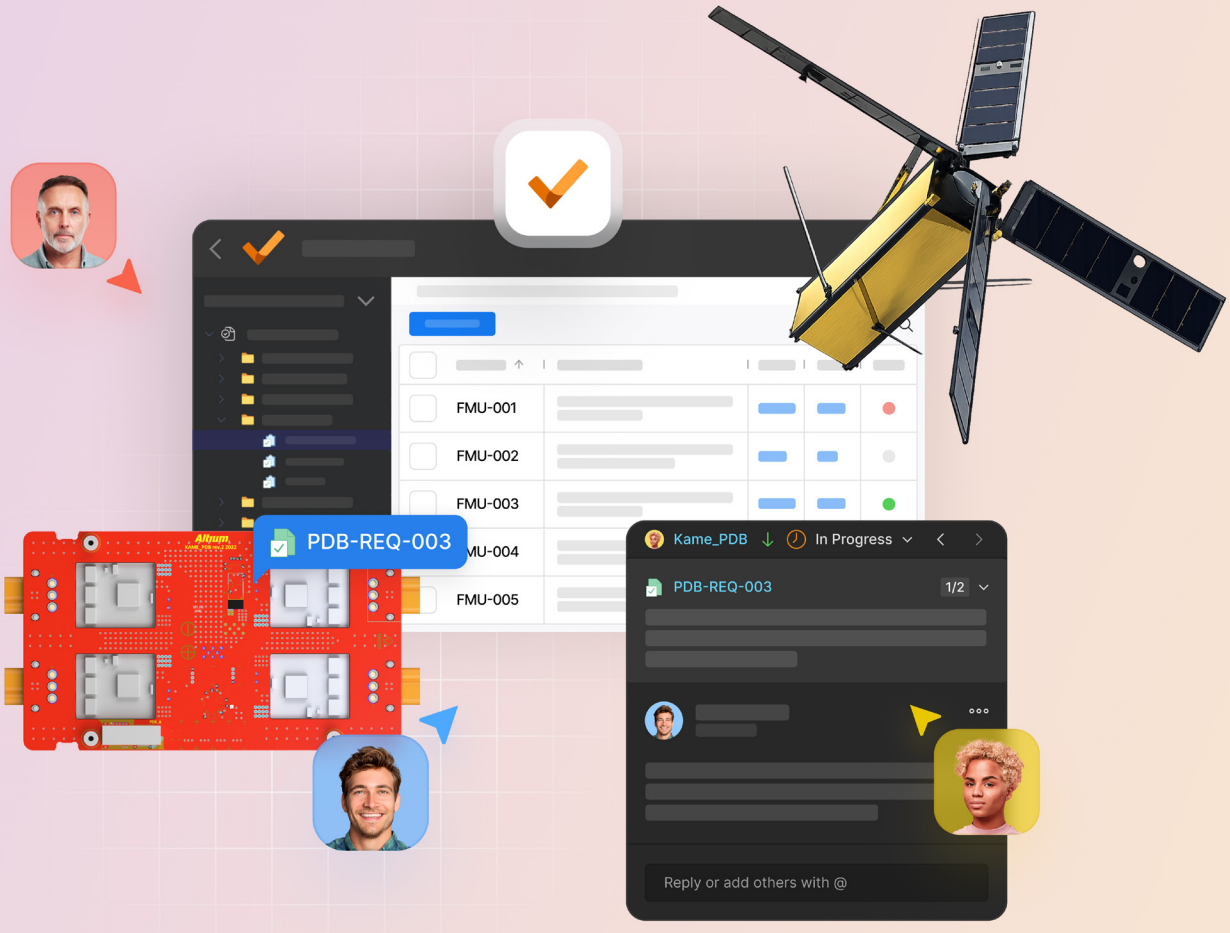
Altium

Lean Requirements For Space Engineering

How to Iterate Faster Without Breaking Traceability

Practical Requirements, Verification,
And Collaboration Workflows For Modern
Hardware Engineering Teams



Contents

Who Should Read This Guide?

1. NewSpace Engineering Leads

You're building a NewSpace constellation with a lean engineering team facing brutal market economics. Learn how requirements management prevents the cascading failures that kills 1 out of 3 programs and enables small teams to reach orbit in months.

2. Leaders in Traditional Aerospace

You're a lead at a traditional aerospace company stuck with legacy processes while watching NewSpace competitors move faster. This guide provides practical strategies to modernize requirements management within your existing governance structure.

3. Cross-Industry Technical Leads

You're a technical leader outside the space industry looking for transferable lessons. Learn how space teams manage complexity, traceability, and verification under extreme schedule and reliability constraints — and how to adapt these practices to your own industry.

Introduction

Market forces are reshaping the landscape in the space industry:

- CubeSats now dominate at 39% market share, with 6U form factor becoming most common [1]
- Paradigm shift from single satellite ops to managing constellations of hundreds or thousands [2]
- Customers expect near-real-time data delivery expectations in minutes to hours vs. days [2]
- COTS satellite components lowers barriers to entry, enabling NewSpace startup formation [3]
- At the same time, only 5% of announced programs are actually launched — with 29% cancelled or dormant and most delayed by 6-12+ months [1]

These market trends change how teams build space systems. Engineering organizations are adapting to compressed timelines:

- Small teams (9-10 people) now deliver full mission stacks in 18 months, replacing multi-year programs with one-year development cycles [4]
- Applications drive design with hub-and-spoke models enabling local payload development with centralized integration, distributing specialized work across geographies [5]
- Software-defined architectures replace hardware complexity and lead to mass savings [4], while onboard AI/ML computing shifts computational workload from ground stations to orbit [2]

Software engineering has been disrupted by AI. Now, hardware teams face similar expectations for delivery speed.

To hit their targets hardware engineers need tools that help them manage complexity, communicate current information, and collaborate across disciplines with confidence.

Yet the tools they use remain largely unchanged.

