.experience

From idea to impact: Requirements engineering

A magazine by ERNI since 1999.

Editorial

Dear Readers,

As systems grow more complex, requirements engineering has never been more important. But what does that really mean? It's about understanding stakeholders, designing clear processes, organising data smartly and making new technologies like AI work for you – without adding unnecessary complexity.

In this issue of .experience, we explore how pragmatism and professionalism go hand in hand – from strategy to architecture to usability. With real-world examples, we show how lean and efficient engineering can deliver results without compromising quality or compliance.

Get inspired, discover fresh approaches and take away ideas you can apply to your own projects. Enjoy the read!

Sincerely,

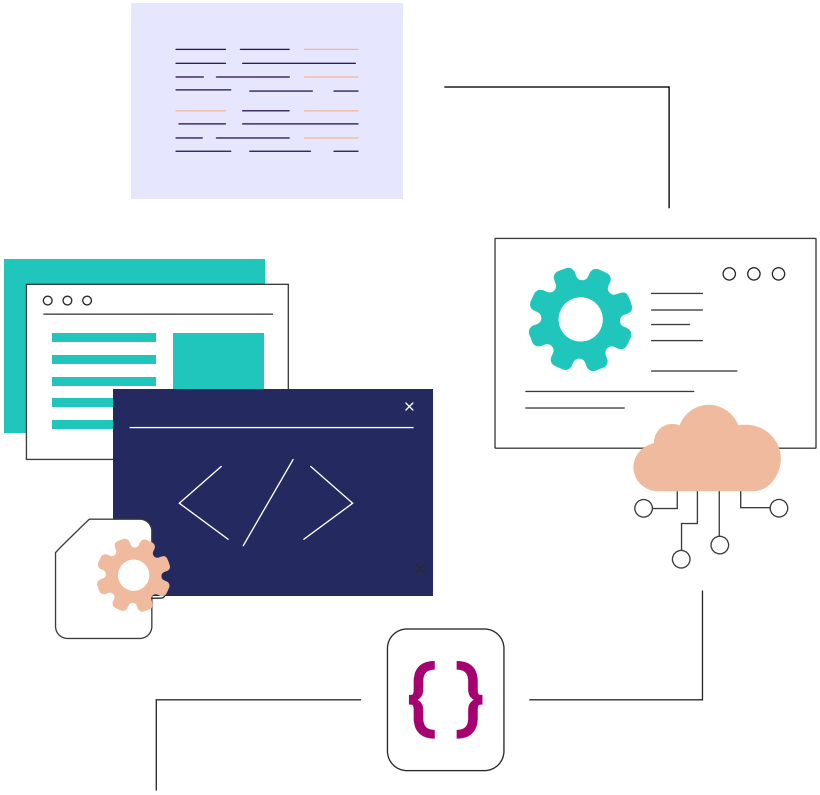
Pavo Kohler
CEO, ERNI Group

Table of contents

How requirements engineering positions itself in modern software development

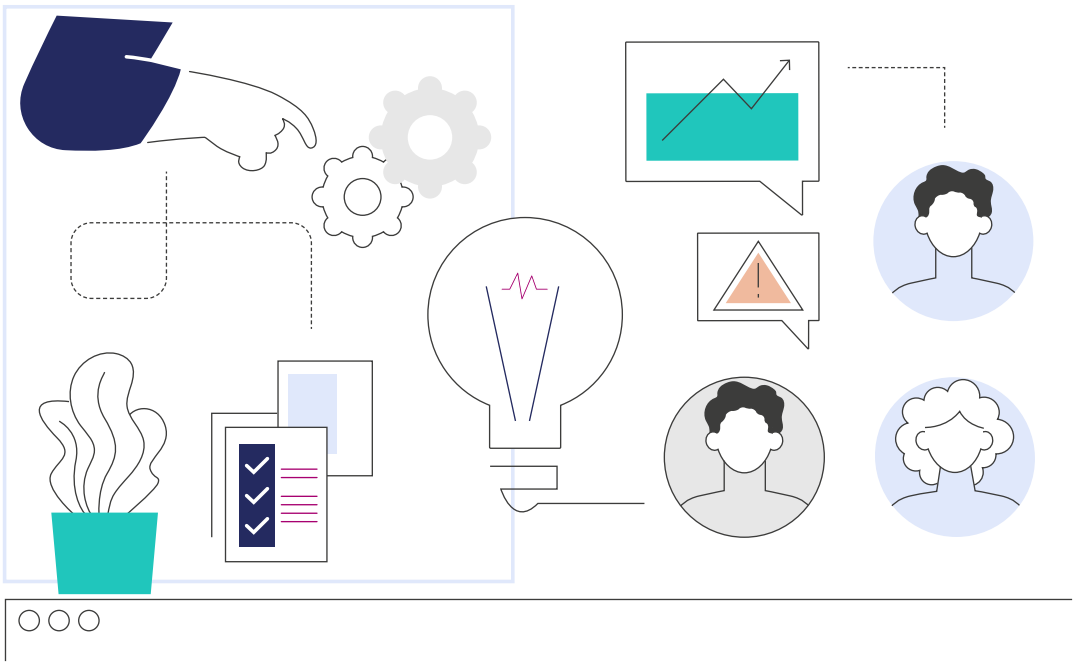
In the digital age, generating ideas has never been easier. Bringing them to life in a user-friendly manner is harder. In the course of time, I see requirements engineering taking on a new set of challenges. With digital products becoming more sophisticated and development cycles faster, aligned adaptable requirements are vital.

