codurance | CRAFT AT HEART

The Minimum Valuable Increment (MVI):

Structuring Value, Requirements, and Backlogs for Effective Software Delivery

By Sandro Mancuso, co-founder of Codurance

12th of February, 2025

Acknowledgements

This e-book is the result of collaboration, insights, and constructive feedback from my colleagues at **Codurance**. I am deeply grateful for their time, thoughtful reviews, and valuable contributions, which have helped shape and refine the ideas presented here.

In particular, I would like to extend my sincere thanks to:
Abdul Al Tayib, Alan Jackson, Anna Barker, Augusto Oliveira, Lee Sanderson, Helder de Oliveira, Jennifer Torquato, Lesmes Lopez, Maciej Kowalski, Mashooq Badar, Matheus Marabesi, Matt Belcher, Nathalie Gavet, Rachel Lyons, Samuel Griffiths, Simon Shaw, and Steve Lydford.

Your generosity in sharing your experience, perspectives, and expertise has been instrumental in bringing this work to life.

— *Sandro Mancuso*

Introduction

Effective backlog management is one of the most critical yet challenging aspects of software project success. Many organisations struggle to articulate value clearly, capture requirements effectively, and align stakeholders around strategic goals. These challenges often result in stakeholders' and engineers' frustration, mismatched expectations, and wasted effort.

This article presents a new approach built on the concept of **Minimum Valuable Increments (MVIs)** and a structured approach to backlog management. This comprehensive approach addresses the common challenges in software development by redefining value, improving requirements writing, and rethinking backlog organisation and prioritisation. The goal is to enable businesses to get the most value out of their software development efforts.

Rethinking Value in Software

In Agile software development, value is traditionally defined through the lens of the customer—the users of the software. Product Owners (POs) and stakeholders often focus on user needs and product features, aiming to delight users and meet their functional expectations. While this approach ensures that software provides value to its end users, it overlooks a critical perspective: the value to the company (represented by "sponsors") building and investing in the software. Value to users is a proxy value to the company.

Focusing solely on customer-centric value can lead to decisions that, while beneficial to users, do not necessarily translate into meaningful returns for the company sponsoring the software. A feature that delights a set of users but requires a significant effort to build and maintain might provide only marginal benefits to the business. Conversely, reducing operational costs or improving system reliability might bring more tangible benefits than a feature enhancement that users appreciate but that generates little revenue or cost savings.

For example, adding a minor feature to improve user engagement might seem valuable. Still, if that feature requires complex backend changes, significant data infrastructure changes, or expensive third-party integrations, the total cost of ownership may far exceed its benefit. Similarly, adding a slightly more convenient way for users to navigate an app might bring marginal improvements. However, if the system lacks adequate monitoring, logging, or automated deployment processes, the time and cost of supporting and maintaining that feature will increase significantly.

This skewed and simplistic understanding of business value is a frequent source of frustration for both sponsors and development teams. **Sponsors invest heavily in software development**, expecting a strong ROI achieved by the product's functional evolution and the efficiency of its development and operations. While they rarely manage product backlog details, they rely on Product Owners (POs) to p**rioritise work that balances delivering valuable features with cost-effective execution**. Their frustration often stems from the perception that features are not being delivered fast enough or that development and operational costs are too high, leading them to question whether their investment is yielding the expected returns and the capabilities of the development teams.

From a development team's perspective, they see these inefficiencies daily—outdated technologies, brittle codebases, inefficient testing practices, and excessive manual work that slows down their ability to deliver new functionality. They recognise the need to improve code quality, refactor legacy systems, automate testing, and enhance deployment processes. Yet, they often struggle to secure time for these improvements because the backlog prioritises customer-facing features above all else. As a result, i**nefficient processes force engineers to work around obstacles**, leading to **slower development, higher defect rates,** and **unnecessary rework**. Over time, this cycle erodes morale, as developers feel they are constantly firefighting rather than improving and innovating.

The Role of Product Owners in Balancing Value Perspectives

Product Owners (POs) play a **critical role** in ensuring that development efforts align with business objectives and deliver meaningful value. Their deep understanding of user needs and their ability to translate them into product features are invaluable. However, many organisations face a common challenge: **backlogs are disproportionately focused on customer-facing features,** often at the expense of operational and efficiency improvements.

This imbalance is rarely intentional. The **traditional expectations placed on POs** often emphasise delivering user-centric features and measurable product outcomes. At the same time, the broader financial and operational impact of their decisions can be harder to quantify. As a result, essential improvements—such as reducing technical debt, optimising deployment processes, or improving system reliability—may not always receive the prioritisation they deserve.

Because POs t**ypically own and manage the backlog**, they are in a unique position to influence these decisions. However, they are often required to balance **competing priorities**. Without a structured way to articulate the value of operational and efficiency improvements, these can be difficult to justify against immediate customer demands.

For example, a **high-profile feature** might be prioritised because of stakeholder interest without considering the long-term maintenance burden, scalability challenges, or operational costs associated with supporting it at scale. Similarly, investments in **CI/CD pipelines, test automation, or cloud cost optimisation**—which can significantly improve delivery speed and reduce ongoing costs—are sometimes seen as "engineering concerns" rather than critical business enablers.

This challenge affects **engineering teams, sponsors, and leadership**. Sponsors expect efficient software delivery and substantial ROI, yet teams may struggle to meet long-term business objectives without a structured way to balance functional, operational, and efficiency requirements. Meanwhile, engineers often see inefficiencies in daily development work—such as slow build times, excessive manual processes, or system instability—but may find it challenging to advocate for improvements within a backlog that prioritises customer features above all else.

The **good news** is that many POs are already expanding their perspective beyond customer-centric value. By **incorporating a more structured approach to backlog management—one that explicitly accounts for functional, operational, and efficiency value—POs can strengthen their role as strategic decision-makers.**

Broadening the Definition of Value

By broadening the definition of value beyond just customer-facing features, companies can ensure they are making strategic trade-offs—prioritising work that balances user needs with business sustainability. This broader perspective means recognising that improving operational efficiency, reducing long-term costs, and enabling teams to work more effectively are just as valuable as delivering new features.

Instead of focusing purely on feature requests, everyone involved in software creation (POs, development teams, stakeholders, and sponsors) must adopt a broader view of value that considers not just end-user satisfaction but also the software's financial and operational sustainability. To adopt this broader view of value, POs, stakeholders and development teams must collaborate to understand technical risks and inefficiencies and engage with sponsors to align prioritisation decisions with business objectives.

Without this shift, the backlog will continue to be driven by a narrow and incomplete understanding of value, leading to unsustainable software development practices, frustrated engineers, and dissatisfied sponsors. By redefining value to incorporate efficiency, operational improvements and functional enhancements, teams can ensure that software investment delivers the maximum possible return.

A Holistic Understanding of Value in Software Development

Value must always be defined from the perspective of the company producing the software and/or its direct sponsors, depending on the context. Ultimately, this perspective is financial in nature. While the most significant source of value generally comes from satisfying the users' needs, this is not the only consideration. Producing and operating software comes with substantial costs, and if these costs become too high, the return on investment (ROI) diminishes—or, in some cases, may make the investment unviable. That's why it is essential to consider other dimensions of value that directly impact the profitability and sustainability of the software.

Understanding Different Types of Value

1. Functional Value

- **Key Concept:** Value is derived from features and functionalities directly impacting the user experience.
- **Examples:** New features, UI/UX enhancements, integrations, or customer-requested capabilities.
- **Rationale:** Attract and retain users, directly influencing customer satisfaction and revenue.

2. Operational Value

- **Key Concept:** Value is derived from reducing the cost and complexity of maintaining the system in production. Focuses on operational aspects like scalability, reliability, monitoring, and support.
- **Examples:** Security enhancements, better monitoring tools, cloud cost optimisation, and improvements to up-time and disaster recovery capabilities.
- **Rationale:** Minimise the operational burden and ensure the software remains cost-effective and sustainable.

3. Efficiency Value

- **Key Concept:** Value is derived from optimising the software development lifecycle (SDLC) and improving engineering productivity. The SDLC focuses on the construction, evolution, and maintenance of software.
- **Examples:** Automated testing, CI/CD pipelines, modular architecture to enable parallel development and automation tools that reduce manual tasks or lead time.
- **Rationale:** Improve the efficiency and effectiveness of the development process, lowering the delivery cost and increasing throughput.

These three types of value are deeply interconnected. Deficiencies in one can undermine the effectiveness of the others. For example, neglecting **Operational Value** (e.g., monitoring, scalability, security) can lead to system instability, degrading the user experience and reducing the **Functional Value** of the product. Similarly, failing to invest in Efficiency Value (e.g., test automation, CI/CD pipelines, modular architecture) can slow development cycles and increase the cost of delivering new features, limiting the business's ability to respond to market needs. An overly feature-centric approach prioritising **Functional Value** without considering its operational and efficiency impact can lead to unsustainable maintenance costs and technical debt.

Understanding these interdependencies is key to making informed trade-offs and maintaining a **healthy, high-performing software product**.