- Most still rely on Excel. Spreadsheets are “good enough” for prototyping, but they turn into a nightmare. As complexity increases, traceability, version control, and distribution of current information to all stakeholders become essential, but are near-impossible to manually maintain at scale.
- Some invest in enterprise requirements management systems and MBSE/SysML tools. However, DOORS and the similar “industry-standard” tools are excellent for storing requirements as a system of record but were designed for long programs and waterfall handoffs. MBSE/SysML tools have powerful modeling capabilities but ultimately lead to organizational silos, obstructing communication and slowing impact.
- Others borrow tools designed for software teams. Yet bits are not like atoms. Software can be partitioned, developed, and tested in isolation, while hardware subsystems are inherently interdependent. Project management tools built for software struggle to represent hardware dependencies and lifecycle constraints.
- A few build internal tools. However, unless you work at SpaceX with its Warp Drive ERP, this approach is rarely viable. Such investments break focus and compete with mission delivery.

> There is a better way to build complex space systems. In NewSpace, engineers can’t afford the traditional 6-month requirements review cycles. They use “lean requirements” — write fast, iterate with prototypes, validate with demos. [13]

Over the past decade, our team has worked with companies to optimize their product development processes and tech stack. In this guide, we distilled best practices used by space teams today to manage requirements and communicate critical engineering information.

You will learn what is a proven path to work more iteratively with requirements while still fulfilling the need for traceability and structure.

Why Care About Requirements?

Ask engineers from different disciplines what requirements are, and you will get vastly different answers:

- Systems engineers think of “shall” statements and INCOSE best practices.
- Hardware engineers think of technical parameters and specification sheets.
- Software engineers think of user stories and PRDs.
- Verification engineers think about standards, test cases, and certification.
- Manufacturing engineers think of production constraints and quality assurance.
- Engineering contractors think about SOWs and scope creep.

Simply put, each discipline views requirements through its own lens and ways of working. Together, these perspectives explain why requirements sit at the center of all engineering activities in new product development.

Requirements as Mini-Contracts

Each requirement acts as a bidirectional contract between the company, its users, customers, certification authorities, and the individual engineers involved in the development process. Requirements ensure that what is designed and delivered meets engineering intent, performance targets, and safety expectations.

Practical example: A requirement becomes a contractual obligation that protects the customer if a test fails — and the company if the customer changes the scope.

Key function: Preventing disputes, establishing accountability, proving regulatory compliance

Requirements as Collaboration Tools

In practice, requirements are how engineers communicate technical information. They help ensure that parts work together as a whole — both at the interfaces of systems and subsystems and across engineers of different disciplines.

Practical example: A functional test drives concrete design choice, such as the addition of a dedicated monitoring pin at a specific location on a PCB.

Key function: Coordinating cross-functional teams, managing supplier handoffs, ensuring integration.

Requirements as Problem Definition Guardrails

The content of requirements and their interconnections define the problem space, constrain the solution space, and guide design decisions throughout development. In other words, requirements define the problem so that engineers can build the best possible solutions.

Practical example: Well-defined requirements help engineers make informed tradeoffs when balancing, for example, power consumption and data transfer constraints.

Key function: Making design tradeoffs, managing scope evolution

Requirements as Meaningful Work

At a high level, requirements align teams around a shared goal. Defining requirements demands critical thinking and engineering judgment. In the age of AI, requirements must remain a core responsibility of the engineer — supported, but not replaced, by technology.

Practical example: Flow down of mission requirements to system and subsystem requirements is a collaborative and creative process

Key function: Aligning fast-moving teams, preserving institutional knowledge

> In the age of AI, requirements must remain a core responsibility of the engineer — supported, but not replaced, by technology.

Which Practices Improve Engineering Collaboration?

When requirements sit at the center of development, collaboration shifts from a cultural nice-to-have to a direct indicator of successful delivery. The following research-backed practices help interdisciplinary teams maintain alignment as requirements evolve, reduce rework, and support faster decision-making.

Establish Shared Understanding and Clear Objectives

Teams need a common view of what is being built and why. As requirements flow from system definition to design and verification, they facilitate communication of clear goals, shared terminology, and early agreement on problem framing, reducing misalignment. [6]

Build Trust to Open Information Sharing

Effective teams share progress, risks, and issues openly. Trust enables engineers to surface requirement gaps, raise concerns, and flag downstream impacts before they become late-stage problems that cause rework. [7]

Facilitate Cross-disciplinary Communication Norms and Rituals

Regular structured communication, such as stand-ups and requirements or design reviews, create predictable points for alignment and reduce silos. These sessions help teams assess requirement changes, review cross-disciplinary interfaces, and confirm verification plans. [6]

Support Distributed and Remote Collaboration with Appropriate Tools

Modern space programs rely on distributed teams. [3] Collaboration tools must support asynchronous work, version control, and shared access to requirements, designs, and verification artifacts without relying on formal handoffs. [8]

Proactively Cultivate Interdisciplinary Team Skills

Engineers work most effectively when they understand adjacent domains. [9] Common practices include reasoning in terms of constraints rather than functions and taking ownership of interfaces rather than isolated components. This mindset improves requirement quality and reduces integration risk. [10]

How to Avoid Common Pitfalls According to Practitioners?

Requirements management in NewSpace programs faces unique pressures: rapid development cycles, constrained budgets, and high-volume production collide with traditional aerospace rigor. We asked 50 practitioners to reveal where requirements processes break down and how teams are adapting.

Ambiguous Requirements

Vague terms like «high reliability» or «low power» cause interpretation conflicts leading to 3x rework cost.

How to Avoid: Define quantitative requirements and acceptance criteria: specify values, operating conditions, and test methods. When customers cannot provide details, document assumptions and track them as program risks.

> **Bad Example:** "The sensor shall be low power."
>
> **Good Example:** "The sensor shall consume 1 W while active and 0.01 W in standby."

Traceability Breakdown

Requirements diverge from design models, ECAD and MCAD implementations, simulation and verification artifacts, creating disconnected silos.

How to avoid: Link requirements to systems, designs, code, and tests. Use a single platform instead of maintaining parallel systems. Automate traceability matrices to reduce manual effort by 70%.

