
# The AK Way Play book

Armakuni


armakuni.com

# Your Guide to What's Inside

# What is The AK Way?

The AK Way isn’t just another development framework; it’s a solid foundation built on & refined over 12 years of hands-on experience.

These refined baseline practices reflect our opinionated approach to delivering software. While not rigid or exhaustive, they ensure every decision—both ours and the client's—is intentional and well-informed. We often call them our "cheatsheet," and we’ve successfully adapted them to fit many clients' needs.

It has been proven by enterprise clients like HSBC, NewsUK, GDS, JP Morgan Chase, Red Bull, and Hilton, as well as fast-moving startups in fintech, consumer, and SMBs. These practices have helped companies innovate, deliver faster, improve quality, and reduce costs.

For Armakuni, The AK Way means all our staff start with a clear idea of what good looks like. This ensures consistency across all projects. While it doesn’t cover every scenario, The AK Way addresses the core aspects of most engineering functions, providing a strong foundation for

These practices are where we begin any engineering or software work. If a client doesn’t have something in place, we rely on our baselines to guide us. They form the foundation of our approach, keeping our teams focused on what truly matters.

The Ak Way is grounded in core values, providing a foundation for good software delivery.  We focus on core principles, not just specific tools, which can change over time. Tools support practices, but it’s the underlying values that ensure lasting effectiveness.

We believe combining our Refined Baseline Practices increases the chances of success multifold. They are a key part of the “special sauce” Armakuni brings to clients. We constantly review and refine The K Way to keep it most effective.

# Cloud Native Outcomes using The AK Way

| Speed | Quality and Reliability | Cost | Innvoation and Modernization/ Cloud Native |
|---|---|---|---|
| 70% | 80% | 60% | 85% |
| Reduced feature release time<br><br>Environment setup times drop from days or weeks to just 30 minutes.<br><br>Code changes hitting production in under 15 minutes | Increased test code coverage<br><br>Repeatable, secure, and resilient environment that easily adapt change<br><br>Continuous availability and rapid recovery from failures | Cost savings by removing manual tasks & streamlined processes<br><br>Productivity boost by integrating Generative AI into workflows.<br><br>Metrics driven development | Efficiency with everything as code, automation, TDD, fully pipelined and automated environment provision<br><br>Quickly respond to changes in demand, handle peak loads, and drive cost efficiencies.<br><br>Agile delivery, end-user-focused product management, and DevOps |
| VS | VS | VS | VS |
| **Months** | **Poor quality software** | **10x** | **Monolithic apps** |
| Spent in Manual setup, slow deployments, and long release cycles | Messy codebase, tech debt, and inefficient team | Spent on fixing bugs instead of innovating | Unable to modernise or adopt cloud native |

# AK Way

## Purpose

Return creativity and joy to the world of software engineering & delivery

## Vision

To be the world's most trusted partner for empowering businesses to accelerate the delivery of their greatest ideas

## Mission

Businesses, at any scale, are awash with great ideas. However, the cost of learning which idea customers will love is too high. We evaporate this constraint and build quality in by accelerating cloud native adoption using next-generation consultancy, engineering and immersive learning

## Values

- Honesty
- Pride
- Ownership
- Creativity
- Diversity

## Principles

- People First
- Empathy
- Radical Candor
- Action Biased
- Experimentation
- Fast Feedback

# The AK Way Checklist

**How to use this sheet**

This is not a checklist. These are your fallback defaults, if nothing else is set, not a golden master you must do.

## Vision & Empathy

**We understand the customer**
**The customer is as the centre**

- ❑ Understand the customer
- ❑ Understand our stakeholders
- ❑ Shared view of how value flows to our customers
- ❑ Regular, early and direct feedback from the customer
- ❑ Understand the current domain, language & terminology

**We have a clear vision**
**Highly aligned and focused**

- ❑ Understand the broader business's vision
- ❑ We can all present a clear vision, mission and purpose
- ❑ We can measure what good looks like
- ❑ We have just enough of a shareable roadmap

## Team-First

**Team can be successful**
**We can have sustainable success**

- ❑ Delivery expectation doesn't exceed the team's cognitive capacity
- ❑ Roles, and role rotation are discussed and understood
- ❑ Working agreements negotiated with the whole team
- ❑ We are sensing / listening for evolutions of our team types and interactions
- ❑ We have a Team API or clearly defined service offering

## Cloud Native

**We start with production first**
**Production-ready from the outset**

- ❑ Production deployed first
- ❑ Deployed Hello World on production
- ❑ Deployed other environments

**We continuously deploy**
**Pipelines!**

- ❑ All deployments are pipelined
- ❑ Build fixed within 10 min of breaking
- ❑ All tests run and are reliable
- ❑ Coding standards are checked

**We automate everything**
**Everything as Code**

- ❑ Environments recreated from only code
- ❑ Environments regularly destroyed and created

**We do day 2 operations on day 1**
**Day2Ops right from the start**

- ❑ Logs are fully accessible to engineers
- ❑ Events are fully traceable to requests
- ❑ Service status is monitored
- ❑ Service Level Objectives set
- ❑ We can recover from disasters
- ❑ We can upgrade continuously

## Code As Craft

**Coding practice**
**Code well, code fast**

- ❑ Commit to master daily
- ❑ Leave code better than you found it
- ❑ Pair program daily

**We test everything**
**Automated tests**

- ❑ All new functionality has automated tests
- ❑ All bug fixes have automated tests
- ❑ Non-functional requirements are verified with automated tests
- ❑ Test Driven Design is the default practice
- ❑ Tests give the confidence to deploy
- ❑ Failing tests fail the pipeline

**Prototype Working Agreements:**
**Definition of Ready (DoR)**

- ❑ The story discussed in an IPM
- All acceptance criteria
- ❑ defined
- The story is prioritised
- ❑ Team has estimated the story

**Definition of Done (DoD)**

- ❑ All tasks on the story are completed (and marked as
- ❑ such)
- All acceptance criteria
- ❑ completed
- All commits are tied to the
- ❑ story
- Story validation instructions included

## XP & Agility

**We set goals every quarter**
**Inception meetings**

- ❑ Goals and Non-goals defined
- ❑ Actors identified
- ❑ Risks & mitigations identified

**We work in weekly iterations**
**Iterations**

- ❑ Started with IPM, restated goals
- ❑ Daily standups, terse and quick
- ❑ Each story completed in priority order
- ❑ Work in progress has limit of 1
- ❑ Pair program daily
- ❑ Ended with a demo & retrospective

**We plan work as user stories**
**User stories**

- ❑ Customer value is defined
- ❑ Stories sized to 0.5 days work
- ❑ Acceptance criteria are defined
- ❑ Prioritised with the Product Owner
- ❑ Accepted with the Product Owner
- ❑ Conform to Working Agreements
- ❑ Document just enough to proceed

## Engagement

**We enhance team enthusiasm**
**Growing engineer confidence**

- ❑ Retro outcomes are actions
- ❑ Actions are carried out
- Rotate roles
- ❑ Everyone demos and talks
- See one, do one, teach one

**We radiate our success**
**Visual working**

- ❑ Display WIP Board/backlog
- ❑ Display deployment pipeline
- Display product metrics
- ❑ Show & tell product and WoW]

**We demonstrate client success**
**Show it's working**

- ❑ The client can make simple
- ❑ investment decisions
- Regular, investment to value-
- ❑ focused reporting

**We deliver client outputs**
**Safe & Secure delivery**

- ❑ Ways of working are always secure and legal
- ❑ Golden source client artefacts live on client systems

AK Way Cards

Understand the Customer

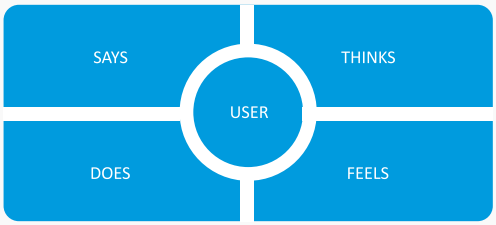
# Vision & Empathy

We understand the customer

## Empathy Mapping

A collaborative tool used to gain deeper insight into users' emotions, thoughts, and behaviors to enhance user-centered design.

Teams understand and address user needs more effectively, leading to more intuitive and user-friendly designs.




1. Gather your team and necessary materials (e.g., a large paper or whiteboard, sticky notes).
2. Create four quadrants labeled: "Says," "Thinks," "Does," and "Feels."
3. Conduct user research or interviews to gather insights.
4. Populate each quadrant with observations and quotes from users, reflecting their thoughts, feelings, actions, and spoken words.
5. Discuss findings to identify patterns and insights for improving user experience.