Balancing Priorities

By adopting a holistic view of value, Product Owners and stakeholders can make more informed decisions about what to prioritise. While it is tempting to focus primarily on **Functional Value**—as it is the most visible to customers and stakeholders—neglecting **Operational** and **Efficiency Value** can lead to **hidden costs, system instability, and slower delivery cycles.**

A well-balanced backlog ensures that Functional, **Operational**, and Efficiency requirements do not **compete against each other but reinforce one another.** For instance:

- Investing in **deployment automation** and **build optimisations (Efficiency Value) accelerates** the release of new **customer-facing features (Functional Value)**, reducing lead time and increasing responsiveness to market demands.
- Implementing proactive **monitoring and alerting (Operational Value)** ensures a more **reliable and performant application** and enhances the experience of key customer-facing features **(Functional Value)**.
- Launching a **streamlined onboarding wizard (Functional Value)** reduces manual **data entry errors** and **support escalations (Operational Value)** while accelerating new customer activation.
- Enhancing **test automation (Efficiency Value)** reduces manual testing effort and accelerates development cycles, allowing **more frequent delivery of features** that improve customer engagement **(Functional Value)**.
- Adding a performance diagnostics tool for advanced users (Functional Value) empowers customers to self-diagnose connectivity issues, reduces the support team's workload (Operational Value), and increases system transparency.

Software teams often face **trade-offs when prioritising the backlog**. The key is to ensure that improvements in one value dimension **enable rather than constrain** the others. Instead of treating these dimensions as separate categories, backlog planning should aim for **strategic alignment**, where each increment contributes to **both short-term business objectives and long-term product sustainability.**

A Unified Approach for Value and Requirements

With value redefined to focus on the company investing in the software, we can categorise requirements into the same three types: Functional, Operational, and Efficiency. Each type ties directly to a key value dimension, ensuring all aspects of business impact are addressed.

Understanding Different Types of Requirements

1. Functional Requirements: Delivering Value to Customers
Functional requirements define the features and functionality of a system. They are prioritised with end users in mind, focusing on customer acquisition, satisfaction, retention, partnerships and integrations.

Examples:

- **Increase customer satisfaction and engagement** by delivering new features requested by users.
- **Enhance usability and reduce friction** by improving the user interface and experience.
- **Expand product capabilities and unlock new use cases** by integrating with third-party systems.
- **Improve user retention and onboarding success** by simplifying and guiding the user experience.
- **Boost conversion rates and drive business growth** by optimising workflows and reducing steps in critical user journeys.

Why They Matter: Functional requirements are the most visible to stakeholders and end users. They directly impact customer satisfaction and are often central to achieving business goals, such as increasing market share or driving revenue.

2. Operational Requirements: Delivering Value through System Reliability
Operational requirements focus on the effort and cost of maintaining the software in production. They address the software's operational health, reliability, and efficiency, reducing costs and risks associated with running the system.

Examples:

- **Ensure the system can handle peak loads without degradation** by improving scalability.
- **Minimise downtime and speed up issue resolution** by enhancing monitoring capabilities for proactive detection.
- **Reduce infrastructure costs and optimise resource utilisation** through cloud efficiency improvements.
- **Guarantee high availability and business continuity** by strengthening redundancy and disaster recovery mechanisms.
- **Protect customer data and mitigate security risks** by enhancing system security and breach prevention.

Why They Matter: Operational requirements ensure that the software is cost-effective and sustainable. They minimise disruptions, reduce operational costs, and contribute directly to profitability and stakeholder confidence.

3. Efficiency Requirements: Delivering Value through Development Productivity
Efficiency requirements focus on optimising the software development lifecycle (SDLC) and improving engineering processes and quality. These requirements aim to reduce lead time, eliminate manual effort, minimise rework, and enhance team productivity.

Examples:

- **Increase deployment frequency and reduce errors** by automating testing and deployment pipelines.
- **Enable modularisation, scalability, and team autonomy** by evolving the system architecture to support parallel development efforts.
- **Speed up development iterations and reduce test execution time** by optimising database queries and indexing strategies.
- **Improve maintainability, security, and development efficiency** by upgrading frameworks or technology stacks.
- **Ensure changes are localised and easier to test** by refactoring the codebase for better modularity.
- **Reduce defects and minimise rework** by adopting better coding practices and development tools.

Why They Matter: Efficiency requirements improve team morale, lower the cost of delivery, and accelerate the realisation of value. They ensure the development process is streamlined and sustainable, enabling teams to respond quickly to changing business needs.

Type of Value	Type of Requirement	Examples
Functional Value	Functional Requirement	Features, integrations, customer journeys, UX/UI
Operational Value	Operational Requirement	Scalability, availability, monitoring, security, cloud cost optimisation
Efficiency Value	Efficiency Requirements	CI/CD, modular architecture, test automation, technology modernisation, code quality

A Single Backlog: Moving Beyond Functional vs Non-Functional

Now that we have established how requirements map to different types of value—**Functional, Operational**, and **Efficiency**—we can reassess how these categories relate to traditional requirement classifications.

The traditional separation of **functional** and **non-functional** (or technical) requirements creates silos that hinder effective backlog management and misalign development efforts with business goals. The term **non-functional** is particularly misleading, as it implies a **secondary status** compared to **functional requirements**, leading to its frequent **deprioritisation**. As a result, critical aspects such as **scalability, security, reliability,** and **automation** are often **overlooked** or **postponed** until they become urgent problems, increasing **long-term costs** and **risks**.

To address this, we must **abandon the outdated distinction** between **functional and non-functional requirements.** Instead, we should **classify all work** according to the **type of value it delivers**—whether **Functional, Operational,** or **Efficiency Value**. This approach eliminates **artificial requirement hierarchies**, ensuring that every requirement is recognised for its contribution to **business objectives** rather than an **arbitrary classification**.

Maintaining separate backlogs for functional and non-functional requirements is a **fundamental mistake. Work is work**—every backlog item represents an **investment of effort** that must be **evaluated** based on its **expected business value. Splitting backlogs** creates **fragmentation**, reduces **visibility**, and makes it harder to **balance priorities** effectively. All requirements should be managed within a single, unified backlog, whether they introduce new functionality, enhance system stability, or improve development efficiency.

A **unified backlog** ensures that **every initiative** is **transparently discussed, appropriately prioritised,** and **aligned** with **strategic objectives**. No work should be undertaken unless **explicitly represented** and **prioritised** within this backlog. By treating all **types of requirements** as **essential components** of **value creation**, organisations can break down **silos**, improve **decision-making**, and ensure that **development efforts** drive **meaningful and sustainable business impact**.

Why the existing User Story template falls short

The most widely used template in Agile software development is the **User Story format**:

As a <user>, I would like to <action>, so that <value to the user>.

While this format helps focus on end-user needs, its narrow scope and user-centric framing create several challenges, particularly when attempting to capture operational or efficiency requirements. Some systems, such as AI and Predictive Analytics, Machine-to-Machine (M2M) communication, real-time data processors, automated trading systems, backend APIs, frameworks, machine learning models, etc., don't have direct or specific human users. These limitations can hinder teams from clearly expressing the broader value of certain requirements to the organisation as a whole.

1. Overemphasis on Functional Value
The user story format is inherently suited for functional requirements that directly deliver value to the end user. However, it is not well-suited for operational requirements, efficiency requirements, or machine-to-machine systems. Applying the user story format to these concerns often results in an unnatural framing that doesn't align with how they are typically expressed.

These requirements often do not have a single, identifiable user and may focus on broader organisational goals, such as reducing costs, improving scalability, or enhancing delivery efficiency.

2. Narrow Scope of Value
By assigning the value of a requirement to a single person or type of user, the format encourages a limited view of value. This narrow framing can:

- Miss the larger business context.
- Artificially attribute value to individuals who are not direct stakeholders or not the only stakeholders.
- Undermine the strategic importance of requirements that deliver systemic or organisational value.

For example:

- *"**As an** operations person, **I would like to** automate monitoring **so that** I can react quickly to failures."*
  - While this makes sense in theory, for the "operations person", the actual value is much broader and systemic:
    - **For customers:** Reduced downtime improves their experience.
    - **For the company:** Ensured business continuity.
    - **For support teams:** Fewer incoming issues to handle.

3. Unnatural Fit for Broader Value
Requirements focused on cost reduction, delivery efficiency, or system reliability don't have a single, clear "user" who is the beneficiary of the value or stakeholder. For example:

- *"As a CTO, I would like to reduce cloud costs so that the company saves money."*
- *"As a Product Owner, I would like delivery teams to be more efficient so that they deliver features faster."*

These examples feel artificial (or forced) because they frame broad, organisation-wide objectives as if they were individual user needs.

4. Not clear who the value should be for
User stories assume a clear, single "user," but the definition of who qualifies as the "user" is often vague or misleading. In most cases, the user is seen as the beneficiary of the value represented in the story, typically associated with a customer or specific type of internal user. Value has multiple beneficiaries, but the most important is the sponsor—the entity funding the feature's development and ongoing maintenance.

Additionally, many requirements and projects involve multiple stakeholders—individuals or groups with the power to influence the feature's definition, construction, and operation. We risk overlooking critical factors if we limit our focus solely to a single beneficiary. The cost of building and maintaining a feature must make sense from a business perspective. A feature that serves the end user well but fails to justify its development and operational costs may not be a sound investment for the sponsor or the organisation.