Requirements Misinterpretation

Engineers working on subsystems interpret parent requirements differently, causing 80% of rework, interface mismatches, and implementation errors.

How to Avoid: Conduct cross-functional flow-down reviews before subsystem design starts. Automate traceability across requirement levels to prevent manual tracking errors.

Requirements "Gold-Plating"

Requirements "gold-plating", in other words the practice of specifying as must-have requirements features that are nice-to-have and not map to customer needs, inflate development costs without adding value.

How to Avoid: Trace every requirement addition against mission objectives to avoid requirements bloat. Gate trade studies to prevent every "what-if" scenario from becoming a requirement.

Verification Cost Skyrocketing (due to Overspecification)

Over-compliance to standards (for example, applying Class B requirements on Class C missions) can consume 40% of total budget making it unfeasible for high-volume constellation programs. [11]

How to Avoid: Apply risk-based verification: test critical safety functions, use analysis for lower-risk items. For constellations, establish platform baseline verification once, then verify only mission-specific deltas.

Disconnect from Real-World Manufacturing

Requirements written without input from the domain experts often ignore constraints around tolerances, assembly processes, and supplier capabilities, causing multi-month re-baselining delays.

How to Avoid: Involve implementation engineers in requirements early. Embed DFM rules in design tools for automated compliance checking. Include design-for-assembly requirements.

“Zombie” Requirements in Tests

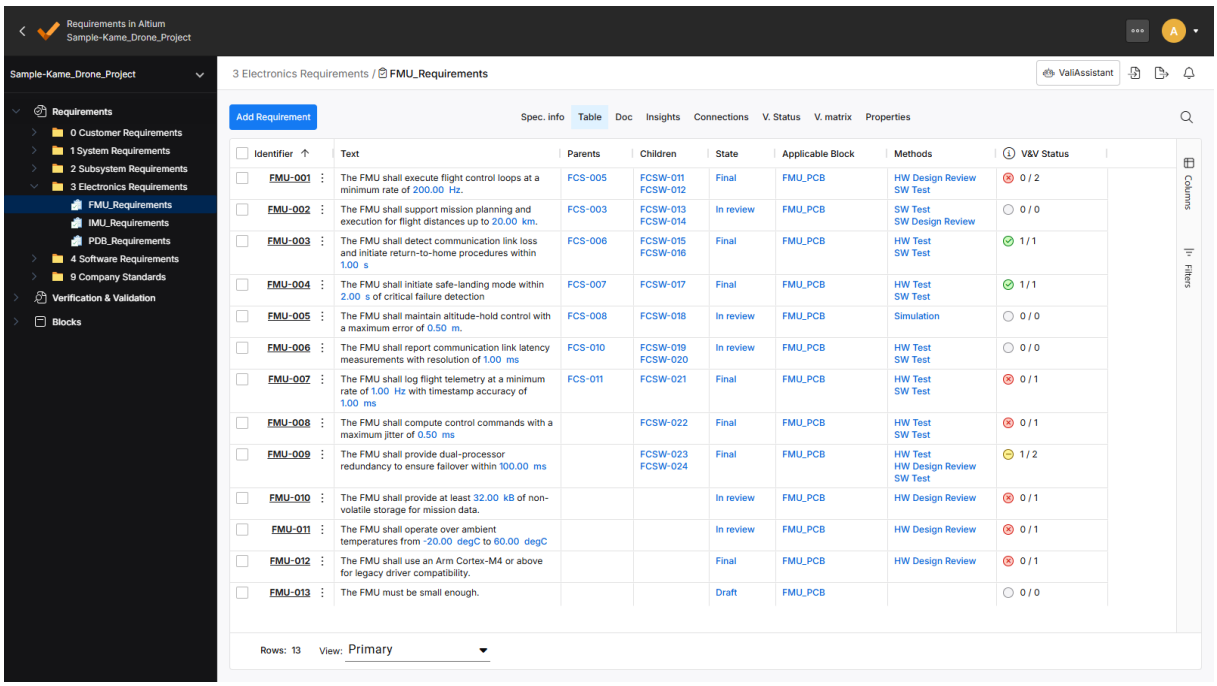
Mid-cycle customer changes and weak baseline and change control leave behind “zombie” requirements in test cases that are no longer relevant.

How to Avoid: Trace requirements to verification activities and test cases. Flag verification activities without linked requirements. Perform a change impact analysis to check for linked test cases.

Confusing Verification for Validation

Teams verify specifications but forget to validate that the verified system solves the customer’s actual problem.

How to Avoid: Separate verification (built it right) from validation (built the right thing) with distinct test cases. Conduct customer demos for validation, not just checklist verification.



Avoid common pitfalls by centralizing requirements, verification, and testing in a single workspace accessible to your whole team — without burdening your engineers with unnecessary processes.

How to Pick the Right Requirements Tool?

How do you put theory into practice? What tools and processes do engineering teams need to manage requirements and share engineering information effectively? Here are five key areas to consider when changing your requirements management tool within your engineering tool stack.

Requirements Traceability Chains

Tools shall bidirectionally link requirements to (useful) engineering artifacts across product development — to other requirements, system and subsystem blocks, ECAD and MCAD designs, simulations, verification activities, and test runs. This ensures consistency of information, enabling collaboration and faster iterations.

Reusable Parameters and Calculation Engine

Requirements are more than text. Engineers think and work in numbers. Turning numerical values in requirements into parameters enables consistent reuse across a project, helps teams understand downstream impact when values change, and enables automated verification rules that compare system performance to requirements.