Understand our Stakeholders

1 hour

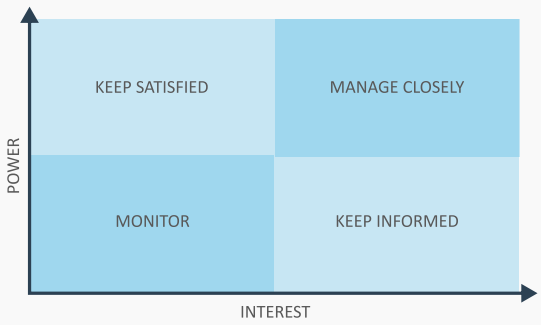
# Vision & Empathy

We understand the Stakeholders

## Stakeholder Mapping

Identify and analyze individuals or groups who influence or are affected by a project, ensuring effective communication and engagement.

Prioritize engagement efforts, ensuring all relevant parties are appropriately informed and involved in the project's success.




1. List all individuals and groups affected by or influencing the project.
2. Assess each stakeholder's level of influence and interest in the project.
3. Place stakeholders on a power-interest grid with four quadrants: high power/high interest, high power/low interest, low power/high interest, low power/low interest.
4. Develop tailored strategies for engaging each category (e.g., keep high power/high interest stakeholders closely informed and involved, monitor low power/low interest stakeholders).

Shared view of how value flows to our customers

# Vision & Empathy

We understand the customer
The customer is at the centre

## Value Stream Mapping

A lean management tool used to visualize and analyze the flow of materials and information required to bring a product or service to the customer.

Identifies inefficiencies and opportunities for improvement, leading to more streamlined and cost-effective processes.

1 Day

**LEAD TIME**

The total amount of time from asking for, to delivery that a process takes, so including any time waiting in the queue.

3.5 H

**PROCESS TIME**

The time it actually takes to complete the process. This must be equal to or less than the lead time.

50%

**COMPLETE AND ACCURATE**

The percentage of time that a process has been completed without problems.

1. **Select the Process:** Identify and define the specific process or product line to be mapped, ensuring it has a clear start and end point.
2. **Gather Data:** Collect detailed information on lead time, process time, and complete and accurate for each step of the process.
3. **Create Current State Map:** Use diagrams to visually represent the current process flow, including all steps, delays, and information flows.
4. **Analyze the Current State:** Examine the map to identify areas of waste, such as excess inventory, waiting times, and unnecessary movements or processes.
5. **Design Future State Map:** Develop a new map that eliminates identified wastes and inefficiencies, streamlines steps, and optimizes the flow of materials and information.
6. **Implement Changes:** Put the new process design into practice, train staff on the changes, and establish metrics to monitor the effectiveness of the improvements.

Regular, early and direct feedback from the customer

# Vision & Empathy

We understand the customer

## Lean Startup Principles

A focus on developing businesses and products through iterative experimentation, validated learning, and customer feedback to minimize waste and increase efficiency.

They reduce the risk of failure by ensuring products are built to meet real customer needs through continuous feedback and rapid iterations.

1. **Validated Learning**:
   Build products based on real customer feedback and data.

2. **Build-Measure-Learn:**
   Develop in short cycles, continuously measuring results and learning from them.

3. **Minimum Viable Product (MVP):**
   Create a basic version of the product to test assumptions and gather feedback quickly.

4. **Pivot or Persevere:**
   Use feedback to decide whether to make fundamental changes (pivot) or continue on the current path (persevere).

5. **Innovation Accounting:**
   Track progress using metrics that matter for startups, focusing on actionable, auditable, and accessible data.

Understand the current domain, language & terminology

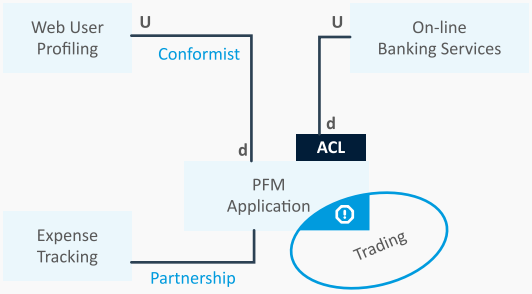
# Vision & Empathy

We understand the Stakeholders

## Domain Mapping

Domain mapping helps visualize and organize the relationships and boundaries within a particular field or area of interest.

Clarifies complex relationships and enhances understanding of the domain, aiding in better decision-making and strategy development.




1. **Define the Scope:** Identify the specific domain or area of interest.
2. **Identify Key Elements:** List all relevant entities, concepts, or components within the domain.
3. **Map Relationships:** Draw connections between these elements to show their relationships and interactions.
4. **Review and Refine:** Analyze the map for completeness and accuracy, and adjust as needed.
5. **Share and Discuss:** Present the map to stakeholders for feedback and further insights.

Understand the broader business's vision

1 hour

# Vision & Empathy

We have a clear vision
Highly aligned and focused

## North Star

The team identifies and aligns on a single, critical metric or goal that represents the ultimate value they aim to deliver to their customers, guiding their efforts and decision-making processes.

Ensures team alignment and focus on delivering the highest value to customers.

1. **Gather Team:**
   Assemble key team members for a focused session.
2. **Define Value:**
   Discuss and identify the core value your product or service delivers to customers.
3. **Identify Metrics:**
   Brainstorm potential metrics that best represent this value.
4. **Select North Star:**
   Choose the single most impactful metric that aligns with your long-term goals.
5. **Align and Communicate:**
   Ensure everyone understands and is committed to the chosen North Star metric.
6. **Regular Review:**
   Continuously monitor progress and adjust as necessary to stay aligned with the North Star.

We can all present a clear vision, mission and purpose

# Vision & Empathy

We have a clear vision
Highly aligned and focused

## Vision Statement

Brainstorm, discuss, and formulate a clear and inspirational statement that articulates the long-term goals and aspirations

Ensures alignment and provides a clear, motivating direction for the organization's future.

**Purpose**
Why do we exist?

**Future Aspiration**
What do we want to achieve in the future?

**Impact**
What change or difference do we want to make?

1. **Understand Purpose:** Reflect on why your organization exists.
2. **Envision the Future:** Imagine your ideal future state.
3. **Identify Key Elements:** Highlight core values and long-term goals.
4. **Draft and Refine:** Write and refine a concise, inspirational statement.
5. **Finalize:** Agree on and finalize the vision statement.
6. **Embed in Culture:** Integrate the vision into all organizational activities and communication

We can measure what good looks like

# Vision & Empathy

We have a clear vision
Highly aligned and focused

## Inputs and Outcomes

Identifies key resources and activities (inputs) and their desired results (outcomes) to develop relevant metrics for success.

Ensures that created metrics are aligned with both the resources used and the desired results, providing a comprehensive measure of success.


1. **Define Objectives**
   Clearly outline the goals of your project or process.
2. **List Inputs:**
   Identify the key resources and activities needed to achieve the objectives.
3. **Identify Outcomes:**
   Determine the desired results and impacts of these inputs.
4. **Develop Metrics:**
   Create specific, measurable indicators that link inputs to outcomes.
5. **Review and Refine:**
   Ensure the metrics are relevant, actionable, and aligned with objectives.
6. **Implement and Monitor:**
   Apply the metrics and regularly track progress towards the outcomes.

We can all present a clear vision, mission and purpose

# Vision & Empathy

We have a clear vision
Highly aligned and focused

## Vision Statement

Brainstorm, discuss, and formulate a clear and inspirational statement that articulates the long-term goals and aspirations

Ensures alignment and provides a clear, motivating direction for the organization's future.

**Purpose**
Why do we exist?

**Future Aspiration**
What do we want to achieve in the future?

**Impact**
What change or difference do we want to make?

1. **Understand Purpose:** Reflect on why your organization exists.
2. **Envision the Future:** Imagine your ideal future state.
3. **Identify Key Elements:** Highlight core values and long-term goals.
4. **Draft and Refine:** Write and refine a concise, inspirational statement.
5. **Finalize:** Agree on and finalize the vision statement.
6. **Embed in Culture:** Integrate the vision into all organizational activities and communication

We can measure what good looks like

# Vision & Empathy

We have a clear vision
Highly aligned and focused

## Inputs and Outcomes

Identifies key resources and activities (inputs) and their desired results (outcomes) to develop relevant metrics for success.

Ensures that created metrics are aligned with both the resources used and the desired results, providing a comprehensive measure of success.


1. **Define Objectives**
   Clearly outline the goals of your project or process.
2. **List Inputs:**
   Identify the key resources and activities needed to achieve the objectives.
3. **Identify Outcomes:**
   Determine the desired results and impacts of these inputs.
4. **Develop Metrics:**
   Create specific, measurable indicators that link inputs to outcomes.
5. **Review and Refine:**
   Ensure the metrics are relevant, actionable, and aligned with objectives.
6. **Implement and Monitor:**
   Apply the metrics and regularly track progress towards the outcomes.