The Need for a New Template

The limitations of the user story format reveal a need for a broader, more inclusive approach to capturing requirements. This new approach should:

- **Broaden the Scope of Value:** Communicate the holistic value of a requirement beyond a single user or persona.
- **Accommodate All Types of Requirements:** Be equally effective for functional, operational, and efficiency requirements.
- **Encourage Strategic Thinking:** Highlight the broader business impact of each requirement, aligning it with organisational goals.

Introducing the Minimum Valuable Increment (MVI)

To overcome the limitations of the traditional User Story format, we need a new approach for capturing requirements: the Minimum Valuable Increment (MVI).
This approach shifts the focus from user-centricity to business-centric value, emphasising the iterative, incremental delivery of value to the organisation. Unlike User Stories, which are often confined to functional requirements, MVIs accommodate all requirements—functional, operational, and efficiency.

Why the Name Matters
The name **Minimum Valuable Increment** encapsulates key principles:

- **Minimum:** Reflects Agile's iterative approach, delivering value in small, manageable increments.
- **Valuable:** Ensures that every increment aligns with and contributes to the business's objectives, whether it impacts users, operations, or development efficiency.
- **Increment:** Highlights the tangible change being made to the system, no matter the type of requirement.

Structure of the MVI Template

The MVI template includes **mandatory fields** that ensure clarity, focus, and business alignment and **optional fields** that provide flexibility and additional detail where needed.

MVI Core Fields — Mandatory Fields

Ensure every increment has a clear value, deliverable increment and verification method.

Mandatory Field	Definition	Example
Value	Describes the expected business value the MVI will deliver. Aligns with one or more types of value: **Functional, Operational, or Efficiency**. ***Note:*** *Always start the value statement with a verb. This will help create a better definition of the desired value.*	*Reduce downtime and improve customer satisfaction by automating system monitoring.* *< Expected value > by < change proposed >*
Increment	Specifies the work required to deliver the expected value. Defines what will be **implemented, changed, or improved**.	*Implement a monitoring solution with automated alerts for failures.*
Verification	Outlines how the team can confirm that the increment is complete and ready to be released. Defines **clear** and **objective success criteria**.	*System monitoring is live, and alerts are triggered in staging for test failures.*

Descriptive Fields (Identification & Traceability) — Optional

Help with MVI identification and traceability—useful for backlog navigation and references.

Optional Field	Definition	Example
Title	A short, high-level summary of the MVI's **Value** field. It helps with readability when scanning a backlog.	*Automate system monitoring.*
Code	A unique identifier for the MVI, helpful in linking it to **code changes, tickets, reports, or dependencies**.	*OP-MVI-2423*

Implementation Support Fields (Execution Clarity) — Optional

Provide additional context to guide implementation, reducing ambiguity and rework.

Optional Field	Definition	Example
Tasks	Lists specific **actions required** to complete the increment. Helps with **scope definition, estimation, and execution**.	• Set up a monitoring tool. • Configure alerting thresholds • Test in staging.
Examples	Provides **concrete scenarios** to clarify the intent of the MVI. It helps reduce misunderstandings and align expectations.	*When a failure occurs, an alert with details of the issue is sent to the operations team.*

Strategic Planning Fields (Backlog Organisation) — Optional

Help structure the backlog, balance work types and align with business priorities.

Optional Field	Definition	Example
Requirement Type	Categorise the MVI as **Functional, Operational, or Efficiency** to clarify the nature of the work.	*Operational*
Value Dimension	Associates the MVI with a **business driver** or **value stream**. Helps prioritise work strategically.	*Customer Acquisition, Risk Mitigation, Cost Reduction, Productivity Improvement*
Domain Area(s)	Links the MVI to a **functional area** (bounded context), helping teams structure and plan work across different domains.	*Payments, Catalogue, CRM*

Prioritisation & Estimation Fields (Decision Support) — Optional

Help prioritise the backlog by assessing impact, value, and effort.

Optional Field	Definition	Example
Target Impact	Specifies the expected measurable outcome of the MVI, providing a clear success target. Helps with accountability, tracking progress, and decision-making.	*Reduce average downtime by 30% in the next three months.*
Value Score	Represents the **relative impact** of the MVI using the **Fibonacci sequence**. It helps with prioritisation when combined with **Effort**.	*1, 2, 3, 5, 8, 13, 21, etc.*
Effort	Estimates the **level of effort** required to implement the MVI. Teams can use their **preferred estimation unit (days, story points, t-shirt sizes, etc.)**.	*2 days, 8 points, M, etc.*

MVI Examples

Here are examples of Minimum Valuable Increments (MVIs) for each type of requirement—functional, operational, and efficiency. These examples demonstrate how the MVI template can be applied to different value dimensions.

It is recommended (but not required) that the Value statement be written using the following pattern:

[**Expected value**] **by** [**change proposed**]

Functional Requirement: Allow user to reset password

MVI

- **Value:** Improve user satisfaction and reduce support requests by enabling users to reset their password securely.
- **Increment:** Implement a "Forgot Password" feature that allows users to reset their password via email verification.
- **Verification:** Test that users can reset their passwords using the feature in staging and the process meets security guidelines (e.g., token expiration, email validation).

Optional Fields

- Tasks:
  - Design UI for password reset.
  - Create a backend endpoint for token generation.
  - Integrate email service.
- **Example(s):** A user clicks "Forgot Password," receives an email with a reset link, and successfully updates their password.
- **Target Impact:** Reduce password-related support tickets by 50% within one month of release.

Operational Requirement: Improving System Reliability

MVI

- **Value:** Minimise downtime and reduce support calls by implementing automated system monitoring and alerting.
- **Increment:** Set up a monitoring tool (e.g., New Relic, Datadog) to track application performance and generate alerts for critical failures.
- **Verification:** Alerts are triggered in the staging environment for simulated failures, and the operations team receives notifications.

Optional Fields

- Tasks:
  - Integrate monitoring tool.
  - Configure alert thresholds.
  - Test alert delivery across communication channels.
- **Example(s):** When the system experiences a database connection failure, an alert with

detailed error logs is sent to the operations team.

- **Target Impact:** Reduce mean time to detect (MTTD) issues by 40% within two months.

Efficiency Requirement: Optimising the Development Process

MVI

- **Value:** Reduce lead time and improve developer productivity by automating the testing process.
- **Increment:** Implement a continuous integration (CI) pipeline with automated unit and integration tests.
- **Verification:** The CI pipeline runs successfully on every code push, with automated test results displayed in the build logs.

Optional Fields

- **Tasks:**
  - Configure CI tool (e.g., Jenkins).
  - Write unit tests for critical modules.
  - Integrate test reporting into the CI dashboard.
- **Example(s):** A developer pushes a new feature branch, and the CI pipeline executes automated tests and reports a failure for an invalid scenario.
- **Target Impact:** Decrease average time for testing and deployment from 4 hours to 1 hour within three sprints.

Benefits of the MVI Template

The Minimum Valuable Increment (MVI) template redefines how requirements are captured and communicated, offering several key advantages over traditional approaches:

1. **Broadens the Definition of Value:** Focuses on value to the business, considering functional, operational, and efficiency dimensions.
2. **Provides Clarity and Alignment:** Structured fields (Value, Increment, Verification) eliminate ambiguity and ensure shared understanding.
3. **Encourages Iterative Delivery:** Emphasises delivering small, manageable increments for reduced risk and faster feedback.
4. **Unified Backlog:** Accommodates functional, operational, and efficiency needs within a unified backlog approach. Nothing is done outside the backlog; everything can be prioritised and tracked in a single place.
5. **Facilitates Better Prioritisation:** Articulates value clearly, helping stakeholders make better decisions.
6. **Promotes Accountability:** Tracks impact through the optional field *Target Impact*, linking increments to business objectives.
7. **Fosters Team Ownership and Purpose:** Helps teams connect their work to tangible business value, enhancing motivation, ownership, and proactive value delivery.

8. **Reduces Siloed Thinking:** Bridges functional and technical requirements, fostering collaboration and holistic planning.
9. **Reduces Feature Creep:** Ensures every increment has a clear, justifiable business value, preventing unnecessary scope expansion and low-impact features from entering the backlog.

How to Organise the Backlog

A well-organised backlog is critical to delivering value efficiently and aligning stakeholders, Product Owners, and development teams. By integrating the **Minimum Valuable Increment (MVI)** template, we can create a backlog that reflects all value dimensions while maintaining clarity and focus.

Challenges with Traditional Backlogs

Traditional backlogs often fall short due to the following:

1. **Flat Backlog Structure:** A single, linear list fails to capture different types of value or strategic goals, making prioritisation more arbitrary and less aligned with business objectives.
2. **Split Backlogs and Siloed Work:** Functional requirements often dominate product backlogs, while operational and efficiency requirements are hidden in separate backlogs or deprioritised altogether. This fragmentation leads to misaligned priorities and reduced visibility.
3. **Lack of Strategic Alignment:** Backlogs often lack a clear connection between requirements and overarching business objectives, causing confusion and misalignment.

A Unified Backlog with Clear Requirement Types

Backlog items should specify their requirement type—Functional, Operational, or Efficiency—to ensure balanced prioritisation and visibility. These **are not separate backlog categories but attributes** that provide a structured way to understand how work is distributed across different value types.

