the project progresses, to speed work up.
- **Solution**: It is well demonstrated that adding more people to a project may not speed up work. It makes communication harder, more complex. Also the new team members are not on the same page as the old members. The whole team has to slow down to share and re-negotiate beliefs, assumptions and values. The new members are not across the “way of working”. However having a description of this speeds up the sharing. If a good team size is already there, then the product team should re-evaluate their objectives. How do we realistically grow the product.
- **Principle**: Everybody all together from early on

Anti-Patterns : Speed up the Work

Anti: The Team is Pushed to Deliver

- **Pattern:** The team has been told there is a deadline soon and they have to get the work done. They are asked to increase velocity. The team feels rushed with their work, and are not producing quality, bugs are surfacing. Some members are working longer hours, and stress is evident. The team is no longer mastering their work.
- **Solution:** The Iteration plan is reduced in scope, rather than quality. The product goal of an MVP is clarified. A working release is still delivered but some less important features are removed. Requirements are moved to the product backlog, for planning in the next iteration. The team uses the checklist to take stock and work out next steps. The team must work at a sustainable pace otherwise they will burn out, get frustrated or make mistakes.
- **Principle:** Work at a sustainable pace

Anti: The iteration is filling with blockers

- **Pattern:** The standup is becoming blocked by too many issues, and tasks are not getting completed. Issues are complex. The card states are not progressing.
- **Solution:** Call *Andon* and stop the line; the iteration. Address the issue right now, rather than continuing production. Perform a retrospective and start a new iteration, with a new plan. Tasks are produced to fix the issues, which may focus on improving the 'way of working'

Anti-Patterns : Iterations

Anti: Micro Iterations

- **Pattern**: The iterations are very short , e.g. less than 1-2 weeks.
- **Solution**: Such short iterations suggest either that micro-management is in play or that planning is not working. Maybe the product is complete and random bug fixes are rolling in, and have we already reached 'Done'. Iterations should be no less than 2 weeks and a max of a 4. A month is just enough time to allow the development team to deliver proper value. Given clear goals and a product plan, an iteration plan should be able to cover 4 weeks of meaningful work. If that cannot be organised maybe the team is way too reactive, and the direction is not really understood or the goals aren't welll defined.

Anti: Every Iteration is a Working Release

- **Pattern:** The team advocates that every iteration produces something that can be demonstrated to stakeholders.
- **Solution**: This is taking this practice to an unnecessary extreme. For products with complex architecture or framework, a base foundation may be required before any interesting features can be delivers. Forcing user features in early will mean needles future re-work; waste. The stakeholder may have no interest in seeing, a server, producing log output. It can be agreed that usable requirements will be produced after a few iterations. Agree on definition of 'Done' at the start of project for these iterations.

Anti-Patterns

Anti: Part of Team is Moving Ahead

- **Pattern:** Some members of team start working on future items, that have not been scheduled. They start asking other team members question and raise issues. The members receiving these, are busy on other tasks and cannot answer or think about the issue right now. More questions arrive, and solutions made independent of the team.
- **Solution:** The Iteration plan is fixed in scope, for a good reason. New work should not be entering, it may not be required. Ask the team members to wait until this work is prioritized and scheduled and offer help on current work instead.

Anti: Standup is for Performance Reporting

- **Pattern:** The Standup becomes a mechanism for a manager to get a report on how each team member is performing. The manager decides the next moves for each member. The team is just following**.** The time required to get a manager up to speed with technical details means the meeting will drag on, and the team will loose focus.
- **Solution:** The standup is for everybody in the team to collaborate and share not report. The project is managed by a combination of iteration planning, and blocker resolution by the team. Everybody in the team helps manage, as servant leaders. The product owner is part of the team so is involved daily.
- **Principle**: Everybody all together from early on

Anti-Patterns

Anti: The 'way of working' is set by others

- **Pattern**: The SDLC process is fixed and is to be adopted by all teams. Management has little idea of the team's background, what they desire and how they interact and want to work. They are not aware of the actual experience of each team member doing the work.
- **Solution**: The team leads the implementation of its "way or working". They improve this frequently via the retrospective. The team owns the responsibility to meet the product plan, and continually monitor their progress, so that real issues can be raised. The product owner is part of the team and is actively involved day to day. No surprises can arise.