We have just enough of
a shareable roadmap

1 hour

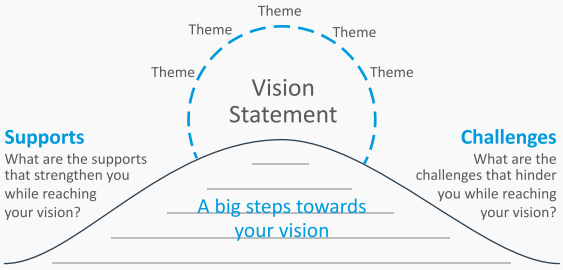
# Vision & Empathy

We have a clear vision
Highly aligned and focused

## 5 Bold Steps

Teams define and prioritize five key actions to achieve their desired future state.

Helps teams define and prioritize key actions that drive significant progress towards their desired future, ensuring focused and strategic efforts.




1. **Visualize the Future:**
   magine the future state where the goal has been achieved and describe it in detail.

2. **Identify Bold Steps:**
   Brainstorm and identify five significant, bold steps that will drive the organization towards the desired future state.

3. **Prioritize Steps:**
   Discuss and prioritize these steps based on their impact and feasibility.

4. **Create Action Plans:**
   For each bold step, develop detailed action plans outlining the necessary tasks, resources, timelines, and responsibilities.

Delivery expectation doesn't exceed the team's cognitive capacity

# Team-First

Teams can be successful
We can have sustainable success

## Cognitive Load Workshop

To recognize and reduce the amount of mental effort required to process and understand information.

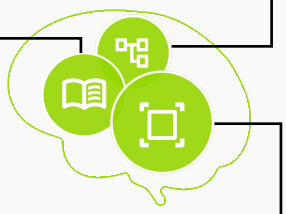
Ensures that delivery expectations align with the team's mental capacity, preventing overload and enhancing productivity.

**Intrinsic, The Base Effort**

Understanding the inherent complexity of the material or task itself.

**Extraneous, The Environment**

The way information is presented and by activities that do not directly contribute to learning.



**Germaine, Area of Special Focus**

Processing, constructing, and automating schemas to facilitate learning and understanding.

1. Write down the tasks you do day-to-day. Remember to include your learning to do things.
2. Sort them into Intrinsic, Extraneous, and Germaine.
3. **Write down ideas to:**
   - Simplify and automate intrinsic tasks.
   - Minimize extraneous load. Reduce the noise and distractions from your environment.
   - Maximize the germane load. Explore, innovate, and create.

Roles and role rotation are discussed and understood

1 hour

# Team-First

Teams can be successful
We can have sustainable success

## Team Overview

A collaboration exercise to introduce your team to one another, their input, skills, goals, and time commitments.

Teams are aligned and focused, sharing their expectations, and reducing potential misunderstandings.

To fill out the Team Overview template, follow these steps for each team member:

1. **My name is...:** Write the name of the team member in the corresponding row under this section.
2. **Days / week on this project:** Specify the number of days per week the team member will be dedicated to the project.
3. **Skills:** List the key skills the team member brings to the project. Include technical skills, soft skills, and any relevant expertise.
4. **I'd like to learn...:** Note any specific skills or areas the team member is interested in learning or improving upon during the project.
5. **Notes:** Add any additional information or special notes about the team member that might be relevant to the project. For example, availability, planned absences, or unique circumstances.

| My name is... | The doctor | | | |
|---|---|---|---|---|
| Days/week on this project | 4.5 | | | |
| Skills | Time lord, PHP... | | | |
| I'd like to learn... | Screwdriver maintenance | | | |
| Notes | I'm regenerating on 25th, so out all day | | | |

Once all sections are filled out, review the completed template together to ensure everyone understands each other's roles, availability, and learning interests.

Working agreements negotiated with the whole team

1 hour

# Team-First

Teams can be successful
We can have sustainable success

## Team Charter

A document that outlines a team's purpose, goals, roles, and operating norms to ensure alignment and effective collaboration.

Clarifies expectations and enhances team cohesion and performance.

1. Go through each section one by one, encouraging open discussion and input from all team members.
2. Write down the agreed-upon points in each corresponding section of the canvas.

- **Team Members**: List who is on the team and what each member brings in terms of roles, values, skills, and character traits.
- **Expectations:** Discuss and write down what team members expect from each other to become a successful team.
- **Team Values**: Identify and note the core values that all team members recognize and agree to live by.
- **Driver**: Determine who will lead the team (the driver), who will guide (the navigator), how the team will choose its direction, and how decisions will be made.
- **Team Goals**: Define the main goal the team wants to achieve, and outline what success looks like and when it should be reached.
- **Energy Source**: Identify what generates energy and motivation in the group, and what keeps everyone going.
- **Obstacles**: Discuss potential obstacles that could prevent the team from reaching their goal and how to overcome them.
- **Trouble**: Plan how the team will handle crises or unexpected problems when they arise.

3. Review the completed canvas to ensure everyone is aligned and understands the content.
4. Keep the canvas visible and refer back to it regularly to stay on track and address any changes or challenges.

We are sensing / listening for evolutions of our team types and interactions

1 hour

# Team-First

Teams can be successful
We can have sustainable success

## Team Topologies - Team Types

Structured roles designed to optimize software delivery and operations through stream-aligned, enabling, complicated-subsystem, and platform teams.

These team types enhance collaboration, reduce dependencies, and improve productivity in software delivery and operations.

1. Take a look at the different kinds of teams in Team Topologies.
2. You may want to think about what type of team you are. Are you a Platform team, an Enabling team, or a Stream-aligned team?
3. How do you want your team to interact with others which provide you with data necessary to complete your tasks? Should they be operating in X-as-a-Service or Facilitation mode? Can you discuss the positioning of your team with respect to others with your stakeholders?
4. Are there existing patterns which articulate how your team could interact with others?

**Stream-aligned Team**

Aligned to a continuous flow of work from a specific business domain or customer need, focusing on delivering valuable outcomes directly to customers.

**Complicated-Subsystem Team**

Responsible for building and maintaining parts of the system that require specialized knowledge and expertise, often due to their complexity or criticality.

**Enabling Team**

Helps stream-aligned teams to overcome obstacles, adopt new skills or technologies, and increase their autonomy.

**Platform Team**

Provides internal services to streamline and support the work of stream-aligned teams, aiming to reduce their cognitive load by offering self-service capabilities and reusable components.

We have a Team API or clearly defined service offering

1 hour

# Team-First

Teams can be successful
We can have sustainable success

## Team API Template

A document outlines the key aspects of how a team operates, communicates, and interacts with other teams, including responsibilities, processes, tools, and expectations.

Ensures clarity and consistency in team interactions, enhancing collaboration and efficiency across the organization.

Let's get started!

1. **Define Mission** — State the team's purpose and objectives.
2. **List Responsibilities** — Specify key responsibilities and deliverables.
3. **Set Communication Protocols** — Outline preferred channels and meeting schedules.
4. **Detail Processes** — Describe standard workflows and procedures.
5. **Identify Tools** — List tools and technologies used.
6. **Set Expectations** — Define response times and service levels.
7. **Link Resources** — Provide links to relevant documentation and support materials.
8. **Review and Share** — Get feedback from team members and stakeholders, then finalize and distribute the template.

Production deployed on day one

1 Day

# Cloud Native

We start with production first
Production-ready from the outset

## Walking Skeleton

A small implementation of the system that performs basic end-to-end function, with a focus on continuous iterative evolution.

Links together the main architectural components. The architecture and the functionality can then evolve in parallel to build a robost product.

A walking skeleton is a stripped-down, minimal version of a system that showcases core functionality. It's essentially a bare-bones prototype that can actually run and demonstrate the basic idea. Here's how you can create one:

**Identify the Core Functionality**

- What's the absolute minimum set of features needed to prove the concept?
- Focus on a single, crucial end-to-end process. Don't get bogged down in bells and whistles.

**Choose Your Building Blocks**

- Leverage existing code from past projects if possible.
- Look for open-source libraries or templates that can provide a foundation.
- You can also build from scratch, but prioritize efficiency here.

**Prioritize Functionality over Perfection**

- The goal is to get something working, not to craft a polished product.
- Use stubs or mocks for non-essential components to keep things simple.
- Focus on demonstrating the core logic and data flow.

**Keep it Running**

- Ensure the core functionality executes successfully, even if it's basic.
- This is the "walking" part of the skeleton - it should function at a rudimentary level.