Identifying Types of Requirements

Each backlog item (MVI) should include a **requirement type**, ensuring that different types of work are accounted for while keeping the backlog unified:

- **Functional** – Focused on customer-facing features.
- **Operational** – Focused on maintaining and optimising the system in production.
- **Efficiency** – Focused on improving the development process and SDLC.

Categorising MVIs by **requirement type** helps teams maintain a **holistic view of value in the backlog**, preventing technical improvements from being deprioritised or overlooked.

Unified Backlog with Strategic Prioritisation

Rather than splitting the backlog into **separate functional and technical streams**, all MVIs are **managed in a single backlog** and prioritised based on **business value, not requirement type.**

This structure **aligns naturally with cross-functional teams**, which can handle all three types of work autonomously. With this approach:

- Teams avoid bottlenecks caused by dependencies on separate departments.
- Operational and efficiency improvements become an integrated part of backlog planning, not an afterthought.
- Prioritisation remains **value-driven**, ensuring that technical sustainability is considered alongside feature development.

Balancing Across Requirement Types
A well-balanced backlog considers all three types of requirements to ensure long-term product sustainability. While **setting intentional allocation targets** (e.g., 50% functional, 30% operational, 20% efficiency) can serve as **a guideline for holistic thinking**, the actual distribution may vary significantly depending on **business needs, value streams, or domain areas.**

More important than fixed ratios is ensuring that **requirements are prioritised based on value** rather than categorisation alone. Teams should take a **holistic view of value**, ensuring that functional, operational, and efficiency improvements work **together** to drive sustainable and impactful outcomes.

Practical Steps for Implementation

1. **Categorise Requirements:**
    a. Label each MVI with its type: functional, operational, or efficiency.
    b. Optionally, add tags for value dimensions (e.g., cost reduction, scalability, user acquisition).

2. **Prioritise Based on Value:**
    a. Use the “Value” field of the MVI to facilitate prioritisation discussions.
    b. Consider both short-term impact and long-term strategic goals.

3. **Visualise the Backlog:**
    a. Use tools like Kanban boards or backlog management software to create clear visual distinctions between categories.
    b. Example: Color-code MVIs by type or grouping them into swimlanes.

4. **Review and Adjust Regularly:**
    a. Conduct regular backlog grooming sessions to reassess priorities and ensure alignment with changing business needs.

Example of an Organised Backlog

Priority	Value Dimension	MVI (Value statement)	Req. Type	Target Impact
1	Customer Retention	Reduce support requests and improve user experience by enabling users to reset passwords securely.	Functional	Reduce support tickets by 50%
2	Risk Mitigation	Minimise downtime and reduce support calls by implementing automated system monitoring and alerting.	Operational	Reduce downtime by 40%
3	Productivity Improvement	Reduce lead time and improve developer productivity by automating the testing process.	Efficiency	Reduce lead time by 75%
4	Scalability	Improve system scalability for peak traffic by enabling auto-scaling on AWS	Operational	Support 2x traffic without downtime
5	Customer Acquisition	Simplify onboarding by creating a guided wizard.	Functional	Increase user retention by 20%

Benefits of This Approach

- **Visibility Across Value Dimensions:** Ensures all types of requirements are visible and prioritised based on business impact.
- **Balanced Investments:** Allocates resources effectively across functional, operational, and efficiency goals.
- **Unified Communication:** Bridges gaps between stakeholders, Product Owners, and developers by consolidating all work in one backlog.
- **Strategic Agility:** Enables the team to adapt quickly to changing priorities while focusing on delivering value.

Macro-Organisation of the Backlog

While MVIs provide clarity and focus at the execution level, they are too granular for strategic product planning. The backlog must be structured at multiple levels to ensure that development efforts align with business objectives and provide a clear direction over time.

A well-structured backlog follows a **hierarchical model** that connects **high-level strategic objectives with actionable increments,** ensuring that all work contributes to broader business goals.

Defining a Roadmap With a Well-Structured Backlog

A **roadmap** is a forward-looking plan that defines the **primary business outcomes** a company or a software product aims to achieve over time. To effectively execute a roadmap, backlog items must be structured to provide a clear path from strategic intent to execution. This structured approach relies on the hierarchy of **Milestones, Goals, MVIs,** and **Tasks,** which provides a systematic way to connect high-level business objectives with day-to-day execution. Each level serves a distinct purpose:

Milestone – Major Business Achievements

- Represent the expected **business capabilities** and **roadmap checkpoints** that guide product backlog creation and refinement.
- Enable periodic high-level **replanning of the roadmap** based on evolving business priorities.
- Typically spans **3 to 6 months** and provides a clear direction for the team.

Example: *Expand into the European market by launching a multi-language version of our platform.*

Goal – **Concrete Step Toward Achieving a Milestone**

- Represent the **key deliverables and intermediate outcomes** needed to fulfil a Milestone.
- Provide a clear, business-relevant focus by **grouping MVIs around meaningful achievements**, not arbitrary tasks.
- Typically spans **2–6 weeks**, depending on the complexity of the milestone.

Example: *Enable multi-language support for all core application pages.*

MVI (Minimum Valuable Increments) – Delivering Value in Small Increments

- The smallest, **independently valuable unit of work** that contributes to achieving a Goal.
- Must **deliver measurable business value**—whether functional, operational, or efficiency-related.
- Typically spans **2–5 days**, enabling rapid, incremental delivery of meaningful outcomes.

Example: *Implement language toggle functionality with automatic detection of user preferences.*

Task – The Execution Details

- Represent the **low-level implementation steps** required to complete an MVI.
- Provide concrete, **actionable instructions** to guide development and execution.
- Typically spans **0.5–3 days**, depending on complexity.

Example: *Add language selector UI component and link it to the backend language preference API.*

Using the MVI Template Across Levels

One key recommendation is to apply the **MVI template** to Milestones and Goals in addition to MVIs. This recommendation ensures that all levels of the hierarchy are aligned with business value and include measurable outcomes.

The following example shows how a **Milestone** can break down into multiple Goals, each Goal into several **MVIs**, and MVIs into **Tasks**. This hierarchical organisation provides a clear path from strategic objectives to actionable tasks.

Hint: Start the title and value statement of the Milestones, Goals, and MVIs with a verb. This approach helps you focus on the expected value alongside the necessary increment.

Milestone 1:

Milestone	Launch a Mobile App for a New Customer Segment
Value	Increase market share by launching a mobile app for a new customer segment.
Increment	Deliver a complete mobile app with MVP features, app store readiness, and initial marketing campaigns.
Verification	The app is live in the Apple App Store and Google Play
Target Impact	• Achieve a 10% increase in new customer sign-ups within three months post-launch. • Minimum of 1,000 downloads in the first month (combined).

Goal 1:

Goal 1	Enable Secure User Purchase Transactions
Value	Enhance transaction security by implementing encrypted payments and user authentication.
Increment	Deliver secure login, payment gateway integration, and transaction history views.
Verification	All features are tested and meet security compliance standards.
Target Impact	Achieve a 95% task success rate during user testing.

MVIs for Goal 1:

MVI 1 (Goal 1)	Implement Secure Login
Value	Strengthen user trust and ensure data protection compliance by implementing a secure authentication mechanism.
Increment	Implement authentication using OAuth 2.0.
Verification	Login functionality is tested and meets all security requirements.
Target Impact	Reduce failed login attempts by 20% compared to existing solutions.
Tasks	1. Implement backend authentication using OAuth 2.0 (e.g., Google, Apple, email-based login). 2. Develop frontend UI components for login, registration, and error handling. 3. Implement secure session management, including token storage and expiration policies. 4. Integrate multi-factor authentication (MFA) if required. 5. Write integration tests for authentication workflows.
Req. Type	Functional

MVI 2 (Goal 1)	Integrate Payment Gateway
Value	Ensure secure transactions and achieve PCI-DSS compliance by integrating a certified payment gateway.
Increment	Integrate Stripe for payments
Verification	Payment processing is tested and secure.
Tasks	1. Set up Stripe API keys and configure payment processing. 2. Implement frontend UI components for payment processing. 3. Develop backend payment processing logic, including webhook handling. 4. Add robust error handling for declined transactions and expired cards. 5. Implement security measures, such as tokenised card storage and fraud prevention checks. 6. Write unit and integration tests to ensure payment flow reliability.
Req. Type	Functional

MVI 3 (Goal 1)	Display Transaction History
Value	Improve financial transparency by enabling users to view their transaction history.
Increment	Build a “Transaction History” screen.
Verification	Users can view accurate transaction records.
Tasks	1. Design and implement UI for transaction history with filters and sorting options. 2. Develop backend API to retrieve and paginate transaction records securely. 3. Ensure data integrity by validating transaction history accuracy. 4. Implement caching or performance optimisations for high-volume transactions. 5. Write unit and integration tests to verify transaction data retrieval and display.
Req. Type	Functional

MVI 4 (Goal 1)	Enhance Fraud Detection for Payment Processing
Value	Reduce financial risks by detecting and preventing fraudulent transactions in real time.
Increment	Implement an automated fraud detection system that flags suspicious transactions for review.
Verification	Fraudulent transactions are detected and blocked in a staging environment with test scenarios.
Tasks	1. Research and select a fraud detection tool or develop custom rule-based anomaly detection. 2. Integrate fraud detection service (e.g., Stripe Radar, machine learning model, or rule-based detection). 3. Implement logging and alerts for flagged transactions. 4. Develop an admin dashboard for reviewing flagged transactions. 5. Run test scenarios to ensure accurate fraud detection with minimal false positives.
Req. Type	Operational

Goal 2:

Goal 2	**Improve Customer Onboarding Experience**
Value	Increase user retention by simplifying the onboarding process.
Increment	Deliver an intuitive onboarding wizard and in-app tips.
Verification	New users complete the onboarding process within 5 minutes on average.
Target Impact	Increase Day 7 user retention rate by 15%.