Verification Planning and Test Management

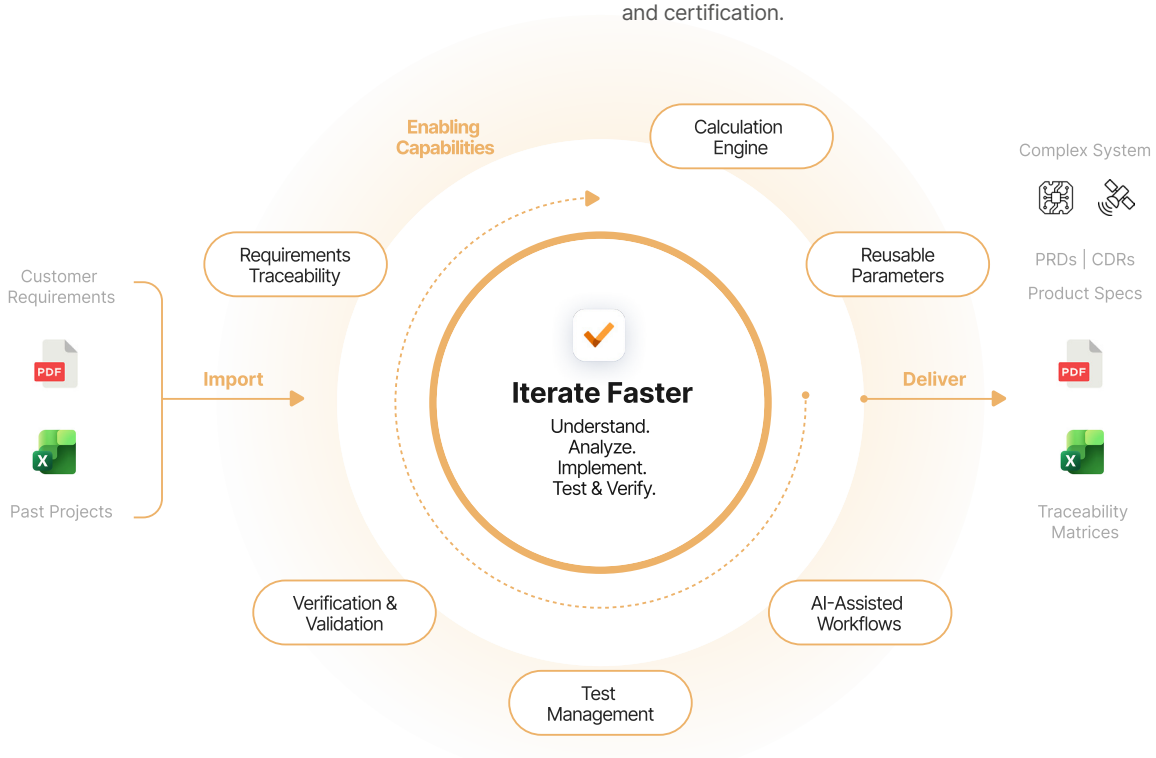
Requirements are only as good as the tests that validate them. A requirements management tool shall support verification planning, execution, and documentation, while linking each test case back to its requirement and capturing related evidence.

AI-Assisted Workflows

Requirements tools should use AI where it makes sense. Current AI models are strong at processing text making them excellent tools for applications such as extracting data from datasheets, providing recommendations for requirements decompositions, or finding inconsistencies.

Flexible Import/Export

Simply put, requirements tools must offer flexible solutions to import and export data. This doesn't necessarily mean complex integrations and APIs, but rather an easy way to bring in requirements in formats provided by your customers, start iterating right away, and later export documentation in any format needed for project hand off and certification.



Iterate faster with Requirements Portal — Import requirements. Connect them to verification and designs. Export documents in the formats you need. Deliver complex systems.

Requirements Traceability

A practical traceability chain should be easy to traverse end-to-end and bidirectionally. For example, from requirement ↔ system ↔ ECAD design ↔ verification activity ↔ test case with attached evidence.

Forward traceability ensures that the product is designed and built as initially intended. Backward traceability links evidence back to the source requirement for reviews, troubleshooting, and audits.

> **Traceability has a cost:** In traditional workflows, engineers spend 5–10 minutes per requirement creating and maintaining links. [12] Scale that across tens or hundreds of thousands of requirements, and traceability becomes a major drain on engineering time.

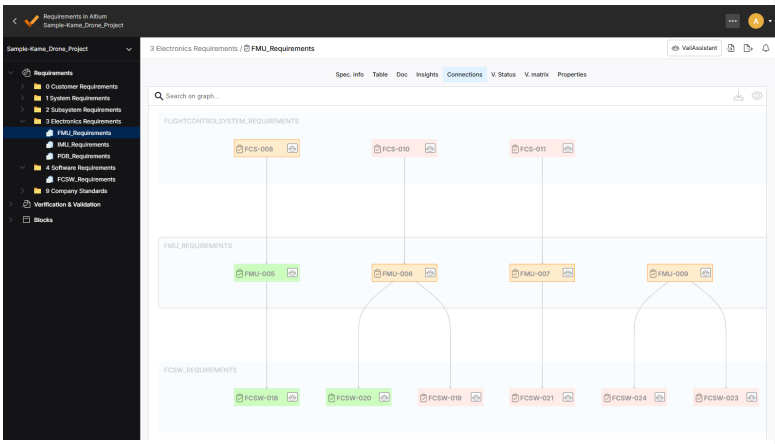
That is why many teams treat traceability as a burden. A modern requirements management tool must reduce this burden by making links easy to create, review, maintain, and use.

Between Requirement

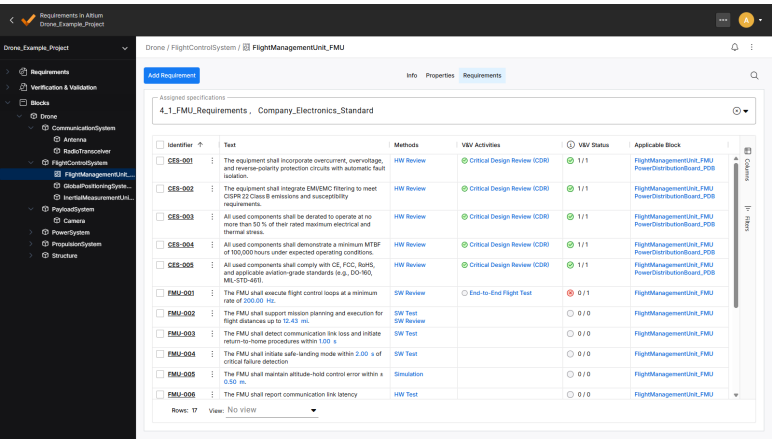
As requirements flow from system to subsystems to components, you need clear parent–child relationships. Engineers use such hierarchical structures to answer basic questions, like: Where did this requirement come from? What requirements does it drive downstream? What's the impact if it changes?