Deploy “Hello World” on Production

1 hour

# Cloud Native

We start with production first
Production-ready from the outset

## "Hello World" Pipeline

Basic web app to return a message to determine minimal pipeline required to deploy and consume the app in a production environment

ensures that the entire deployment pipeline is functional and can handle deployments from development through to production

Deploying a simple "Hello World" application to production can be a great way to test your deployment pipeline and infrastructure. Here's a basic approach:

1. **Create a "Hello World" Application**
   This could be a simple script that prints "Hello World" to the console or a web application that returns the same message.
2. **Set Up Version Control**
   Use Git to create a repository for your code and track changes.
3. **Configure Deployment**
   Depending on your chosen technology, you'll need to set up a way to deploy your code to production. This might involve a simple script that copies files or a more complex configuration management tool.
4. **Deploy**
   Push your code to your version control system and trigger the deployment process.
5. **Verify**
   Once deployed, access your "Hello World" application (console or web interface) and confirm it displays the expected message.

All deployments are pipelined

# Cloud Native

We continuously deploy Pipelines!

## Pipelined Deployments

Software changes are automatically built, tested, and deployed through a streamlined, automated pipeline

This ensuring rapid and reliable delivery to production.

Top 5 tips to make deployment pipelines a success:

Automate Everything

- Ensure all stages of the pipeline (build, test, deploy) are fully automated to reduce human error and increase consistency. Use scripts and tools to handle repetitive tasks.

Implement Comprehensive Testing

- Integrate various levels of automated testing (unit, integration, end-to-end) to catch issues early. Ensure tests are fast, reliable, and cover critical aspects of the application.

Use Incremental Improvements

- Start with a simple pipeline and gradually add more stages and complexity. Continuously improve the pipeline based on feedback and performance metrics.

Monitor and Alert

- Set up monitoring for the pipeline to track performance and detect failures. Configure alerts to notify the team immediately if something goes wrong, enabling quick resolution.

Foster a DevOps Culture

- Encourage collaboration between development and operations teams. Promote a culture of continuous improvement, shared responsibility, and open communication to ensure everyone is aligned with the deployment goals.

Build fixed within 10 min

# Cloud Native

We continuously deploy Pipelines!

## Stop the Line

If a problem is detected in the pipeline, the entire process is halted immediately so that the issue can be addressed and fixed as quickly as possible.

This ensures that defects are not propagated further down the line, maintaining the integrity and quality of the codebase.

Let's put it into practice!

Immediate Halt

When a problem is detected, such as a failed test or build error, the pipeline stops all further actions.

Prioritize Fixes

The team's immediate focus shifts to diagnosing and fixing the issue to resume normal operations quickly.

Prevent Propagation

By stopping the line, defects are contained and do not affect downstream processes or environments.

Quality Assurance

This practice ensures that only stable and reliable code progresses through the pipeline, maintaining high standards of quality.

Continuous Monitoring

Implementing this principle requires robust monitoring and alerting mechanisms to detect issues as soon as they arise.

All tests run and are reliable

# Cloud Native

We continuously deploy Pipelines!

## Test Driven Development

Tests are written before the code, guiding the coding process and ensuring functionality.

Ensures code reliability and quality by verifying functionality before implementation, reducing bugs and improving maintainability.

The Red-Green-Refactor cycle is a fundamental process in Test-Driven Development (TDD) consisting of three steps:

Red

- Write a test for the next bit of functionality you want to add.
- Run the test and watch it fail. This step ensures that the test is valid and the functionality doesn't exist yet.

Green

- Write the minimum amount of code necessary to make the test pass.
- Focus on getting the test to pass, not on perfecting the code.
- Run the test and see it pass.

Refactor

- Refactor the code to improve its structure and readability without changing its behaviour.
- Ensure the code is clean and maintainable while keeping all tests passing.
- Run all tests to confirm that the refactored code still passes.

This cycle encourages incremental development, ensuring that each new feature is properly tested and integrated into the codebase.

Coding standards are checked

# Cloud Native

We continuously deploy Pipelines!

## Automate your toil

Remove the need for discussion and painful manual process that is time intensive for your team

Ensure your team are focusing on delivering value for your customers and not manual processes

Top tips!

Define Clear Standards

Establish and document consistent coding guidelines that all developers must follow.

Use Automated Tools

Implement linters and formatters to automatically check and enforce coding standards.

Integrate with CI/CD

Integrate these tools into your continuous integration/ continuous deployment pipeline to ensure code is checked on every commit and pull request.

Configure Commit Hooks

Set up commit hooks to run linters and formatters before code is committed, ensuring compliance early in the development process.

Environments recreated from only code

# Cloud Native

We automate everything
Everything as Code!

## Infrastructure as Code

A practice of managing and provisioning computing infrastructure through machine-readable code, rather than through manual configuration.

Ensures consistency, repeatability, and automation in deploying and managing infrastructure, reducing human error and speeding up deployments

How to get started!

1. Start with Non-Critical Infrastructure
   - Select non-essential parts of your system to minimize impact. Choose an open-source IaC tool.
2. Implement IaC
   - Make all infrastructure changes through code, committing to the repository, and avoid manual changes.
3. Automate with CI/CD
   - Create pipelines to automate provisioning and updates, including automated tests to validate changes before deployment.
4. Parameterize and Modularize
   - Create reusable modules and use variables/ configuration files to customize settings for different environments.

Environments regularly destroyed and recreated

# Cloud Native

We automate everything
Everything as Code!

## Simulated game days

Systems are intentionally disrupted to test their resilience and the team's ability to respond effectively.

This strengthens system resilience and enhances team readiness for real-world failures.

Have fun with a simulated game day.

**Define Objectives and Plan Scenarios**
Set clear goals and develop realistic failure scenarios to test system resilience and team response.

**Prepare and Communicate**
Inform the team about the game day, their roles, and ensure all necessary tools and monitoring systems are in place.

**Execute Scenarios**
Introduce the planned failures into the environment, either manually or using automation tools.

**Monitor and Document**
Observe system behaviour and team responses, documenting everything for analysis

**Conduct Post-Mortem and Improve**
Review the outcomes, identify weaknesses, and implement necessary improvements. Schedule regular game days for ongoing enhancement.

Tests complete quickly

# Cloud Native

We automate everything
Everything as Code!

## Test optimization

This involves various strategies and techniques to streamline the testing process, reduce execution time, and maintain high-quality software.

Test optimization ensures rapid feedback and efficient development cycles, enhancing productivity and code quality.

Strategies you can use:

Run Tests in Parallel
Use tools to execute tests simultaneously.

Optimize Test Suite
Remove redundant tests and focus on high-value tests.

Use Mocks and Stubs
Isolate components to avoid slow dependencies.

Implement Incremental Testing
Only test changes, not the entire suite.

Efficient Test Data Management
Use lightweight, in-memory databases or fixtures.

Debugging information is fully accessible to engineers

# Cloud Native

We do day 2 operations on day 1
Day2Ops right from the start

## Logging and Centralized Management

Aggregate and manage logs from all services in one place.

Makes sure you have efficient log management and quick access to critical debugging information.

Make all your logs easy to find!

1. Set Up Structured Logging
   - Choose a logging framework and implement it in your application.
2. Centralize Log Aggregation
   - Select a log management solution and configure log shippers to send logs to the central system.
3. Create Unified Log Access
   - Set up dashboards in your log management system for visualization and search.

Events are fully traceable to requests

# Cloud Native

We do day 2 operations on day 1
Day2Ops right from the start

## Distributed Tracing

Aggregate and manage logs from all services in one place.

Makes sure you have efficient log management and quick access to critical debugging information.

Let's get started!

1. Select and Integrate a Tool
   - Add tracing instrumentation to your code and propagate trace context across services.
2. Deploy and Configure Infrastructure
   - Set up trace collectors and a storage backend. Configure sampling rates and ensure trace data is reported timely.
3. Visualize and Monitor
   - Create dashboards to visualize traces and service maps, and set up alerts for anomalies or errors in traces.

Observability & other abilities are a core consideration

# Cloud Native

We do day 2 operations on day 1
Day2Ops right from the start

## The 3 Pillars

Aggregate and manage logs from all services in one place.

Makes sure you have efficient log management and quick access to critical debugging information.

Logs tell the story

They provide rich context about specific events, including errors, warnings, and successful actions. This detail allows you to pinpoint the "what" and "when" of an issue.

Metrics show the vitals

Metrics offer a real-time overview of key performance indicators (KPIs) like CPU usage, memory consumption, or response times. They help you understand the "how well" a system is performing and identify potential bottlenecks.

Traces map the journey

Traces track the individual journey of a request as it travels through different parts of a system. This helps you pinpoint the exact location of an issue and understand the "why" behind performance problems.

Service status is monitored

# Cloud Native

**We do day 2 operations on day 1**
**Day2Ops right from the start**

## Alerts and Notifications

Creating meaningful and actionable alerts and notifications to inform actions to be taken

Effective way to leverage monitoring tooling outputs, for those of whom are operationally concerned with the running of the system.