By Urs Koepfli, Expert Consultant in Requirements Engineering and Business Analysis, ERNI Switzerland



What requirements engineering (RE) stands for today

From what I have experienced in recent years, the era of conventional, heavy documentation and rigid RE processes is over. Modern RE requires collaboration between cross-functional teams, with an ability to match the developing needs of the stakeholders, and strong traceability over the lifecycle of the requirements. All this is especially crucial in a regulated or high-stakes environment where misunderstandings or incompleteness in the requirements may cause expensive setbacks. Throughout my work, one key transformation has become clear: RE now needs to scale and integrate



seamlessly into agile development environments without sacrificing the detail or compliance needed in regulated industries. In this lead article, I would like to point out what the recent issue of .experience will handle in more detail: how organisations are dealing with today's challenges and developing RE practices to support their operational goals in increasingly complex systems.

For whom do we engineer requirements?

In nearly every project I've worked on, it's been clear that there's never just one 'customer'. There are users, buyers, maintainers, regulators and developers, and they all have their valid perspectives. Taking all of them into account creates complexity. And this is where RE steps into the scene, bringing structure to the complexity.

Stakeholder management, in my view, goes far beyond ticking boxes and collecting sign-offs. It's about actively discovering what matters to each stakeholder and translating those insights into a shared direction. It is about identifying all legitimate voices in the process, learning what they are aiming for and ultimately generating a common direction that guides development. From the field technician working in the rain (or any other weather conditions), wearing gloves and trying to compress a full location on an app; to the Governance, Risk and Compliance Officer who is responsible for protecting information in terms of GDPR; every one of them matters, and all provide critical information that decides future success.

In one of the projects I was part of, the shift in the role of requirements engineering became particularly clear. Initially, the team expected RE to simply collect and document the requirements. But as we progressed, it became evident that our real value lay elsewhere. Acting as the customer's voice within the solution team, we helped translate vague expectations into tangible priorities. Together with developers, designers and business stakeholders, we worked out what should actually be built – not just what was technically possible, but what would truly bring value and be usable in the customer's real context. That's when I realised: requirements engineers are no longer gatekeepers of specifications. They are facilitators of shared understanding and direction.

"It's not about building what the customer says they want – it's about understanding their true goals and helping to find and implement the best solution."

RE is the bridge between business intent and technical realisation. It translates abstract ideas into structured inputs for development. And in agile teams especially, RE ensures that rapid iterations don't lose sight of the overall view. Perhaps most importantly, it helps teams make better decisions early when change is still affordable and the impact high.

Making the system work

RE is not homogenous; it consists of four interconnected tasks:

1. Elicitation

Discover real needs behind the requests through focus group interviews, workshops and observation.

2. Documentation

Requirements are captured in a way that is readable, testable and changeable, mainly through structured text or model-based approaches.

3. Validation

Validation ensures that what was captured is addressing the right problem.

4. Management

Continuously track changing priorities and ensure traceability throughout the lifecycle.

In more agile situations, RE changes from a very heavy lift at the front to more continuous collaboration as the work transitions from documentation into general continuous interaction and information sharing with stakeholders. Also, the capabilities of tools evolve – from AI-assisted analysis of regulatory texts to a collaborative prototyping platform always with the one goal in mind – to ensure that the team is building the right solution.

The first 100 hours – Laying a solid foundation

From my experience at ERNI, I've seen again and again how the first days of a project determine its overall success. One case comes to mind where this became very clear. We were working together with a customer in the infrastructure sector, and the task seemed straightforward at first glance: deliver a mobile app for field technicians to document maintenance activities on site.

The customer expressed a clear idea of what they wanted, at least on the surface. A sleek interface, fast performance and seamless backend integration. But during the initial discussions, our requirements engineering team dived deeper. We conducted field visits, spoke directly with the technicians and tried to understand how and where the app would be used. What we discovered changed almost everything.