Not every requirement has a clear parent. Some are derived from an interface definition, a physical constraint, a safety target, or a design trade study.

A requirements management tool shall support both these use cases with visual aids. For example, connection graphs help engineers validate structure fast. They expose orphaned requirements, weak justification, and missing links. They also make change reviews more reliable.



Hierarchical Requirements: Connection graph showing hierarchical parent–child relationships between requirements, color-coded by verification status.



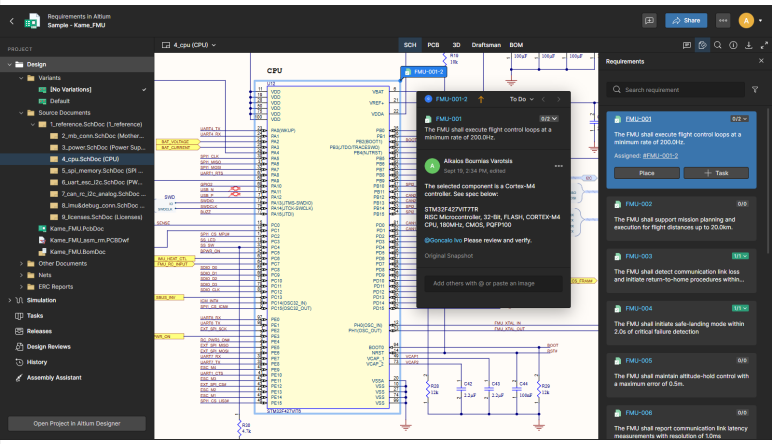
Traceability to System Blocks: List of requirements allocated to a subsystem, with their corresponding verification activities.

Between Requirements & Systems

It is common practice to allocate requirements to systems and subsystems – in other words, mapping the problem space to the solution space – to clarify ownership. This helps, but does not prevent a common failure mode: the same requirement being interpreted differently by different people at different levels.

The tool should make the requirement-to-system link explicit and easy to navigate. Any engineer should open a requirement and immediately see details about the subsystem it applies to, its owner, and relevant interfaces with other subsystems.

This context supports faster alignment and better decisions, without the need to re-iterate intent in meetings.



ECAD Integration: Requirements shown next to a PCB schematic (right), and a requirement linked to a component for traceability (left).

Between Requirements & Designs

Linking requirements to their implementation in ECAD, MCAD, or BOMs, matters both during development and during verification and audits:

- During design, the implementation engineer should see linked requirements in context and receive notifications when a requirement changes so they can start rework immediately.
- During verification and audits, engineers should be able to trace from a requirement to its physical realization — for example, a specific IC, constraint, or region — for quicker troubleshooting and proof of compliance.

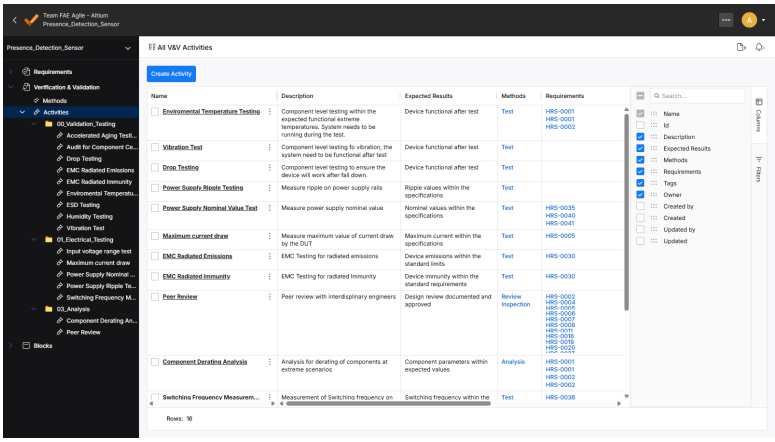
Direct ECAD or MCAD integration reduces manual cross-referencing, tool-switching, and errors caused by outdated requirements.

Between Requirements & Verification

Linking each requirement to a V&V activity, ensures full coverage. It also keeps tests aligned with the current spec, preventing “zombie” requirements (i.e. running test cases that still reference deleted or changed requirements).

At minimum, the tool should capture the verification method per requirement (analysis, simulation, inspection, test, etc.) and tie it to the requirement’s risk.

It should then show, by spec or subsystem, which requirements are unverified and which have no test planned. This enables teams to close gaps early, not during final qualification.



Verification Management: List of all verification and validation activities of a project, traced to requirements.

Reusable Parameters & Calculation Engine

Requirements are more than text. Engineers think and work in numbers, and requirements very often contain extremely important numeric values. A common example is running trade studies across alternative solutions and comparing their performance against requirement targets.

A requirements management tool that treats numeric values as parameters that can be referenced throughout the project — from requirements to systems to verification activities — adds value beyond just serving as a "system of record" for compliance.

Example Workflow

Imagine a simple but representative engineering scenario:

1. Start with the basic functions your system must fulfill.

- You note your targets—for example, a power consumption target.
- You create a quick diagram of the system and its components.

2. Bring it into your requirements tool.

- You drop your Excel or Word document into the tool and import preliminary requirements using the AI assistant.
- You create a power-consumption requirement. Instead of a text-only "shall" statement, it includes a parameter that defines the target.
- You create blocks for the system and its components. Each component block includes a power consumption parameter. The system's power consumption is calculated using a formula that takes into account losses.