How to begin crafting impactful alerts and notifications

1. Focus on Actionable Events
   - Don't just inform, empower! Alerts should signal issues requiring a response, not simply report every minor blip.
2. Prioritize Clarity
   - Keep it concise and clear. The notification should instantly tell the recipient what's wrong and what (if anything) they need to do.
3. Target the Right Audience
   - Don't bombard everyone. Tailor alerts to the specific people who can address the situation.
4. Less is More
   - Avoid overwhelming users with a constant barrage of alerts. Prioritize critical information and send notifications sparingly.
5. Consider Tone and Timing
   - Be mindful of the context. A security breach requires a serious tone, while a game update might be more playful.

Service Level Indicators & Objectives negotiated

# Cloud Native

**We do day 2 operations on day 1**
**Day2Ops right from the start**

## SLAs and SLOs

SLAs and SLOs are used together to manage expectations and ensure service quality. SLAs set the expectation. SLOs are internal goals.

Have happy customers with clear expectations and a data-driven approach for your team to deliver that quality service.

### For Service Level Agreements (SLAs)

Focus on Customer Needs

Craft your SLAs around what truly matters to your customers. Don't get bogged down in technical details; focus on the user experience.

### For Service Level Objectives (SLOs)

Set Measurable Objectives

Ensure your SLOs are clearly defined and can be quantified with specific metrics. This allows for clear measurement and tracking.

Prioritize Wisely

Don't overwhelm yourself with too many SLOs. Choose a few key metrics that truly reflect the most critical aspects of your service.

Leave Room for Improvement

Aim to set SLOs that are a bit more stringent than your SLAs. This creates a buffer zone and allows your team to innovate while still meeting customer expectations.

We can recover from disasters

# Cloud Native

**We do day 2 operations on day 1**
**Day2Ops right from the start**

## Chaos Engineering

A practice to build in resilience into the system via deliberately injecting failures into a system in a controlled way

Build confidence that your software can handle unexpected events (like server crashes or network issues) without causing major outages

Understand Your System

1. Understand Your System
   - Map out your system's architecture and dependencies. Identify potential failure points where things could go wrong.
2. Form a Hypothesis
   - Predict how your system will respond to specific failures (e.g. a database outage, resource exhaustion, etc).
3. Start Small
   - Choose a safe, non-production environment for your first experiments. Inject a controlled failure based on your hypothesis.
4. Monitor and Analyze
   - Track how the system reacts to the simulated failure. Did it behave as expected? Did it recover gracefully?
5. Learn and Adapt
   - Based on your findings, refine your system or create new experiments to test different failure scenarios.

Remember, chaos engineering is an ongoing process. The more you experiment, the better you'll understand and improve your system's resilience.

We can upgrade continuously

# Cloud Native

**We do day 2 operations on day 1**
**Day2Ops right from the start**

## Blue/Green Deployments

This involves deploying the new version alongside the existing one (blue), running tests, and then switching traffic over to the new version (green) if everything is stable.

This minimizes downtime and rollback risk during upgrades.

Here's the gist.

1. Maintain two identical environments (blue: running version, green: new version).
2. Deploy and thoroughly test the new version in the green environment.
3. Switch traffic from blue to green using a load balancer (all at once or gradually).
   - All at once (cutover) is faster but riskier.
   - Gradual rollout (canary deployment) minimizes risk but takes longer.
4. Closely monitor the green environment after switching traffic.
5. Roll back to blue if critical issues occur.

Leave code better than you found it as part of story delivery

# Code as Craft

Coding practice
Code well, code fast

## The Boy Scout Rule

This emphasizes taking responsibility for the overall codebase, not just the specific functionality you're working on within a user story.

It improves overall code health, making it easier to maintain and reducing future bugs.

The Boy Scout Rule can help developers avoid technical debt by making small, consistent improvements over time.

This rule can be applied in many ways.

Look for opportunities to refactor existing code to make it cleaner, more readable, and easier to maintain such as:

- Renaming a variable
- Removing duplication
- Breaking up a long function
- Refactoring
- Improving documentation
- Optimizing a segment

All new functionality has automated tests

# Code as Craft

We test everything
Automated tests

## Shift-Left Testing

This emphasizes integrating testing activities earlier in the development lifecycle to catch bugs and potential issues as soon as possible.

Testing is no longer a separate phase at the end of development. Tests are written and executed throughout the development process, from requirements gathering to coding and deployment.

Shift-Left Testing Techniques

1. **Unit Testing**
   - Writing unit tests that verify the functionality of individual units of code (functions, classes).
2. **Integration Testing**
   - Testing how different modules or components of the system work together.
3. **API Testing**
   - Testing the functionality and behavior of APIs used by your application.
4. **Static Code Analysis**
   - Utilizing tools to analyze code for potential issues like syntax errors, security vulnerabilities, and coding best practices violations.
5. **Test-Driven Development (TDD)**
   - Writing tests before writing the actual code for a feature. This forces developers to think about the functionality from a testable perspective.

Bug fixes have automated tests

# Code as Craft

We test everything
Automated tests

## Regression Testing

Re-running existing tests to ensure software updates haven't broken previously working features. It's like a safety net to catch bugs introduced during changes.

Making sure new features don't disrupt existing functionalities.

Top tips for regression testing

1. **Test Incrementally**
   - Test new features or changes alongside their corresponding regression tests. This helps pinpoint the root cause of regressions more quickly.
2. **Go Beyond Functionality**
   - Consider non-functional aspects like performance, usability, and security during regression testing.
3. **Explore Different Devices and Browsers**
   - Test your software on a variety of devices and browsers to ensure compatibility and catch regressions specific to certain environments.
4. **Document Everything**
   - Record test results, including successes, failures, and any observed regressions. This documentation helps track progress and identify trends.

Non-functional requirements are verified with automated tests

# Code as Craft

We do day 2 operations on day 1
Day2Ops right from the start

## Example Mapping

Used within BDD, clarifies user stories with examples, ensuring everyone agrees on how the software should behave.

This prevents misunderstandings by clarifying user stories with concrete examples, leading to a smoother development process.

How to run an Example Mapping Workshop:

1. **User Story Takes the Spotlight**
   - Clearly write the user story on a central sticky note (physical or digital).
2. **Break Down the Rules**
   - Use separate notes/sections for each acceptance criteria (rules) defining the user story.
3. **Examples Breathe Life into Rules:**
   - For each rule, add specific examples illustrating how it works (multiple examples per rule are okay).
4. **Questions? Don't Sweat It!**
   - Capture any questions or confusions on separate notes/sections to address later.
5. **Discuss, Refine, and Collaborate**
   - Talk through the examples, ensuring everyone understands the user story and how the rules apply.
5. **Make it Shine!**
   - Refine the story, rules, and examples based on the discussion.

Test Driven Development
is the default practice

# Code as Craft

We test everything
Automated tests

## TDD Practical Examples

There are various ways and styles of pairing together, dependent upon the audience one maybe favoured over another.

We can adapt our style and keep things interesting whilst developing

1. **Driver/Navigator**

   Most common style. One person (driver) writes the code, while the other (navigator) reviews, suggests improvements, and keeps an eye on the bigger picture.

   **Used when:** General-purpose pairing, learning new codebases, debugging

2. **Ping-Pong**

   Rapid switching between driver and navigator roles potentially e.g. after a test pass.

   **Used when:** Keeps both programmers engaged and fosters active discussion.

3. **Unstructured**

   No predefined roles. Both programmers discuss, design, and code collaboratively in a free-flowing way. Works well for experienced programmers.

   **Used when:** Experienced engineers collaborating and experimenting with differing approaches

4. **Tour Guide**

   One programmer (usually senior) guides a less experienced programmer through the code, explaining thought processes and best practices.

   **Used when:** Mentoring, onboarding, knowledge transfer

Tests give confidence
to deploy

# Code as Craft

We do day 2 operations on day 1
Day2Ops right from the start

## What's a good test?

Good testing practices involve writing clear, independent, and measurable tests that target specific functionalities and contribute to overall software quality.

We can be confident in knowing software functions as expected, preventing bugs and saving time and resources in the long run.

A good test should be several things

1. **Clear and concise**
   - It should be easy to understand what the test is trying to achieve.
2. **Atomic**
   - It should test one specific piece of functionality or behavior.
3. **Independent**
   - The outcome of the test shouldn't rely on other tests passing or failing.
4. **Repeatable**
   - It should produce the same results consistently when run multiple times.
5. **Measurable**
   - It should have a clear pass/fail criteria based on a desired outcome.
5. **Relevant**
   - It should test something valuable to the overall quality of the software.

Failing tests fail the pipeline

# Code as Craft

We test everything
Automated tests

## Fail-Fast Gates

A checkpoint in a system designed to halt progress quickly if an issue is detected, preventing wasted effort on faulty code.