Anti: Requirements not "satisfactory", ready

- **Pattern:** Requirements are not "Conceived", "Bounded" or "Coherent". In many cases there are no requirements, and the product team does not agree on what's required. Despite this the team ploughs on and builds, to find the system is not what's wanted.
- **Solution**: This situation requires immediate feedback and escalation upward. The risk of not knowing what is to be built is significant. It can result in wasted build exploring what may be required, via Spikes. "Some thinking before doing" is required. Unfortunately without good requirements the product can't be tested or 'Done' defined. Significant extra work may be required to get requirements to "satisfactory" and will push out delivery. Team leadership requires raising significant risks. The development team can reject these requirements.

Anti-Patterns

Anti: Team is Moonlighting

- **Pattern**: The team members are busy on multiple projects and never seem to be available for the current project. The management team wants to maximise the use of employees. The members are juggling multiple tasks, and always short of time.
- **Solution**: Task-switching (Multitasking) is a form of waste; humans are not capable of doing multiple tasks at one time as the human brain is linear. Each task change results in a delay. “Everybody all together” means the team is focused on one project, is a member of one team. If team members are involved in other projects, they are not be available right now to help and focus with the team on an issue. This means work is delayed, and if other members are also delayed, a simple issue can extend out in time for days or weeks. Although it appears to reduce costs to maximise the use of employees, it reduces efficiency, and significantly impacts all projects. As the team is self organizing if members have free time, they should help others out on the one project, with their one team.

Anti: Upfront detailed requirements

- **Pattern:** The use cases and stories are all detailed up front, as in waterfall.
- **Solution**: The use case start off as high-level Epics and, JIT, through the iterations are developed in more detail. The stories are developed in a similar way iteratively. Some requirement may not be needed, or change, so only detail when prioritised.

Anti-Patterns

Anti: Velocity has become an obsession

- **Pattern**: companies spend a lot of time discussing velocity and devising ways to increase it, they see drops in velocity as a problem. Companies brainwashed by velocity worship do not respect software craftsmanship, nor the delivery of value.
- **Solution**: Velocity isn't part of Scrum. The Sprint Backlog is nobody's business except the development teams. The only sprint commitment is the sprint goal, and if this is met then velocity is no one else's business. Companies should show more trust and respect to their developers. Teams often work on new aspects of a product, new solutions, new technology; past velocity often relevant to new work, a poor predictor.

Anti: No slack time for Tech debt

- **Pattern**: the development team is not getting enough time to handle tech debt or is not getting slack time and are always fully utilised on new work. The team members are pushed on their own work and can't help other members.
- **Solution**: the development team needs to demand time to address tech debt and the product owner needs to accept this. If not, future delivery will be impacted and create bottlenecks. The team also needs to factor in slack time (20%) , to support other teammates, and tackle new issues. Free time must be made available to collaborate. Over utilisation will reduce collaboration, and a focus on my work only mentality.

Anti-Patterns

Anti: Dominant Planners

- **Pattern**: The development team is not collectively participating in planning, and some members represent the whole team.
- **Solution**: There are no leaders in a scrum team, and all members need to participate and have a voice.

Anti: Poor Commitment

- **Pattern:** The sprint commitment is not team based. The commitment is influenced by outside factors, such as management, tech leads, or the product owner dominates. Prior velocity is used to determine what can be done to ensure team is used to full capacity
- **Solution**: The team needs to determine the root cause of the external influence, and then agree on how to get control back. Only the team should decide on what they commit to

Anti: Multiple Sources of Requirements

- **Pattern**: Specs, document and notes on the product are all over the place.
- **Solution**: The backlog is the single source of requirements. The product owner manages this. There is no doubt as to what's required, where it is, and how its defined.

Anti-Patterns

Anti: Efficiency via overuse

- **Pattern**: In an attempt to be as efficient as possible and avoid underusing their programmers, some managers make sure that developers have more to do than possible.
- **Solution**: This approach produces many problem, burn out, no slack time in project, reducing skill levels and no time to communicate. Most often quality is reduced to meet deadlines. Provide breathing room to respond to unexpected circumstances, improve knowledge and skills, communicate and develop as a team.

Anti: Reduce and Manage Variability

- **Pattern**: Even though most programming situations are unique, many software development managers are hostile to variability and try to reduce and actively manage this.
- **Solution**: Manage flows in the presence of variability. Variability is a reality.