3. Start designing and prototyping the solution.

- Based on datasheets, you allocate power budgets and run "what-if" analyses to evaluate alternative solutions.
- At the same time, an automated rule continuously checks current power consumption against the requirement.
- Once the results are acceptable, you implement the design. Requirements are visible next to the design in your ECAD or MCAD tool.

4. After implementation, move to testing.

- When writing the test procedure, you can directly reference the parameters you used in the system design and requirement definition.

5. Adapt to change when it (inevitably) happens.

- Because parameters are reused, you just have to change one value once and the update propagates across the whole project, retriggering verification rules and updating documentation automatically.

Now imagine this repeated for every functional requirement in your system. How much time would you save by avoiding scattered data and manual document updates?

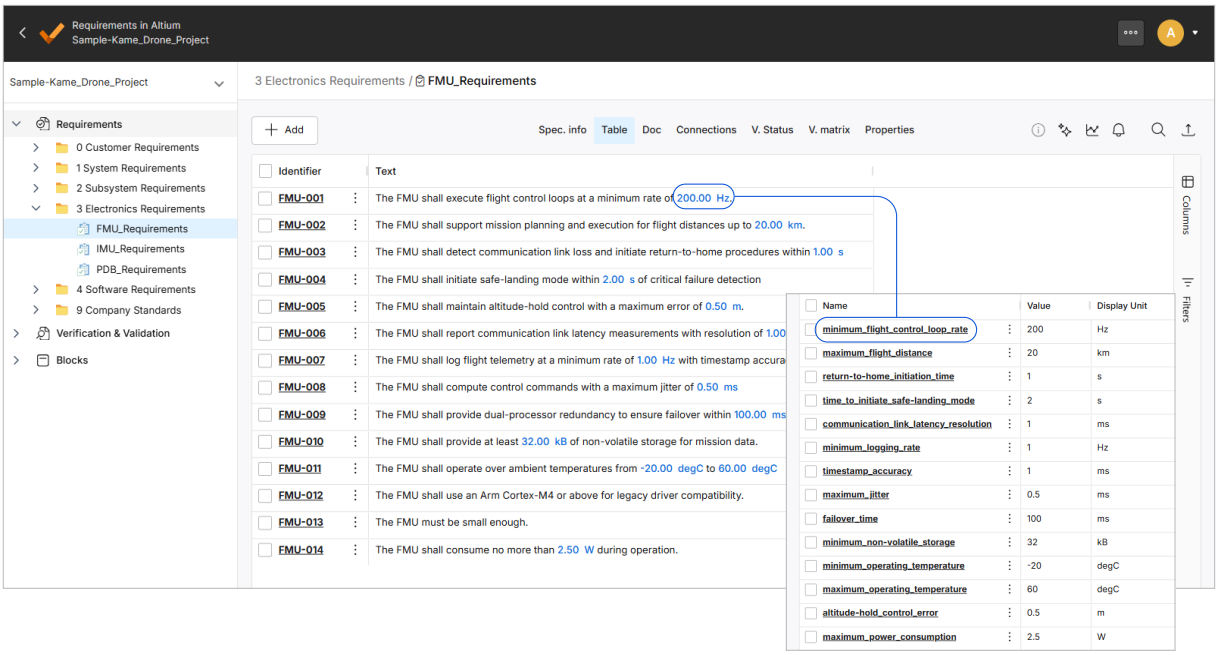
Benefits of Reusable Parameters in Requirements

Implicit Traceability

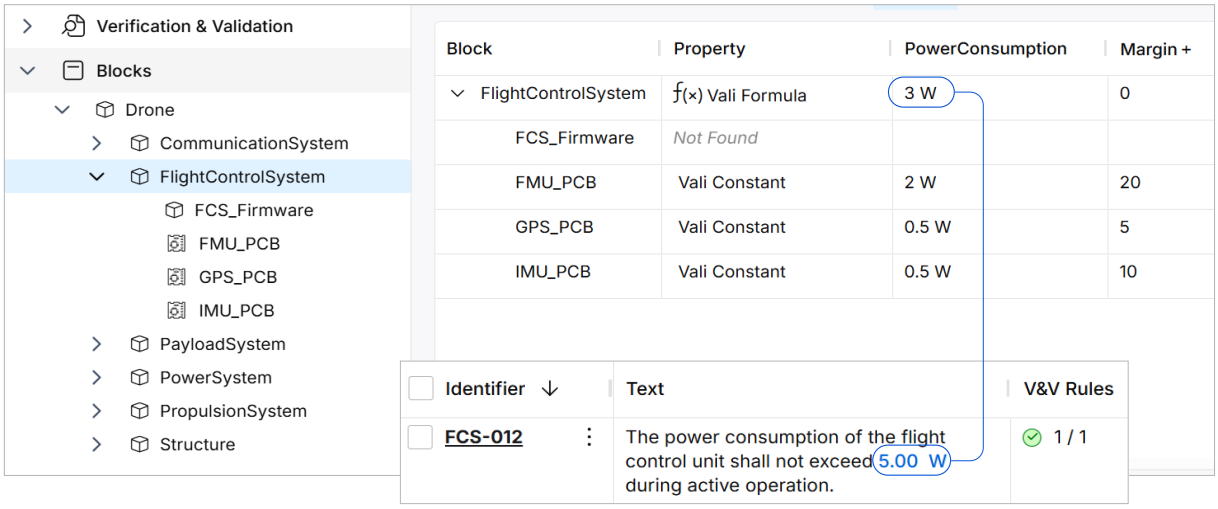
Change it once and it automatically updates throughout your project, ensuring consistency of information, avoiding errors, and saving engineers' time.

Continuous Verification

Set up V&V rules that continuously check system performance against requirements, streamlining tradeoff studies and giving you a complete overview of the system.



Parameters in requirements: Blue numeric values in the requirement text represent reusable parameters.



V&V Rules: The calculated power consumption of the Flight Control System is compared to its corresponding requirement via a V&V rule.