This can save time and resources by catching bugs early, preventing them from propagating through the development process.

1. **Flag trouble spots**
   Identify crucial development phases where bugs are expensive.
2. **Set the bar**
   Define clear criteria (test failures, code coverage) for each checkpoint.
3. **Automate checks**
   Use CI tools to run tests and analysis automatically at these points.
4. **Fix fast, move on**
   Address issues immediately upon a failed gate.
5. **Learn and adapt**
   Monitor and refine your fail-fast gates for optimal workflow.

By implementing these steps, you can create a development environment where issues are identified and addressed promptly.

Delete code that isn't valuable

# Code as Craft

We do day 2 operations on day 1
Day2Ops right from the start

## Dead code elimination

Removing code that is no longer used or needed, enhancing the codebase's efficiency and maintainability.

This will keep your codebase efficient and maintainable for the evolving needs of the product and stakeholders.

Try it out!

1. Choose a small module or component to start with.
2. Review and examine any identified dead code and verify its lack of usage.
3. Safely delete the unused code, ensuring that it doesn't affect functionality.
4. Run your test suite to confirm that the removal hasn't introduced issues.
5. Commit the clean code to your version control system.
6. Move on to the next module or component and repeat the process.

Goals, non-goals, anti-goals defined

# XP & Agility

We set goals every quarter
Inception meetings

## Goals & Non-Goals

A practice to align and prioritise as a team what to achieve in an iteration cycle

Attain clarity as a team through an interactive visualisation with the stakeholders in the room that need to be there to move forward effectively.

Clarify your team's vision, prioritize efforts, and establish boundaries for a project.


Actors identified

1 hour

# XP & Agility

We set goals every quarter
Inception meetings

## Actors Assessment

A practice to understand your audience the customers of your product

This enables clear understanding of of user base to help write our User Stories and focus your delivery efforts.

Try this activity to help identify your actors:

1. Add individuals, teams, and organisations who are customers of what you are creating to a board.
2. Group the actors when they will have the same expectation or deliverable from your team
3. Refine the actors to less than 10 in total.
4. The actors will be the subjects of your User Stories, so make sure you have an appropriate range

Risks & mitigations identified

# XP & Agility

We set goals every quarter
Inception meetings

## Risks & mitigations identified

This emphasizes taking responsibility for the overall codebase, not just the specific functionality you're working on within a user story.

It improves overall code health, making it easier to maintain and reducing future bugs.

Let's identify your risks and mitigations with this activity!

As a team, use a board with a traffic light color system to categorize risks based on their likelihood and impact.

List all potential risks to the project, and discuss any that need clarification.

Begin a voting session, where you can allocate multiple votes to a single risk if you believe it's particularly significant.

For the top three risks, collaboratively add mitigation strategies in the right-hand column.

Started with IPM, restated goals

# XP & Agility

We work in weekly iterations
Iterations

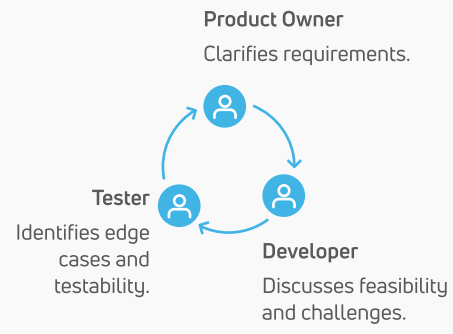
## 3 Amigos

Ensure shared understanding of a user story or feature by discussing requirements, identifying edge cases, and defining acceptance criteria before development begins.

This fosters early collaboration to identify issues, clarify ambiguities, and align on what "done" looks like.

Prepping for your IPM can help make sure it's successful. Try running a 3 amigos session!

The Three Amigos is a collaborative approach in Agile development where three key roles meet to discuss user stories:




The Product Owner briefly presents the user story's goal and context. Then together, ensure the acceptance criteria for the user stories for this iteration are clear, specific, and testable, so everyone knows that the story meets the "definition of done" and the "definition of ready" that the team agreed upon during the inception.

Daily standups,
terse and quick

# XP & Agility

We work in weekly iterations
Iterations

## Daily standup

Quickly share progress, highlight blockers, and keep the team aligned.

Keeps the team synchronized, surfaces issues early, and ensures continuous progress toward goals.

Tips for a successful stand-up:

1. **Set clear expectations**
   Define the purpose and goals of the stand-up.
2. **Timekeeping**
   Strictly enforce time limits to keep the meeting focused.
3. **Active listening**
   Encourage active participation and avoid interruptions.
4. **Visual aids**
   Use boards or digital tools to visualize progress and impediments.
5. **Rotate the facilitator**
   Distribute the responsibility to different team members.
6. **Experiment and adapt**
   Try different styles to find what works best for your team.
7. **Continuous improvement**
   Regularly assess the effectiveness of your stand-ups and make adjustments.

Work in progress has
limit of 1

# XP & Agility

We work in weekly iterations
Iterations

## WIP Limit

WIP limit of one means that only one item can be in progress at a given stage and would be ideally pair programmed/mobbed to be effective

Reduces context-switching, improves task quality, and accelerates the flow of work by minimizing unfinished tasks and bottlenecks.

**The benefits of working on 1 thing at a time:**

1. **Maximise focus**
   By limiting the number of tasks, team members can concentrate fully on one item at a time, reducing context switching and improving efficiency.
2. **Identify bottlenecks**
   If a single item is stuck, it becomes immediately apparent, allowing for quicker resolution.
3. **Reduce multitasking**
   Encourages a single-tasking approach, often leading to better quality and faster delivery.
4. **Visualize workflow**
   Makes it easier to see where work is piling up and where improvements can be made.

Ended with a demo
& retrospective

# XP & Agility

We work in weekly iterations
Iterations

## Sailboat retrospectives

A visual way for teams to reflect on their progress, identify what's helping or hindering them, and define actions for improvement.

Helps the team to pinpoint areas for improvement and take action to continue moving toward their objectives.

Retros should be fun as much as they are productive.

Here's how to run the sailboat retro:

1. Draw a sailboat (team), island (goal), wind (helping), anchor (hindering), and rocks (risks).
2. Team members write down what helps (wind), hinders (anchor), and potential risks (rocks).
3. Review and group similar items, discussing key insights.
4. Prioritize and assign concrete actions to address hindrances and risks.

Customer value
is defined

# XP & Agility

We plan work as user stories
User stories

## The 5 whys

A problem-solving method used to explore the root cause of an issue by repeatedly asking the question "Why?"

Quickly and efficiently uncover the root cause of a problem, helping teams address the core issue rather than just treating symptoms.

Figuring out the "why" of a user story involves understanding the deeper purpose or value behind the feature or task. Here's one way to uncover it:

The 5 Whys Technique

1. **Ask "Why" Multiple Times**

Keep asking "Why?" to drill down into the root cause or purpose behind the request. This helps move beyond surface-level features and get to the core value.

2. **Example**
   - Why does the user need this feature?
   - To save time.
   - Why is saving time important?
   - Because they have a busy schedule.
   - Why does having a busy schedule matter?
   - Because they need to maximize efficiency.

3. This line of questioning helps reveal the true motivation behind the request.

Stories sized to
0.5 days work

# XP & Agility

We plan work as user stories
User stories

## "Rightsizing" User Stories

Creating a User Story that is appropriate bite size chunks to be effective and focused for delivering the customer value.

Assists in effective planning, development, and testing.

To create effective User Stories that can be completed in half a day, focus on:

1. **Clarity and focus**
   Each story should have a single objective with clear acceptance criteria.
2. **Small and independent**
   Break down larger features into smaller, self-contained stories.
3. **Value delivery**
   Ensure each story provides value, even if it's small.
4. **Collaboration**
   Work with the team to identify natural breakpoints and refine stories.
5. **Continuous improvement**
   Regularly review story size and adjust your approach as needed.

Acceptance criteria
are defined

# XP & Agility

We plan work as user stories
User stories

## Acceptance criteria

Good acceptance criteria are clear, concise, and testable statements that define the conditions that a user story or feature must meet to be complete.

Well-written acceptance criteria provide a shared understanding of "done" and help guide development, testing, and product validation.

Acceptance criteria are the specific conditions that must be met for a user story to be considered complete. They bridge the gap between the abstract user story and the concrete implementation.

Good acceptance criteria should be:

1. **Clear and concise**
   - Easily understood by everyone involved.
2. **Testable**
   - Can be verified with specific test cases.
3. **Specific**
   - Clearly define the desired outcome.
4. **Independent**
   - Each criterion should stand alone.
5. **Measurable**
   - The results can be objectively assessed

Prioritised with the Product Manager

# XP & Agility

We plan work as user stories
User stories

## Value-Driven User Stories

User stories that prioritize delivering the most significant value to the customer or end-user

A value-driven user story ensures development delivers meaningful benefits to users while aligning with business goals.