MVIs for Goal 2:

MVI 5 (Goal 2)	**Create Onboard Wizard**
Value	Accelerate account setup by implementing a guided onboarding wizard.
Increment	Develop a guided onboarding flow.
Verification	The onboarding flow is completed successfully in staging tests.
Tasks	1. Design UI/UX for the onboarding wizard. 2. Implement frontend components for step-by-step onboarding. 3. Develop backend support for onboarding data persistence. 4. Add validation for required onboarding inputs. 5. Conduct usability testing and gather feedback for improvements.
Req. Type	Functional

MVI 6 (Goal 2)	**Add Contextual Help Tips**
Value	Enhance user guidance by adding contextual help tips throughout key workflows.
Increment	Implement in-app help popups.
Verification	Help tips are triggered in relevant app sections.
Tasks	1. Identify key areas in the app where help tips are needed. 2. Design UI/UX for contextual tooltips and help popups. 3. Implement tooltip functionality with triggers based on user interactions. 4. Store and manage user state (e.g., dismissing help tips permanently) 5. Validate that tooltips appear in the proper context and do not interfere with usability.
Req. Type	Functional

MVI 7 (Goal 2)	**Automate User Onboarding Validation**
Value	Reduce manual verification effort by automating validation of user-provided information during onboarding.
Increment	Implement automated checks for validating email, phone numbers, and identity verification documents.
Verification	Automated validation correctly processes valid and invalid user inputs in staging without manual intervention.

Tasks	1. Integrate an automated email verification service (e.g., SendGrid, AWS SES). 2. Implement phone number validation with an SMS OTP mechanism. 3. Automate identity document verification (e.g., OCR-based ID verification or third-party service). 4. Add real-time feedback to the onboarding form to notify users of validation issues. 5. Measure and compare time saved vs. manual onboarding verification.
Req. Type	Efficiency

Structuring the Backlog for Strategic Alignment

Structuring the backlog using a hierarchical approach—Milestones, Goals, and Minimum Valuable Increments (MVIs)—provides a clear and systematic way to refine backlog items according to their alignment with business value. This structure allows teams to assess whether their planned work meaningfully contributes to strategic objectives and ensures that no effort is wasted on misaligned or low-value increments.

Milestone: Launch a Mobile App for a New Customer Segment			
	Goal 1: Enable Secure User Transactions		
		MVI 1: Implement Secure Login	**Functional**
		MVI 2: Integrate Payment Gateway	**Functional**
		MVI 3: Display Transaction History	**Functional**
		MVI 4: Enhance Fraud Detection for Payment Processing	**Functional**
	Goal 2: Improve Customer Onboarding Experience		
		MVI 5: Create Onboard Wizard	**Functional**
		MVI 6: Add Contextual Help Tips	**Functional**
		MVI 7: Automate User Onboarding Validation	**Functional**

At the highest level, **Milestones** represent significant business objectives that provide strategic direction for the next 3 to 6 months. These are the high-level commitments that an organisation aims to achieve within a defined timeframe. Teams must decompose a Milestone into Goals—each representing a meaningful step towards fulfilling that broader objective—to deliver it successfully. By visualising the backlog in this structured way, teams can validate whether the combination of Goals under a Milestone is sufficient or ideal to achieve it. If gaps or misalignments exist, the Milestone itself may need to be refined, or additional Goals may need to be introduced.

Similarly, each **Goal** needs to be broken down into actionable and valuable increments **(MVIs)**. By visualising Goals and their associated MVIs, teams can determine whether the planned MVIs are enough to fulfil the Goal, whether any increments are missing, or whether there are MVIs

that might not contribute to the Goal. This visualisation helps eliminate unnecessary work and ensures all backlog items are directly aligned with strategic and tactical objectives.
This structured backlog allows for two complementary approaches:

- **Top-down backlog creation:** When defining a backlog from scratch, the process starts at the **Milestone** level, establishing the high-level business outcome. From there, the Milestone is broken down into **Goals**, representing major deliverables and value-driven objectives. The next step is decomposing the prioritised Goals into **MVIs**, ensuring that each increment contributes directly to delivering the intended business value.
- **Bottom-up backlog review and refinement:** When refining an existing backlog, teams can work in the opposite direction, ensuring alignment and completeness. Starting at the **MVI** level, teams can inspect whether each MVI meaningfully contributes to its **Goal**. If an MVI does not support its parent Goal, it may need to be re-evaluated, removed, or moved elsewhere. The same applies to **Goals**—teams can review whether each Goal is necessary and whether its combination of MVIs is sufficient to fulfil it. Finally, at the **Milestone** level, teams can validate that all Goals collectively drive the business outcome, refining them if necessary.

By adopting this hierarchical approach to backlog visualisation, teams gain clarity, alignment, and the ability to continuously inspect and adapt their backlog. This approach ensures that development efforts remain focused on delivering the most valuable outcomes while maintaining the agility to respond to evolving business needs.

Why This Hierarchy Works

1. **Strategic Clarity:** Milestones provide a long-term vision, aligning the roadmap with business objectives.
2. **Tactical Alignment:** Goals break Milestones into manageable chunks, guiding teams in the right direction.
3. **Actionable Execution:** MVIs ensure that day-to-day work contributes directly to Goals and Milestones.
4. **Flexibility:** The periodic reassessment of Milestones, Goals, and MVIs allows teams to adapt to changing priorities without sacrificing their long-term goals.
5. **Measurable Outcomes:** Applying the MVI template ensures that value is articulated and quantifiable at every level.

Drive Strategic Insights with Strategic Planning Fields

In an MVI-driven backlog, categorising requirements with ***Strategic Planning Fields*** provides powerful tools for organising and planning work across various perspectives. By applying consistent categorisation to all requirements, teams can create product roadmaps and backlogs that address business needs systematically and effectively. Let's explore the use of Strategic Planning Fields: **Value Dimension, Domain Area, and Requirement Type**.

Value Dimension: Understanding Business Impact

Purpose	Capture the primary business impact the MVI contributes to, aligning it with strategic goals.
Examples	• **Customer Acquisition:** Features that attract new customers. • **Cost Reduction:** Efforts to reduce operational or development costs. • **Risk Mitigation:** Improvements that lower security, reliability, or compliance risks. • **Productivity Improvement:** Enhancements that improve developer efficiency or reduce lead time.
Use Case	Ensure that backlog priorities align with strategic objectives and value streams. For example: • During a growth phase, prioritise MVIs tagged with Customer Acquisition. • When costs are a concern, focus on MVIs tagged with Cost Reduction.

The Value Dimension field helps teams categorise MVIs according to the business value stream they contribute to. Different MVIs within the same Goal may contribute to different value dimensions. This categorisation provides multiple benefits:

- **Clarifies the business impact of each MVI:** Product Owners and teams gain visibility into how each MVI contributes to broader business goals. This visibility ensures a balanced investment across customer-facing features, cost optimisations, risk mitigations, and productivity improvements. It prevents an overemphasis on any single type of value and helps leadership track and adjust priorities effectively.
- **Helps prioritise work strategically:** Product owners can balance investments across different value dimensions according to business needs.
- **Surfaces potential team or skill dependencies:** For example, an MVI related to **Risk Mitigation** might require security specialists. In contrast, one related to **Cost Reduction** might depend on infrastructure or DevOps teams.

By explicitly assigning a **Value Dimension** to MVIs, teams can better understand how their work aligns with overall business objectives.

How to Choose the Right Value Dimension?
The Value Dimension field should be customised based on the company's business model, strategic objectives, value streams, and industry focus. Some organisations may need more granular value streams, while others may prefer a simplified model. For example:

- A **B2B SaaS** company may emphasise **Customer Acquisition, Retention**, and **Scalability**.
- A **fintech company** may prioritise **Risk Mitigation, Compliance,** and **Revenue Generation.**
- A **developer tooling** company may focus on **Developer Enablement, Productivity,** and **Time-to-Market Acceleration.**

What happens when one MVI or Goal impacts more than one Value Dimension?
It is common to find MVIs and Goals contributing to multiple value dimensions. However, to better understand and facilitate strategic planning, choose the primary value dimension that originated the need for the MVI or Goal.

Domain Area: Identifying Impacted System Areas

Purpose	Group MVIs by functional areas of the product to track and prioritise work across specific business domains.
Examples	Payments, Orders, Catalogue, User Management.
Use Case	Analyse and prioritise work based on the needs and health of specific areas. For example: • If the "Payments" area has high bug counts and performance issues, prioritise Operational and Efficiency requirements to mitigate risks. • For a "Catalogue" area lacking competitive features, prioritise Functional requirements to boost customer acquisition and retention.