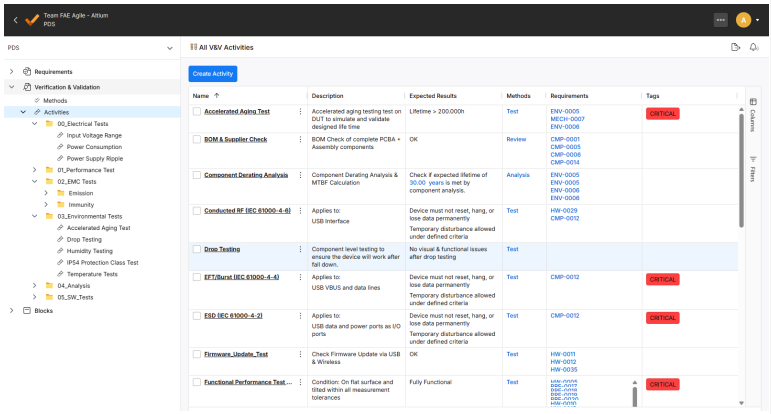
Verification Planning & Test Management

Since performing a full qualification on hundreds of satellites of a constellation would blow most program budgets, NewSpace teams are moving towards risk-based verification. In other words, they test what matters, and analyze the rest.

For this use case, a requirements management tool shall provide enough structure to guide the development of verification and validation plans and test cases, without getting in the way of this lean and iterative way of qualifying complex hardware.

Step 1: Verification & Validation Planning

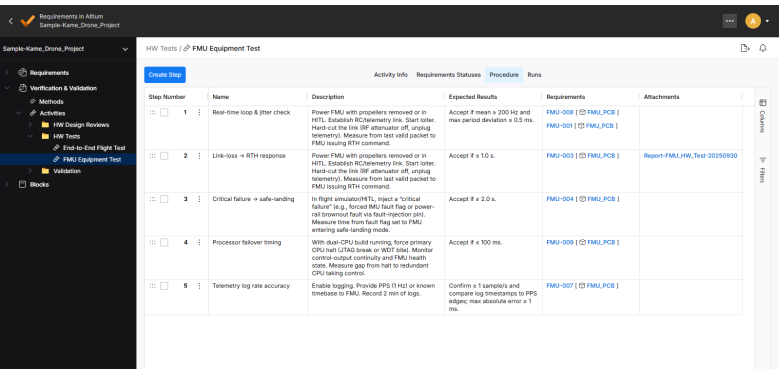
First, identify the requirements with the highest risk, in terms of mission criticality, functional safety, interface sensitivity, and cost of failure. Then assign a verification method to each requirement: testing for critical requirements, analysis, simulation, or review for the rest.



List of planned verification activities and their criticality.

Step 2: Create Verification Activities & Test Cases

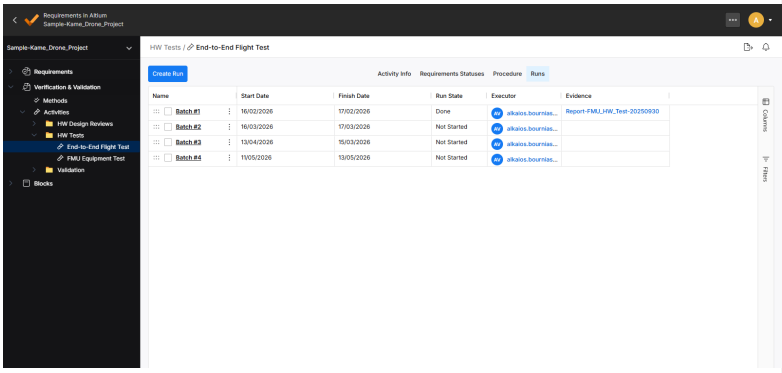
Once the plan is reviewed and confirmed by stakeholders, then create verification activities, including expected results and test case procedures. By linking each verification activity to requirements, you can ensure full coverage and traceability.



Test case procedure steps, with each step linked to the requirement it verifies.

Step 3: Run Tests & Document Results

Then comes the hard part: running the test and evaluating the results. A requirements management tool shall streamline this step as much as possible by enabling the engineers to plan and track verification runs, store evidence of executed runs, and automatically update the pass / fail verification status across the whole project.



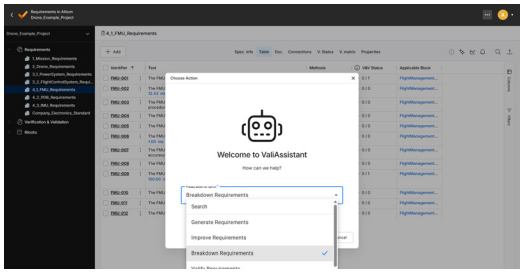
List of planned test case runs with date, owner, and attached evidence.

AI-Assisted Workflows That Add Value

AI is powerful. Large language models (LLMs) are strong at analyzing, processing, and generating text. When configured correctly, AI systems can produce repeatable and accurate outputs.

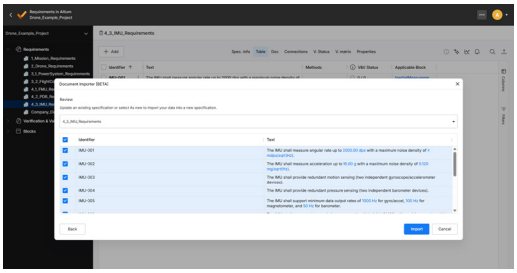
> Requirements are a structured, text-based representation of a system, which makes them a good candidate for AI applications.

After the initial hype, it is clear that some AI use cases are more useful than others. The examples below add value to engineering workflows.



Requirements Suggestions

It is not uncommon for engineers working on complex systems to miss important requirements. An AI assistant can suggest derived requirements to help ensure important requirements are not missed.



Importing from Documents

The reality today is that many teams still use PDFs and Word documents to share requirements. AI tools can quickly parse such unstructured documents, identify requirements, and convert them to structured data.