A value-driven user story ensures development delivers meaningful benefits to users while aligning with business goals.

**Instead of:** As a user, I want to see a list of products.

**Try:** As a customer interested in purchasing electronics, I want to easily find products that match my needs and budget, so I can make an informed purchase.

**Checklist for Writing Value-Driven User Stories:**

- ☑ Does the story focus on a specific user role or persona?
- ☑ Is the feature described in a clear and understandable way?
- ☑ Does the story explain the tangible benefit or value to the user?
- ☑ Is the value something that can be measured or validated after implementation?
- ☑ Does the story align with broader business objectives?

Accepted with the Product Manager

# XP & Agility

We plan work as user stories
User stories

## Story acceptance

Product manager reviews completed stories to ensure they meet the "Definition of Done" and the pre-agreed acceptance criteria.

Story acceptance ensures work meets quality standards, aligns with business goals, and delivers customer value.

To get user stories accepted:

**Align** with product vision and understand the product roadmap.

**Communicate** clearly and concisely and leverage clear conditions for completion via acceptance criteria.

**Build Strong Relationships** seek feedback and gain insights into their priorities and challenges.

**Data Driven Approach** data to support your ideas, track and measure, ROI analysis, etc.

**Effective Collaborate** closely with the product manager, be open to change, and ensure the team understands the story and goals

Conform to Working Agreements

# XP & Agility

We plan work as user stories
User stories

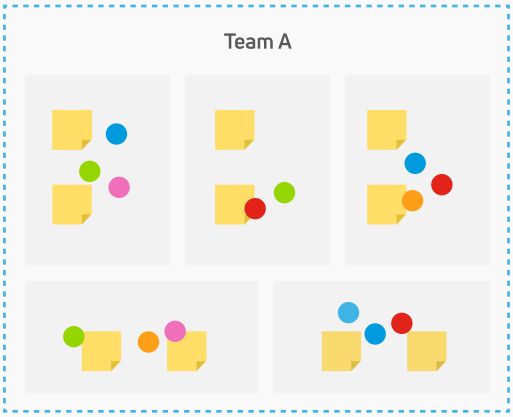
## Dot Voting on Agreements

Prioritize working agreements that matter most to the team.

Ensuring that key practices are adopted and followed for effective story management and execution.

**How to do dot voting:**

- Each team member receives a set number of dots or votes.
- They vote on which proposed working agreements they believe are most important.
- The top-voted agreements are adopted by the team.




Document just enough to proceed

# XP & Agility

We plan work as user stories
User stories

## INVEST Criteria

Teams ensure that their user stories are well-structured, actionable, and aligned with Agile principles, making them easier to implement and deliver.

It helps teams create stories that are clear, deliverable, and aligned with Agile principles.

When writing a User Story, keep the INVEST criteria in mind:

1. **Independent**
   - The story can be developed and delivered on its own, without dependencies.
2. **Negotiable**
   - The story is open to discussion and refinement, not a rigid contract.
3. **Valuable**
   - The story provides clear value to the user or customer.
4. **Estimable**
   - The story is defined enough to estimate the effort required.
5. **Small**
   - The story is manageable and can be completed within an iteration.
5. **Testable**
   - The story has clear acceptance criteria, making it easy to verify completion.

A SMART experiment is defined

# XP & Agility

We measure everything
Learn by experimentation

## Smart Experiment

SMART is a popular framework for setting goals that are Specific, Measurable, Achievable, Relevant, and Time-bound

Fosters focused and effective experiments to test your hypotheses and gather valuable insights

**Here's some tips on how you can conduct a SMART experiment:**

**Identify your hypothesis and Craft a SMART experiment:**

1. **Specific**
   - Clearly define the experiment's scope and objectives.
2. **Measurable**
   - Determine how you will measure success or failure.
3. **Achievable**
   - Ensure that the experiment is feasible within your resources and timeframe.
4. **Relevant**
   - Make sure the experiment aligns with your overall product goals.
5. **Time-bound**
   - Set a clear deadline for conducting and analysing the experiment.

Experiment is run to the agreed scale, duration and criteria

# XP & Agility

We measure everything
Learn by experimentation

## Think Big, Act Small

This approach ensures experiments are manageable and actionable, while driving toward larger goals, minimizing risk, and promoting continuous learning.

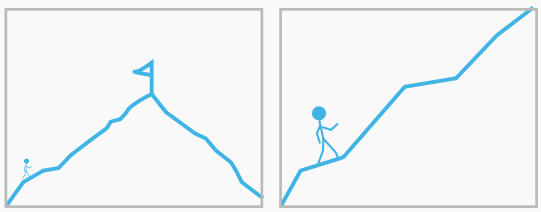
Make steady progress toward ambitious goals, while reducing risk, learning quickly from small actions, and continuously adapting based on real-world feedback.

**Think Big**

- Envision the long-term impact, broader goals, or major innovations you want to achieve.
- Leave out current user stories, feature requirements, one-off improvements, or things that would only benefit yourself.

**Act Small**

- Break those goals into smaller, testable increments or experiments, delivering value step by step, allowing for feedback and continuous improvement.
- We recommend something that can be done in a few sprints or within a couple of months.



Retrospect on if the experiment succeeded & learning generated

# XP & Agility

We measure everything
Learn by experimentation

## Experiment Retrospective

An experiment retrospective involves reviewing goals, analyzing results, identifying insights, and applying learnings to improve future efforts.

The value is continuous improvement through learning, refining actions, and making better decisions for future outcomes.

**Make sure to have a retrospective after your experiment:**

1. **Review Goals**
   - Revisit the experiment's purpose, scale, duration, and success criteria.
2. **Analyze Results**
   - Determine if the experiment met the success criteria and achieved the expected outcomes.
3. **Identify Learnings**
   - Reflect on insights gained, what worked, and what didn't.
4. **Apply Learnings**
   - Discuss how to adapt future efforts based on these insights.

Retro outcomes are actions and the actions are carried out

# Engagement

We enhance team enthusiasm
Growing engineer confidence

## Building ownership retro actions

Simple techniques on how to enact on feedback and ideas generated through a team retrospective.

Ensure that the insights gathered during retrospectives lead to actionable improvements.

To ensure action items from agile retrospectives are completed, focus on:

1. **Clear action items**
   - Define goals, assign responsibility, and set deadlines
2. **Team involvement**
   - Encourage collaboration and shared responsibility.
3. **Visualisation**
   - Use boards to track progress and celebrate achievements.
4. **Regular review**
   - Schedule check-ins and share updates.
5. **Recognition**
   - Acknowledge and appreciate team members' efforts.
6. **Leadership support**
   - Demonstrate commitment and provide resources.
7. **Empowerment**
   - Grant decision-making authority and support team growth.

By implementing these strategies, you can foster a culture of ownership and accountability within your agile team.

Rotate roles

# Engagement

We enhance team enthusiasm
Growing engineer confidence

## Rotation in action

Rotating to facilitate different team role functions

Fosters growth in teams, giving a broader perspective, developing new skills, and increase their overall confidence

**Examples of roles that you can rotate:**

**3 Amigos Session**

Each of the roles of product owner, developer, and tester can be switched out, there does not need to be the same person each session to support growth of team members

**Tech Lead**

A great way to grow technical ability and working with the business needs is the Tech Lead working with the Product Owner. Sharing the load of this role is a great way to grow and learn with the right mentoring around you.

**Agile Facilitator**

There are various chores and team health checks to be performed this does not require to be a dedicated team role. The facilitator could orchestrate and decide different ways of running daily stand-ups, retrospectives, demos, etc.

Everyone demos and talks

# Engagement

We enhance team enthusiasm
Growing engineer confidence

## Keep it Casual

Everyone in a team takes turns demonstrating and presenting

Teams can create a more dynamic, engaged, and high-performing work environment.

Tips for a good demo without the formalities.

1. **Collaborative Tone**
   - Emphasize learning and create a safe space for sharing progress, not perfection.
2. **Casual Atmosphere**
   - Use informal language, humour, and celebrate small wins to keep things light.
3. **Reduce Spotlight**
   - Pair team members for demos and rotate responsibilities to ease individual pressure.
4. **Conversational Demos**
   - Encourage interactive discussions and questions during demos for a more natural flow.
5. **Short and Focused**
   - Keep demos brief and avoid formal slides, focusing on showing work in progress.
6. **Positive Reinforcement**
   - Promote peer feedback and constructive, low-stakes critique to support team confidence.

See one, do one, teach one

# Engagement

We enhance team enthusiasm
Growing engineer confidence

## See one, do one, teach one

Watch an expert, perform the task yourself, teach the task to someone else.