Anti: Stick to the plan

- **Pattern**: managers try as hard as possible to conform to the original plans and spend huge amounts of effort to get back to those plans, ignoring reality.
- **Solution:** leveraging the new information that has become available and devise new plans. Plan iteratively knowing that change is ever present.

Flow
Don Reinertsen

The Principles of Product Development Flow

1. Take actions on software development projects by quantifying and comparing the cost of taking them versus the cost of not taking them.
2. Since it is impossible to change just one thing at a time, perform sensitivity analyses to quantify how changing one aspect of a software development project affects the other aspects of the project.
3. If you can only quantify one aspect on a software development project, quantify the cost of delay associated with each action on the project.
4. When measuring any aspect of a software development project, be sure to quantify in terms of value to the *customer* instead of value to the *organization*.
5. Measure progress on the software development project instead of the activity (or inactivity) of the project members.
6. Most aspects of software development projects have "U-curve" characteristics, so it is not necessary to reach the optimal (i.e., best) for any aspect as substantial improvement is typically sufficient.
7. When managing any project, decisively reaching even imperfect answers improves decision making.
8. Contrary to the Pareto principle that suggests that 80% of a project's leverage lies in 20% of the project's decisions, correctly managing the numerous small decisions (the 80%) can make a significant difference.
9. Evaluation of a project's economics is an ongoing process that should occur on a continuous basis.
10. Opportunities vanish swiftly so economic choices are more valuable when made quickly.
11. When faced with a decision that appears to only have bad alternatives, break the alternatives down into greater detail so hidden benefits of the alternatives come to light. Use those benefits to pick an alternative.
12. Allow software developers to make their own trade-offs up to a predetermined threshold. After that, escalate decision-making to the next level of management which has a higher threshold. The threshold process can be applied to all levels within an organization.

The Principles of Product Development Flow

13. Allow software developers to make their own trade-offs up to a predetermined threshold. After that, escalate decision-making to the next level of management which has a higher threshold. The threshold process can be applied to all levels within an organization.
14. Establish a set of decision-making rules for a project so that decision making can be aligned yet spread out among those who are directly responsible for the software development.
15. Instead of project managers doling out resources based on *first to ask* or a *requestor's persuasive capabilities*, use a supply/demand model so that project members who reap the benefits also bear the costs.
16. Rather than front-load all decisions on a project or delay every decision as long as possible, determine the economic impact of each decision and use that information to determine its optimal timing.
17. When making content versus schedule trade-offs, be sure to determine the marginal (i.e., added) cost, not the total cost, of adding a new feature to a project versus the marginal benefit.
18. Consider money *to be spent* in decision-making instead of money *already spent*.
19. The cost of *determining whether or not to take an action* on a project should not exceed the cost of *actually performing the action*.
20. The cost associated with *creating project alternatives* should not exceed the cost of *not taking the project action*.
21. A high probability of failure does not mean bad economics.
22. When reviewing project actions with upper levels of management, it is possible to speak in financial terms since this methodology proposes that everything is quantified in customer value.

Collection of Project Issue

Delivery Issues

- Projects late, over budget, not what's wanted;
- Delivered outcome not strategically aligned and not based on market need;
- Process re-developed from scratch, for each project, re-inventing the wheel;
- No one knows how to start the project, everyone is running in different directions;
- Poor, missing, wrong requirements or requirements as long incomprehensible prose;
- Project is to deliver a list of features not outcomes;
- Projects start build without requirements or architecture;
- Risks pushed out in time, and not understood;
- Requirements are fixed early on and change creates project risks, and schedule blowouts;
- Teams poorly formed, resources never available, schedule conflicts;
- Inability to estimate projects, estimates are way out, not objective or understood;
- Too greater reliance on unreliable estimates, assumed to be predictions;
- Unclear where a project is at what's been done or why;
- Low or unknown quality;
- Stakeholders unclear what IT is doing;
- Let's add more features – endless scope creep;

More Issues……

- Lack of documentation (no corporate memory) / too much documentation;
- The code is the only documentation/architecture
- Project gates are based on document delivery, not solution/product delivery;
- Changing conditions: requirements, technology and politics;
- Stress, long hours; success by 11th hour heroes;
- Top down planning (management) introduced to fix lack of control/visibility;
- Command and Control, direction from above, however those above are in the dark, cycle of more control to fix more issues, escalates out of control;
- Micromanagement; Gant charts, tighter project management verses Iterative/Agile;
- Project plans that are not aligned to development reality;
- Add more PMs as problems arise to control chaos;
- Growing rework to fix bugs or wrong deliverables;
- Growing expectations and shrinking timeframes;
- Process: too heavy / too light / none;
- ‘Process’ viewed as a dirty word, it has negative connotations;
- Agile viewed as not meeting the need in larger organisations;

Even More Issues……..