Other useful applications of AI in requirements engineering include:

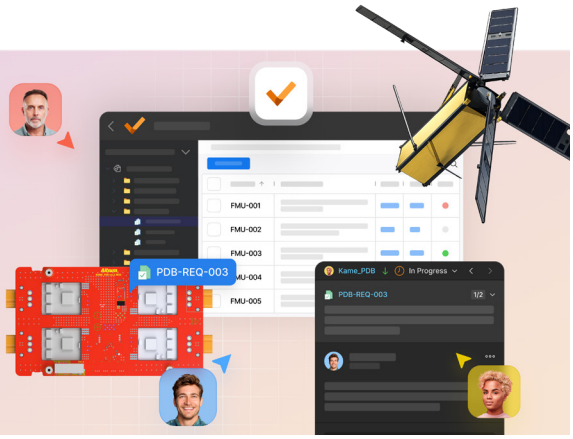
- Identification of duplicate requirements
- Automated requirements review and recommendations
- Consistency checks (for example, between customer and system requirements)
- Automatic assignment of requirements to verification methods
- Automatic generation of test cases and procedures

Altium

Requirements Portal

Iterate faster with a requirements management tool your whole team can access.

Learn More | Take Guided Tour



Requirements Management Tool for Engineering Teams

Stop Chasing Scattered Requirements

Ditch spreadsheets and switch to a modern requirements management tool for engineering teams.

Link Requirements to Designs & Verification

Connect requirements to systems, designs, and test cases for end-to-end traceability and faster iterations.

Manage Requirements Without Limitations

Create unlimited requirements and invite unlimited collaborators — included in all Altium subscriptions.

Key Features

- Requirements Management
- Verification & Validation
- AI Requirements Assistant
- Native ECAD Integration
- Traceability Graphs
- Change History
- Parametric Requirements
- Continuous Verifications
- Dashboards & Analytics
- Smart Requirements Importer
- Templated Document Exporter
- Comments & Tasks

> Requirements Portal has been an instrumental tool, allowing us to gather all-level requirements in one database and ensure full traceability throughout the project's life. Its ease of use, flexibility and scalability are features we really enjoy.

Faviola Kettig
Systems Engineer at ISISPACE

Comparison to Other Requirements Solutions

	REQUIREMENTS CAPTURE	DESIGN & IMPLEMENTATION	VERIFICATION & VALIDATION
Requirements Portal For engineering teams that need to iterate fast while maintaining traceability	+ Built for inter-disciplinary hardware teams + Requirements as the focus of the iterative engineering process + Supports hierarchical and parametric requirements + Balances speed with the structure needed to scale	+ User friendly and quick to onboard by non-specialists + Engineers see requirements in full context + Connects requirements to systems, designs, and verification + Change impact is explicit, enabling faster, safer iterations	+ Treats verification as a core activity + Links requirements to verification methods, test cases, and evidence. + Supports risk-based V&V without forcing + Generates audit-ready outputs from live project data.
Documents & Spreadsheets For prototyping and small projects but breaks with complexity	+ “Good enough” for small projects + Fast to start and universally understood – Manual traceability becomes a nightmare at scale – No versioning, ownership, or change control.	+ Maximum flexibility; engineers can adapt formats freely – No traceability to implementation artifacts – Engineers routinely design against outdated specs – Impact analysis is manual, and error-prone	+ Simple for small tests and informal verification – Manual tracking of verification status – No requirements coverage visibility – Fragmented evidence storage
Legacy Requirements Tools DOORs, Jama, Polarion... For keeping a system of record but hard to use, leading to silos	+ Excellent as a system of record + Strong for formal baselines, change control workflows – High setup overhead with unintuitive interface – Optimized for governance, hindering iteration speed	+ Maximum flexibility; engineers can adapt formats freely – No traceability to implementation artifacts – Engineers routinely design against outdated specs – Impact analysis is manual, and error-prone	+ Simple for small tests and informal verification – Manual tracking of verification status – No requirements coverage visibility – Fragmented evidence storage
Project Management Software Jira/Confluence... For task tracking but lacks traceability and hardware rigor	+ Excellent at coordinating cross-functional work + Basic requirements objects through add-ons – Requirements are a secondary work item – Weak traceability across systems and verification	+ Good visibility into task progress + Clear ownership and execution tracking – Hardware dependencies are poorly represented – Weak linkage of requirements to hardware designs	+ Strong test execution status tracking – Hardware verification poorly represented – Weak backwards traceability for audits – Fragmented evidence storage

Included in All Altium Subscriptions

Is your company already using Altium products for electronics design?

Requirements Portal is included in Altium Develop and Altium Agile Teams subscriptions.

	Altium Develop	Altium Agile Teams
	For engineering teams getting started with structured requirements	For scaling organizations that need governance and process control
Requirements Management	✓	✓
Requirements Traceability	✓	✓
Requirements Verification	✓	✓
Reusable Parameters	✓	✓
AI Assistant	Limited tokens	Unlimited tokens
Test Case Management	—	✓
Customizations	—	✓
Pricing	From $995/year flat* *included in every Altium Develop workspace	Contact Sales
	Learn More	Talk to Sales

References

[1] : NewSpace Index: Satellite Constellations - 2024 Survey, Trends and Economic Sustainability

[2] : From Grassroots to Global: Aaron Rogers on the Small Satellite Revolution

[3] : EnduroSat: How This Bulgarian Company Is Scaling the Global Small Satellite Constellation Market

[4] : The Space Industry in 2024, and How to Build a Satellite Company

[5] : SmallSat Europe 2025 | Endurosat's Simon van den Dries: Size doesn't matter—application does.

[6] : A systematic review of empirical studies on multidisciplinary design collaboration

[7] : Collaborative Engineering for Research and Development

[8] : Collaborative systems for NASA science, engineering, and mission operations

[9] : Innovative interdisciplinary models in engineering education

[10] : NASA Systems Engineering Handbook

[11] : Informing Large Missions from SmallSat/Class D Lessons Learned

[12] : AI-Enhanced Requirements Traceability Using MBSE and LLMs for Complex Systems

[13] : Survey of Verification and Validation Techniques for Small Satellite Software Development