This deepens understanding, builds confidence, and ensures knowledge is effectively transferred, turning learners into skilled practitioners and eventually mentors for others.

**Identify a Skill**

Choose a relevant task or skill crucial for team development.

**Demonstration**

A skilled team member demonstrates the task, explaining the 'why' behind each step.

**Practice**

The learner attempts the task with guidance and feedback, learning through mistakes.

**Teach**

The learner explains the process to another, reinforcing their understanding.

**Iterate**

Repeat the cycle for different skills, adapting to learning styles and creating a supportive, growth-focused environment.

Individual's learning & development is considered

# Engagement

We enhance team enthusiasm
Growing engineer confidence

## SWOT

Provide a clear, structured overview of an individual's strengths, weaknesses, opportunities, and threats.

This enables better self-awareness, strategic planning, and informed decision-making for personal and professional growth.

How to get started!

1. . **Create the SWOT Grid**
   - Provide a simple four-quadrant grid labelled Strengths, Weaknesses, Opportunities, and Threats.
2. **Self-Reflection**

| | |
|---|---|
| S<br>**Strengths**<br>List personal skills and qualities that set you apart. | W<br>**Weaknesses**<br>Identify areas needing improvement. |
| O<br>**Opportunities**<br>Look for external factors that could support your growth. | T<br>**Threats**<br>Consider external challenges that could hinder progress. |

3. **Action Planning**
   - Create specific goals based on the SWOT analysis, prioritizing actions to leverage strengths and address weaknesses.

Display WIP Board/backlog

# Engagement

We radiate our success
Visual working

## Making work visible

provides transparency, facilitates communication, and aids in decision-making

Getting Started with Visualizing Work:

**Choose a visualization tool**

This could be a physical board, a digital tool like Trello, Jira, or a simple whiteboard.

1. **Define your workflow**
   - Map out the stages of your work, from backlog to done.
2. **Create columns**
   - Set up columns on your board to represent each stage of your workflow (e.g., Backlog, To Do, In Progress, Done).
3. **Add cards**
   - Represent each work item (user story, task) as a card.
4. **Start filling the board**
   - Add cards to the appropriate columns based on their current status.
5. **Regular updates**
   - Keep the board up-to-date by moving cards as work progresses.

Display WIP
Board/backlog

# Engagement

We radiate our success
Visual working

## Making work visible

provides transparency, facilitates communication, and aids in decision-making

Getting Started with Visualizing Work:

**Choose a visualization tool**

This could be a physical board, a digital tool like Trello, Jira, or a simple whiteboard.

1. **Define your workflow**
   - Map out the stages of your work, from backlog to done.
2. **Create columns**
   - Set up columns on your board to represent each stage of your workflow (e.g., Backlog, To Do, In Progress, Done).
3. **Add cards**
   - Represent each work item (user story, task) as a card.
4. **Start filling the board**
   - Add cards to the appropriate columns based on their current status.
5. **Regular updates**
   - Keep the board up-to-date by moving cards as work progresses.

Display deployment
pipeline

# Engagement

We radiate our success
Visual working

## Overview Dashboard

You can monitor every stage of the pipeline from code commits to final deployments.

This can help you identify issues quickly, and ultimately ensure smoother, faster, and more reliable software delivery.

Try these key metrics in your dashboard:

1. **Build Performance**
   - Track build times, success/failure rates, and build queue times.
2. **Test Metrics**
   - Monitor test pass/fail rates, flaky tests, and the duration of test suites.
3. **Deployment Metrics**
   - Track deployment success rates, time to deploy, and rollback occurrences.
4. **Error Rates**
   - Identify and monitor frequent build or deployment errors.
5. **Resource Usage**
   - Monitor the infrastructure supporting your CI/CD pipelines, such as CPU, memory, and disk usage.
6. **Pipeline Duration**
   - Measure the total time it takes for code to go from commit to deployment.

Display product metrics

# Engagement

We radiate our success
Visual working

## NPS

A metric used to measure customer loyalty and satisfaction.

It provides a clear, actionable measure of customer loyalty, helps predict business growth, and guides efforts to improve customer experience.

How to Calculate NPS:

1. **Survey Customers**
   - Ask them to rate their likelihood of recommending your product/service on a scale from 0 to 10.
2. **Categorize Responses**
   - **Promoters (9-10):** Loyal customers
   - **Passives (7-8)**: Satisfied but not enthusiastic.
   - **Detractors (0-6)**:Unhappy customers.
3. **Calculate Percentages**
   - **% Promoters:** (Promoters / Total Responses) x 100
   - **% Detractors:** (Detractors / Total Responses) x 100
4. **Calculate NPS**
   - Subtract % Detractors from % Promoters: NPS = % Promoters - % Detractors
5. **Interpret**
   - Positive NPS indicates more promoters than detractors; above 50 is excellent.

Show & tell product and WoW

1 hour


# Engagement

We radiate our success
Visual working

## Weekly Notes

Regularly present what has been built or accomplished and sharing insights into the team's workflows, tools, and collaboration methods.

Fosters open communication, feedback, and alignment with stakeholders, ensuring that both the product and the working processes are clearly understood and continuously improved.

Example Format for Weekly Update

**Notes: Subject:** Weekly Update - [Project Name] - [Week Ending Date]

**Product Progress**

1. **Completed:** [Brief summary of completed features or tasks]
2. **Current Focus:** [What the team is working on this week]
3. **Key Metrics:** [Optional: Metrics or data highlighting progress]
4. **Demos:** [Link to demo or screenshots]

**Ways of Working:**

1. **Process Updates:** [Any changes in the team's workflow or tools]
2. **Challenges & Resolutions:** [Blockers faced and how they were resolved]
3. **Team Collaboration:** [Cross-team activities or improvements in collaboration]

**Stakeholder Feedback Request**

1. We value your input! Please share any feedback or questions.

**Upcoming Milestones**

1. [Brief overview of what's coming next]

Leaders & the team can make simple investment decisions

# Engagement

We demonstrate success
Show it's working

## Decentralizing decision-making

Distributing authority and responsibility across the team, rather than concentrating it solely with leaders.

Distributing authority and responsibility across the team, rather than concentrating it solely with leaders.

Things to think about when fostering decentralize decision-making:

**Set Clear Boundaries**

- Define authority levels and financial limits for team decisions.

**Empower Autonomy**

- Give team members ownership of their work and trust them to make decisions.

**Provide Data and Tools**

- Ensure the team has access to necessary information for informed decision-making.

**Foster Collaboration**

- Encourage peer input and shared decision-making within the team.

**Promote a Safe-to-Fail Culture**

- Encourage experimentation and use failures as learning opportunities.

Regular, investment to value-focused reporting

# Engagement

We demonstrate success
Show it's working

## Lightweight reporting

Demonstrating product success and ensuring that efforts are aligned with business objectives.

Regular Reporting is Important to help facilitate decision-making, demonstrate value, identify trends, and track progress.

Top Tips for Lightweight Reporting!

**Define Key Metrics**

- Focus on metrics aligned with product goals.

**Use Clear Visualizations**

- Create simple, easy-to-interpret charts and graphs.

**Highlight Key Findings**

- Emphasize important insights and trends.

**Provide Context**

- Relate metrics to business objectives.

**Automate Reporting**

- Streamline data collection and report generation.

**Share Regularly**

- Distribute concise reports on a consistent schedule.

Ways of working are always secure and legal

# Engagement

We deliver Safe & Secure delivery

## Security first approach

Proactively embedding security best practices into daily operations, tools, processes, and culture.

Ensuring that security is not an afterthought but an integral part of daily operations.

Let's try to foster a security-first approach:

**Data Protection**

- Ensure that all data handling practices, from storage to transfer, comply with industry security standards (e.g., encryption, secure access controls).

**Regular Security Audits**

- Conduct regular reviews of tools, processes, and systems to identify and address potential security vulnerabilities.

**Secure Communication**

- Use secure communication tools (e.g., encrypted messaging, secure file sharing) to protect sensitive information.

Golden source artefacts live on the owner's systems

# Engagement

We deliver Safe & Secure delivery

## Avoiding dependencies

Giving data ownership and control for vendor independence

Enables empowerment, sustainability, trust, and client satisfaction

Avoiding Dependencies to 3rd parties:

**Empowerment**

- Feels empowered to control and manage their systems without needing external support, fostering independence and confidence.

**Sustainability**

- Ensures that your team can sustain the system over time, even after a consultancy engagement ends.

**Trust**

- Building a trust-based relationship where you can sees us as a partner focused on your long-term success, rather than someone they must continuously rely on.

**Client Satisfaction**

- You would be more satisfied if you were in control and able to manage their assets without external dependencies.

**Share Regularly**

- Distribute concise reports on a consistent schedule.

# Would you like to order AK Way Cards for your team to play with?

Contact Us →