The Domain Area field helps teams understand which parts of the system will be impacted by a given MVI. This understanding provides several advantages:

- **Assign the right people to work:** Product Owners and engineering leads can determine which teams or individuals have the expertise to implement the change by identifying the affected system area.
- **Understand potential dependencies:** By understanding which areas of the system need to be changed, teams can identify dependencies early and avoid late-stage conflicts.
- **Recognise complexity and constraints:** Some system areas may have regulatory constraints, require specialised knowledge, or involve integration with external services. The Domain Area field helps highlight these factors.
- **Organise work by bounded contexts:** As in Domain-Driven Design (DDD), the Domain Area field can indicate bounded contexts and sub-contexts, ensuring that teams working within specific domains can effectively plan and manage work.

By identifying and explicitly specifying the domain areas impacted by each MVI and each Goal, teams can structure backlogs to facilitate work distribution, improve planning, and proactively manage dependencies across teams.

Requirement Type: Defining the Nature of the Work

Purpose	Identify the category of the requirement to ensure a balanced distribution of work.
Examples	• **Functional:** Features and functionalities focused on end users. • **Operational:** Improvements to system performance, reliability, or scalability. • **Efficiency:** Optimisations to SDLC and improvements to engineering processes and quality.
Use Case	Assess how much effort is allocated to each requirement type for a Milestone, Goal, Value Dimension or Domain Area. For instance: • Ensure sufficient investment in Operational and Efficiency requirements to maintain system health and delivery velocity. • Avoid over-prioritising Functional requirements at the expense of long-term system sustainability.

The **Requirement Type** field categorises MVIs as **Functional, Operational, or Efficiency** requirements. This categorisation helps teams:

- **Understand the type of value an MVI addresses** – Whether an item delivers a user-facing feature (Functional), improves system reliability (Operational), or enhances developer productivity (Efficiency).
- **Ensure balanced backlog planning** – Avoiding a feature-heavy backlog while neglecting necessary operational and efficiency improvements.
- **Inform prioritisation decisions** – For example, if an Operational requirement prevents downtime, it may need higher priority than a lower-impact Functional requirement.

Leveraging Strategic Planning Fields for Backlog Management

Assigning **Value Dimension, Domain Area,** and **Requirement Type** to MVIs provides teams and Product Owners with essential insights into the impact, dependencies, and business value of backlog items. These fields help teams plan and refine their backlog to ensure alignment with business objectives while also optimising the distribution of work across teams. Let's revisit our Goals and MVIs assigning these tags.

Goal 1: Enable Secure User Transactions			
MVI	**Value Dimension**	**Domain Area**	**Req. Type**
Implement Secure Login	**Risk Mitigation** – Enhances security and prevents unauthorised access.	User > Authentication	Functional
Integrate Payment Gateway	**Revenue Generation** – Enables users to make payments, driving revenue.	Payments > Gateway	Functional
Display Transaction History	**Customer Retention** – Improves user trust and transparency.	Payments > Transactions	Functional
Enhance Fraud Detection for Payment Processing	**Risk Mitigation** – Reduces fraud and financial loss for the business.	Payments > Fraud Detection	Operational

Goal 2: Improve Customer Onboarding Experience			
MVI	**Value Dimension**	**Domain Area**	**Req. Type**
Create Onboarding Wizard	**Customer Acquisition** – Helps new users get started smoothly.	User > Onboarding	Functional
Add Contextual Help Tips	**User Engagement** – Enhances usability, reducing confusion and churn.	User > Onboarding	Functional
Automate User Onboarding Validation	**Productivity Improvement** – Reduces manual work, improving development speed.	User > Onboarding	Efficiency

Unlocking Strategic Insights by Combining Strategic Planning Fields

While each individual field provides value, the real power comes from **combining them** to gain deeper insights into how work is structured across the backlog.

Strategic Insight for Backlog Creation:

- Understanding **how Value Dimensions impact different Domain Areas** ensures that business priorities are **reflected across the system**.
- **Example:** A company focused on **risk mitigation** might find that fraud-related MVIs impact multiple domain areas **(Payments, authentication, and user accounts)** and require coordinated execution across teams.

Strategic Insight for Backlog Refinement:

- Combining **Requirement Type with Domain Area** allows teams to ensure that each system area gets **a balanced mix of functional, operational, and efficiency improvements**.
- **Example:** If the **Checkout Domain** has multiple Functional MVIs (new payment options) but lacks Operational MVIs (fraud prevention) and Efficiency MVIs (deployment automation), the backlog may be **unbalanced**, creating risks for future stability.

Example: A company launches a Customer Loyalty Program initiative. By reviewing tags across Value Dimensions, they realise:

- **Customer Retention (Value Dimension)** MVIs are focused on **Rewards and Discounts (Domain Area).**
- The Payments (Domain Area) team lacks the necessary knowledge to handle changes regarding loyalty-based transactions.
- The backlog is **heavily functional**, but **Operational and Efficiency improvements** (e.g., fraud detection, performance optimisations) are missing.

By combining Strategic Planning Fields, the Product Owner realigns the backlog, ensuring that dependencies are addressed before rollout, preventing costly delays and last-minute fixes.

A Strategic and Holistic Approach to Backlog Management

By leveraging **Value Dimensions, Domain Areas, and Requirement Types,** Product Owners and teams can **transform backlog management from a reactive process into a proactive strategy.**

- **Value Dimension** ensures that work aligns with **business and product strategy.**
- **Domain Area** helps teams **assign work effectively, manage dependencies, and plan resources.**
- **Requirement Type** ensures a **balanced investment** across feature development, system health, and delivery efficiency.

By combining these perspectives, teams gain **complete visibility into their backlog,** enabling them to prioritise work effectively, anticipate dependencies, and ensure that all efforts contribute meaningfully to the organisation's goals. Whether creating new backlog items or refining existing ones, this structured approach ensures that teams are **constantly working on the right things in the right areas with the right balance.**

Prioritisation in MVI-Based Backlogs

Prioritisation is one of the most critical aspects of backlog management. Without a structured approach, teams risk investing effort in increments that do not deliver meaningful value or aligning work in a way that fails to balance effort and impact. The **MVI approach** introduces three key fields—**Target Impact, Value Score, and Effort**—to support structured prioritisation and help teams make informed decisions about what to build next.

Balancing Value, Effort, and Target Impact

When prioritising backlog items, teams must balance **high-impact work** with **feasible execution timelines.** A structured approach ensures that development efforts align with business objectives while remaining adaptable to Agile workflows.

A **common mistake** in backlog management is arbitrarily assigning a **Value Score without a clear Target Impact**. The Target Impact field is key in prioritisation because it defines **how success will be measured**. Without it, the Value Score becomes subjective and challenging to compare across MVIs.

Thus, before an MVI is assigned a **Value Score**, teams should ensure that:

1. **The Target Impact is clearly defined** with measurable outcomes.
2. **There is consensus on the expected benefits**—assigning a Value Score becomes arbitrary when there are no clear expectations.
3. **The Target Impact justifies the investment**—MVIs with a vague or weak impact should be refined before prioritisation.

By using **Target Impact as the foundation, Value Score as the relative measure, and Effort as the feasibility indicator,** teams can **prioritise backlog items in a structured, data-driven manner.**

Using Value Score for Prioritisation

The **Value Score** represents the **relative impact** of an MVI using **Fibonacci numbers (1, 2, 3, 5, 8, 13, etc.).** A higher score indicates a **more significant business value**, helping teams prioritise work with the **most strategic benefit**.

However, this score should not be **subjective or arbitrary** but based on the **Target Impact.** If an MVI lacks a **clear Target Impact**, its Value Score is meaningless.

Example Scale:

- **1–3:** Low business impact.
- **5–8:** Moderate impact, contributes to core objectives.
- **13+:** High impact, strategic priorities.

Example Mapping of Target Impact to Value Score:

MVI	Target Impact	Value Score
Automate system monitoring	Reduce downtime by 30% in 3 months	8
Improve login security	Reduce failed login attempts by 20%	5
Optimise database queries	Decrease query response time by 10%	3
Refactor old logging code	Improve logging maintainability (minor impact, non-quantifiable)	2

Automating system monitoring (8) takes precedence over **query optimisation (3)** due to its **stronger Target Impact**.