- Roles not understood, and team members not sure what to deliver or why. Wrong people / skills allocated to tasks, the blame game starts, arses covered;
- Unclear what architecture is and how it interacts with other roles/deliverables;
- Artefacts delivered after required, way before required, or not delivered at all, and produced for unknown reasons, or without team agreement;
- People waiting idle while other tasks are completed;
- Too many managers, too much hierarchy, with completely unrealistic goals;
- Too many long pointless drawn-out rambling meetings;
- The creative capability of a software team ignored;
- The business considers ITs function is only to deliver what they are told;
- Testing is too late, its not clear what's tested, coverage unknown;
- Tests do not trace back to requirements, wrong things tested;
- More testers added to fix quality issues, QA work grows, and becomes bottleneck;
- Concepts and terminology not shared, with different interpretations across team;
- Process is assumed to be 'followed' but there really is no 'process description; so no one 'really' knows; Confusion is assumed to be the norm.
- Need to get as much work out of team as possible, as soon as possible;

Finally … More Issue

- The business cannot articulate what they want, keep changing their minds, or don't agree;
- Lack of objective measurement or any measurement at all; the state of the project is not known, and no one knows how to work this out. Are we there yet ?
- We say we are 'Agile' but documents and detailed specs are required, we have many tools, and as we are a large enterprise, we have complex legal contracts;
- Geographically separate teams work in isolation;
- No agreed architecture;
- Decisions made but later not accepted by others; there was no agreed review process.
- Decisions forgotten – why did we do that ?
- Tasks forgotten – "we were meant to put this back in later"
- Requirements forgotten;
- Unclear how to accept requirements. Have they really been delivered;
- System does not meet non-functional aspects, and no one knows what they are;
- Trade-offs cannot be understood;
- Team is worn out, overworked, deadline driven, the death march syndrome;
- Same next time round; lessons not learnt; results of retrospectives disappear.

Real Agile Issue

Real Comments from Medium Article - verbatim

Maarten Dalmijn wrote an article in Medium where he shared issues over his last 7 years with strange experiences with Scrum and Agile:

- “A deadline for a new product that was decided by the C-level, even though nobody knew what the product should do, the market it would serve, nor the underlying technology we would use to build it. Regardless this uncertainty, it was communicated with absolute certainty when our new product would be live, together with the scope. Needless to say, we never met that deadline and nobody felt particularly compelled to meet it.”
- “We hired an extra development team to speed up the delivery of a new project. All the communication and coordination of the new team cost our existing development team so much time, that basically we were paying three times as much to go at the same pace as before.”
- “Teams were expected to increase their velocity, as otherwise why would we pay extra for hiring a Scrum Master? We needed to make sure the Scrum Masters were worth their money.”
- “Development Teams that spent 50% of their times on Spikes, because their Product Owner didn’t have enough time to explain everything to them.”

Real Comments from Medium Article - verbatim

- “During the Daily Scrum a lot of people participated who were not part of the Development Team. The event became a big mess, instead these powerful stakeholders could have just been listening in, or not attend at all.”
- “Daily Scrum was a status report and nobody talked about the Sprint Goal. The three questions, minus the Sprint Goal, were key. Kinda missing the whole point of the event.”
- “Developers did not like testing, and I was a QA, so at the end of the Sprint I would always work like crazy to make sure we would still be able to complete the Sprint.”
- “We had a Definition of Ready that was more complicated than our Definition of Done. Our Product Owner would already start sweating before refinement started. It was considered a sport to poke holes and figuring out missed details in presented issues.”
- “The Product Owner held the role of Scrum Master as well. Like the Product Owner role wasn’t complicated and overwhelming enough. The Scrum Master role was seen as overhead and we want our developers to focus on coding.”
- “Only QA’s were allowed to test, even though it was the biggest bottle-neck in our development process.”
- “Everything we did was a User Story, even if it made no sense to use a User Story.”