Using Effort for Prioritisation

Unlike the Value Score, the **Effort** field remains flexible, allowing teams to use their preferred estimation method (e.g., **days, story points, and T-shirt sizes**). The Effort field represents the **work required to complete an MVI.**

Example Scale (if using days):

- < **1 day:** Trivial
- **1–3 days:** Small effort
- **4–7 days:** Moderate effort
- > **7 days:** Large effort

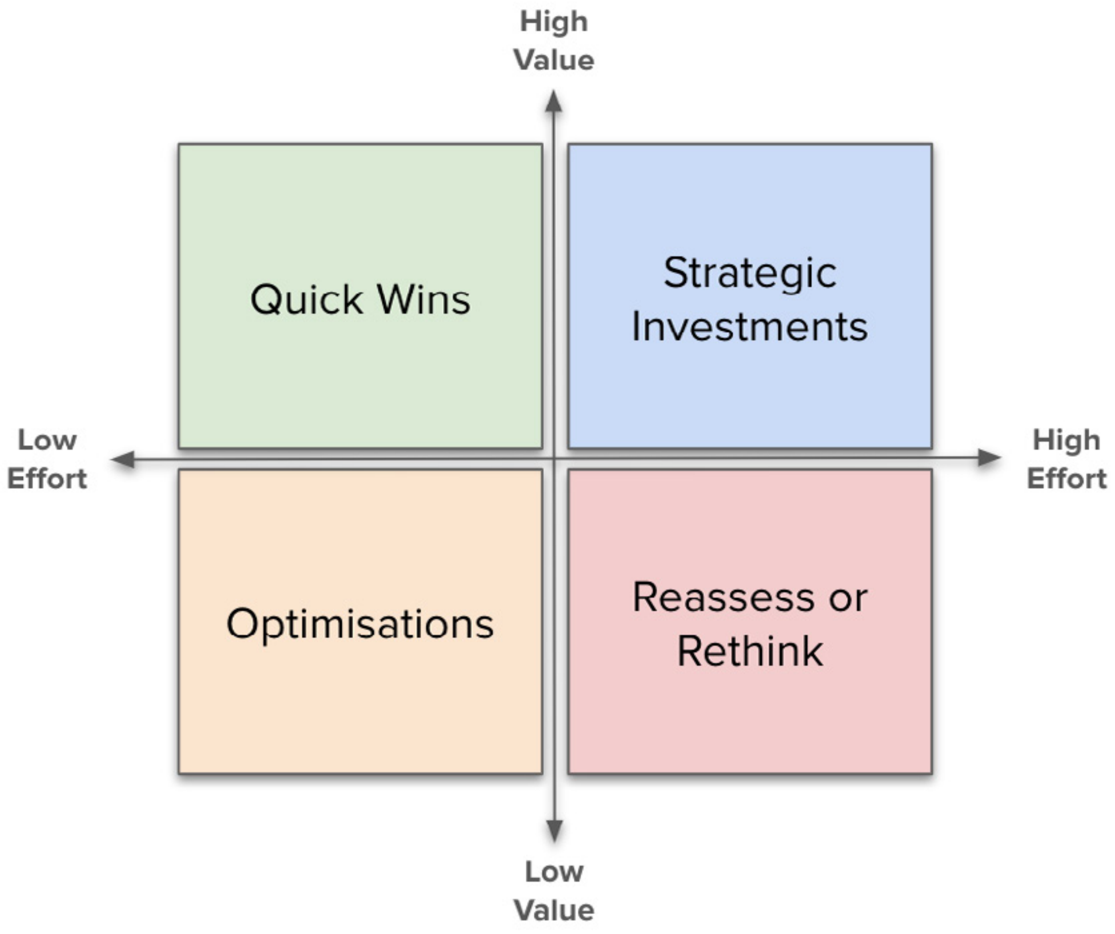
Example:

MVI	Target Impact	Effort (Days)
Automate system monitoring	Reduce downtime by 30% in 3 months	5 days
Improve login security	Reduce failed login attempts by 20%	3 days
Optimise database queries	Decrease query response time by 10%	2 days
Refactor old logging code	Improve logging maintainability (minor impact, non-quantifiable)	8 days

Here, **improving login security (3 days)** may be **prioritised before** automating monitoring (5 days) if there is a pressing security concern.

Prioritisation Quadrant: Agile & Business-Aligned Approach

Once **Target Impact, Value Score, and Effort** are assigned, teams can plot MVIs into the following **Agile-friendly prioritisation quadrant:**



This structure helps teams **balance immediate value with long-term strategic planning**, ensuring that **incremental gains and major initiatives** receive the appropriate attention.

Putting It Together: Prioritisation Matrix

By combining **Target Impact, Value Score, and Effort**, teams can visualise MVIs in a **prioritisation matrix:**

MVI	Target Impact	Value Score	Effort (Days)	Quadrant
Automate system monitoring	Reduce downtime by 30% in 3 months	8	5	Strategic Investment
Improve login security	Reduce failed login attempts by 20%	5	3	Quick Win
Optimise database queries	Decrease query response time by 10%	3	2	Optimisation
Refactor old logging code	Improve logging maintainability (minor impact, non-quantifiable)	2	8	Reassess or Rethink

Key Insights with Target Impact:

- **"Improve login security" (Quick Win):** Although its Value Score (5) is lower than system monitoring (8), its Target Impact is well-defined and easier to achieve.
- **"Automate system monitoring" (Strategic Investment):** High value, but more effort is required—however, the strong Target Impact (30% downtime reduction) justifies scheduling it soon.
- **"Optimise database queries" (Optimisation):** A low Value Score and minor impact—it's an improvement but not a high priority.
- **"Refactor old logging system" (Reassess or Rethink):** Requires significant effort but does not provide enough immediate business value.

Prioritisation Best Practices

Technique	Description
Regularly reassess priorities.	Business needs to evolve, so revisit the Value Score vs. Effort analysis regularly.
Use a collaborative approach.	Engage stakeholders in assigning Value Scores to ensure alignment with strategic goals.
Balance high-impact vs. quick wins	Don't focus only on high-value, high-effort work—quick wins can provide incremental improvements.
Refine estimates over time.	If an MVI's effort is unclear, start with a rough estimate and refine it as more information becomes available.
Prioritise work that has a clear, measurable target impact.	If an MVI lacks a strong, well-defined Target Impact, consider refining it further before prioritising. If two MVIs have similar Value Scores, prioritise the one with a more impactful, measurable outcome.

Prioritisation is essential for maximising value delivery in software projects. By leveraging **Target Impact, Value Score, and Effort,** teams can **systematically evaluate** the impact vs. effort of each MVI, ensuring that development efforts align with business objectives. Using a structured Value vs. Effort framework, teams can prioritise effectively, increase transparency, and make better strategic trade-offs.

Summary: A Unified Approach for Managing Software Projects

Effective backlog management is critical to the success of software projects. However, traditional Agile practices often struggle to provide a structured way to balance functional priorities with operational efficiency and long-term sustainability. This article introduces the **Minimum Valuable Increment (MVI)** framework, offering a comprehensive approach to redefining value, structuring requirements, and organising backlogs to maximise impact.

By defining value from the perspective of the company producing the software and its sponsors rather than solely from the end user's viewpoint, teams can ensure that every increment contributes meaningfully to the organisation's objectives—improving customer experience, reducing operational risks, or enhancing development efficiency.

Key Concepts and Takeaways

Broadening the Definition of Value
Traditional backlog management overemphasises functional features while neglecting operational and efficiency improvements. MVI expands the definition of value beyond end-user benefits to include:

- **Functional Value** – Delivering features that enhance customer experience.
- **Operational Value** – Ensuring system reliability, scalability, and cost-effectiveness.
- **Efficiency Value** – Optimising the development process to improve delivery speed and reduce technical debt.

By adopting this broader definition of value, teams can make **more informed prioritisation decisions**, ensuring that software remains **impactful and sustainable**.

The MVI Framework: A Structured Approach to Requirements

The MVI template replaces traditional User Stories with a **value-centric format,** ensuring every backlog item (MVI) is **well-defined and measurable**. Instead of focusing solely on user needs, it structures requirements based on tangible business impact.

- **Value** – The business impact the increment will achieve.
- **Increment** – The specific change or functionality being delivered.
- **Verification** – How success will be validated.

Additionally, **Strategic Planning Fields** categorise backlog items by value dimension, domain area, and requirement type to provide better organisational insights. **Prioritisation & Estimation Fields** help teams prioritise work by assessing its expected impact, value, and effort.

A Hierarchical and Holistic Backlog Structure
Instead of managing a flat backlog of loosely defined Epics and Stories, MVI structures work across four levels, clearly linking strategic objectives and execution.

- **Milestones:** Strategic business objectives aligned with the business roadmap.
- **Goals:** Higher-level deliverables that guide development efforts.
- **MVIs:** The smallest units of measurable business value.
- **Tasks:** Execution details that break down MVIs.

This structure ensures that all backlog items contribute to **both short-term goals and long-term business sustainability.**

Enhancing Backlog Organisation with Strategic Planning Fields
The MVI framework improves the organisation of the backlog by categorising requirements using **Strategic Planning Fields.** This categorisation ensures that work is prioritised effectively and aligned with business needs.

- **Value Dimension:** Defines an item's **business impact**, ensuring alignment with a value stream such as customer acquisition, cost reduction, or risk mitigation.
- **Domain Area:** Identifies which **business or technical domain** is impacted, improving cross-team collaboration and dependency management.
- **Requirement Type:** Classifies work as **Functional, Operational, or Efficiency** to balance feature development, system health, and engineering productivity.

By structuring backlog items with these fields, teams gain **better visibility, improved strategic alignment, and deeper insights** into how work is distributed across different priorities.

Strategic Prioritisation Using Impact-Driven Metrics
Prioritisation must go beyond gut feelings or arbitrary urgency. The MVI framework introduces **structured prioritisation methods** to help teams focus on **high-value, high-impact work**.

- **Target Impact:** A measurable business outcome.
- **Value Score:** A numerical ranking of business importance.
- **Effort Estimate:** The effort required for implementation.

By combining these factors, teams can **balance quick wins with strategic investments**, ensuring that high-impact and high-effort initiatives receive appropriate attention.

Why This Approach Works (Benefits of the MVI Framework)

Adopting the **MVI framework** transforms backlog management by bringing clarity, strategic alignment, and measurable outcomes. Key benefits include:

- **Business-Driven Decision-Making:** Ensures that every increment delivers measurable value, moving beyond feature-focused backlogs.
- **Unified Backlog Across Requirement Types:** Prevents technical debt accumulation by integrating functional, operational, and efficiency requirements into a single backlog.
- **Better Alignment with Business Strategy:** Facilitates prioritisation based on business goals, enabling leadership to track and adjust investments effectively.
- **Scalability and Adaptability:** Supports organisations of all sizes by enabling top-down strategic planning and bottom-up backlog refinement.
- **Improved Developer Efficiency and Morale:** Reduces frustration caused by neglected technical improvements and ensures engineers work on impactful initiatives.

Call to Action: Transform Your Backlog for Strategic Impact

Whether you're a CTO, VP of Engineering, Product Owner, or Agile Coach, implementing MVI can help you bridge the gap between business objectives and execution. Empower your teams to deliver software that drives meaningful, long-term business impact. Start today by restructuring your backlog, refining prioritisation methods, and aligning your teams around measurable business outcomes.

MVI FAQ and Adoption Guidelines

1. Agile Mindset and the MVI Approach

Q1. How does the MVI approach align with the Agile mindset?

The MVI approach is fully aligned with the Agile mindset, which emphasizes short feedback loops, continuous inspection, and adaptation. The backlog organisation proposed follows the same principle: it can be reviewed and adjusted at any time, and only prioritised backlog items need to be refined with full implementation details. Anything else can remain high-level until it becomes relevant.

The key difference between MVI and traditional Agile backlog management is how backlog items are structured. Instead of Epics and User Stories, MVI organises backlog items into Milestones, Goals, and MVIs, all written using the MVI template. This ensures that every backlog item explicitly states its business value while maintaining Agile flexibility.

Q2. How does the MVI approach differ from traditional Agile backlog management?

The MVI approach does not change the Agile backlog management principles but enhances how backlog items are written and grouped. Traditional Agile backlogs use Epics and User Stories, whereas MVI replaces them with Milestones, Goals, and MVIs, which follow a structured format emphasizing measurable business value.

Furthermore, only backlog items that are prioritised for implementation require full refinement. Items lower in the backlog can remain loosely defined, just as in traditional Agile practices. Regular backlog refinement sessions ensure that priorities are continuously reassessed and adjusted based on new insights.

2. Compatibility with Agile Practices and Frameworks

Q1. How does the MVI approach accommodate clients who prefer traditional User Story templates?

The MVI approach retains the flexibility to accommodate teams and clients familiar with the traditional User Story format. Functional MVIs can still include user personas in their value statements if necessary, ensuring continuity with existing Agile practices.

For example:
Traditional User Story: "As a user, I want to reset my password so that I can regain access to my account."

Equivalent MVI:

- **Value:** Improve customer retention and reduce support requests by enabling users to securely reset their passwords.
- **Increment:** Implement a password reset feature with secure email verification.

While the user's needs are still captured, the MVI approach explicitly ties the requirement to measurable business value, ensuring that the feature is justified from an organisational perspective.

Q2. Can MVI integrate with Agile frameworks such as TDD, BDD, and DDD?

Yes, the MVI approach integrates well with established Agile development frameworks:

- **TDD (Test-Driven Development):** MVIs provide a clear definition of what should be built, guiding the creation of acceptance, integration, and unit tests before implementation.
- **BDD (Behaviour-Driven Development):** BDD scenarios can be documented in the *Examples* or *Verification* fields of the MVI template to clarify expected system behaviour.
- **DDD (Domain-Driven Design):** MVIs align well with DDD principles by ensuring backlog items correspond to specific bounded contexts, reinforcing proper domain modelling and backlog planning.

By structuring backlog items as MVIs, teams improve traceability and clarity while maintaining compatibility with best practices.

3. Team Organisation and Collaboration

Q1. How does MVI impact team structure, particularly in DDD-aligned organisations?

The MVI approach does not dictate how teams should be organised. Agile organisations typically structure teams around value streams or domain areas—or a mix of both. MVI supports both models and allows backlog organisation to be adjusted accordingly.

The MVI template includes optional planning and tracking fields (*Value Dimension* and *Domain Area*), which enable organisations to filter and structure their backlogs based on their team organisation. Teams can maintain either a unified backlog or separate backlogs by domain, depending on their approach.

Q2. How does MVI facilitate cross-team collaboration?

MVI enhances collaboration by structuring backlog items to ensure alignment across teams while allowing for autonomy in execution. Shared milestones provide clear high-level objectives that multiple teams can contribute toward, even if they are working on different goals and MVIs within those milestones.

This approach prevents siloed development and ensures that all teams remain aligned with the overarching business strategy.

Q3. Can MVI be used with frameworks such as SAFe and Team Topologies?

Yes, the MVI approach can be used with any team organisational structure. Whether an organisation prefers to group teams into **Agile Release Trains (ARTs)** as in **SAFe** or differentiate between **Stream-Aligned and Platform Teams** as in **Team Topologies**, the MVI approach remains applicable and flexible.

For large organisations with multiple teams and initiatives, Milestones can be grouped into **Initiatives**. These Initiatives may represent different business areas, products, or value streams, aligning closely with Portfolio Management and PMO structures. Different Initiatives (and their respective teams) could define their own Milestones. This approach is similar to SAFe's hierarchical backlog structure, which includes **Portfolio Backlog, Solution Backlog, Program Backlog, and Team Backlog**. Items within these backlogs could be written using the MVI template to ensure consistency in value-driven delivery.

For **Stream-Aligned and Platform Teams** in Team Topologies, it's important to recognise that every team will likely work with all three types of MVI requirements—**Functional, Operational, and Efficiency**. However, the distribution of work may vary:

- **Stream-aligned teams** may focus more on Functional Requirements while still handling Operational and Efficiency requirements to ensure they support and improve the systems they build.
- **Platform teams** may emphasise Operational and Efficiency requirements but should also treat the tools and automation they provide to Stream-Aligned Teams as Functional Requirements, considering those teams as their direct clients.

Separating teams entirely by requirement types is a mistake. Stream-aligned teams are expected to support what they produce and optimise their delivery processes, meaning they also need Operational and Efficiency requirements in their backlog. Likewise, Platform Teams need to enable Stream-Aligned Teams to better support their systems and improve efficiency, meaning their tools and automation efforts should be treated as Functional Requirements targeted toward Stream-Aligned Teams.

In both SAFe and Team Topologies structures, the MVI approach provides a structured yet adaptable way to manage requirements and align teams with business value across different organisational setups.

4. Balancing Agility and Predictability

Q1. How does MVI balance Agile flexibility with the need for predictability?

MVI maintains Agile principles while adding structure to backlog management. Like traditional Agile backlogs, only high-priority items are fully refined, while lower-priority items remain loosely defined.

The backlog follows the same backlog refinement ceremonies as traditional Agile teams. The current milestone and its prioritised goals should be well-defined, whereas future milestones and lower-priority goals remain flexible. This ensures alignment with strategic objectives while allowing teams to adapt to new insights and evolving business needs.

Q2. How is effort measured in MVI, and how can it be used for predictability?

MVI does not prescribe a specific effort estimation technique, but teams can use story points, time-based estimates, or other methods as needed. The important thing is to maintain consistency in whichever approach is chosen.

While MVI includes an optional *Effort* field, it does not enforce a particular unit of measurement. However, adding clear *Verification* instructions and breaking the MVI into smaller tasks will help teams make more precise estimates. Doing so is highly recommended, as it provides better predictability and enhances planning accuracy.

If the *Effort* field is used consistently across all MVIs, it will be easier to derive estimates for Goals and Milestones, allowing leadership and senior stakeholders to gain a clearer understanding of the overall timelines and progress when reviewing higher-level backlog items.

5. Managing Change and Scaling

Q1. How does MVI support change management when milestones or goals need to pivot?

The MVI approach embraces change as a fundamental Agile principle. Since backlog items are structured around measurable business value, priorities, goals, and milestones can be adjusted efficiently. Regular backlog refinement ensures that pivots occur deliberately and with minimal disruption, maintaining alignment with business strategy. The mandatory Value field in the template provides a great way to discuss business priorities and adjust the backlog accordingly.

Q2. How does MVI support scaling and portfolio management?

The hierarchical structure of Milestones, Goals, and MVIs provides a scalable framework for managing large portfolios. Leadership can visualize progress across different initiatives and adjust priorities dynamically.

By tagging MVIs with *Value Dimension and Domain Area*, organisations gain insights into effort distribution across functional, operational, and efficiency-focused workstreams. This ensures balanced investment and alignment with strategic goals, even as multiple teams contribute to shared milestones.

Additionally, MVI's structured backlog format supports dependency management, cross-team coordination, and continuous adaptation as organisations scale.