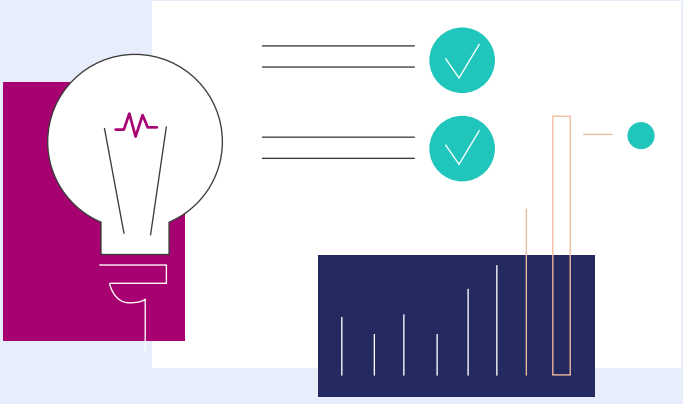
Most of the technicians worked outdoors, year-round, often in cold and wet conditions. They wore gloves, operated in low-light environments, and had minimal time to navigate complex menus. This was not covered in the original briefing. Had we rushed into development based only on the initial specifications, we would have built something technically sound but practically unusable. In fact, a previous attempt by another vendor had failed for precisely this reason. The UI simply didn't work in real-world conditions.

This experience reinforced a principle we hold in our projects: you need to build the right foundation early. Or, as I often say:

"If you're building a four-storey house, you need a different foundation than for a garden shed."

In our RE practice, we treat the first 100 hours of any project as a critical window in which to explore, understand and align. The steps we take during this phase typically include:

1. Understanding the business context and objectives	Clarifying goals, constraints and the underlying drivers for the project.
2. Engaging all relevant stakeholders	From end users to compliance officers, to ensure no critical voice is missed.
3. Defining the problem space	Capturing the pain points, system boundaries and environmental conditions.
4. Eliciting high-level requirements	Focusing not just on features but also on usability, performance and constraints.
5. Structuring and prioritising early inputs	Using proven methods to separate must-haves from nice-to-haves.
6. Scoping the initial solution	Defining what will be delivered and, just as importantly, what won't.
7. Communicating clearly with the team	Establishing common language, shared tools and aligned expectations.
8. Assessing risks and feasibility	Surfacing any technical, organisational or regulatory risks as early as possible.



That project succeeded not because we followed a checklist, but because we took the time to listen, observe and question before building. That, to me, is the essence of requirements engineering done right.

The future of requirements engineering

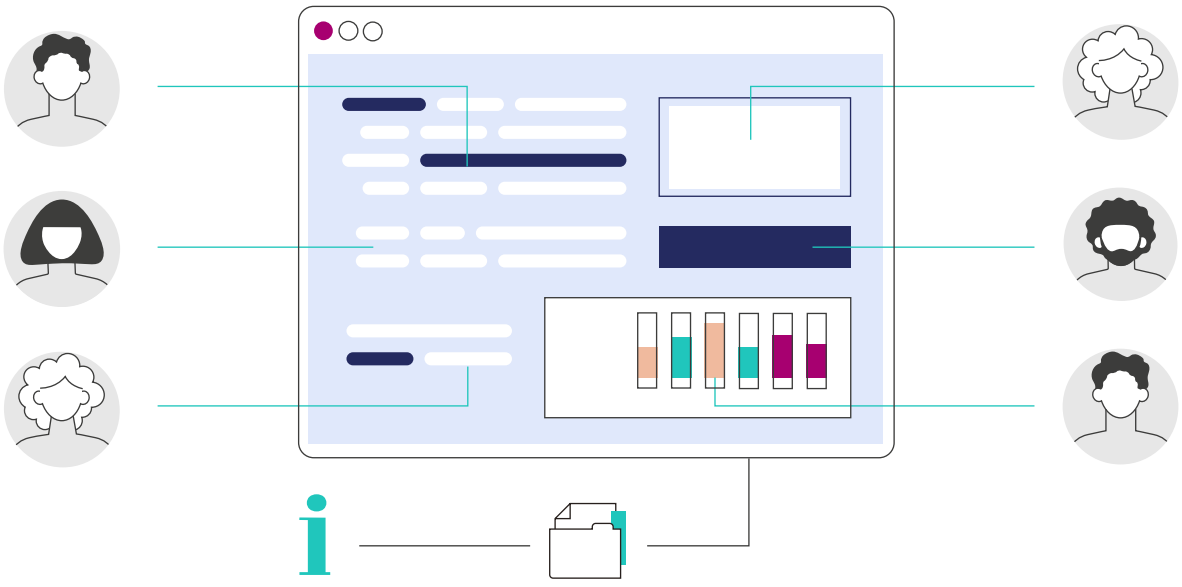
RE as a discipline hasn't changed at its core – but its context certainly has. It will no longer be a phase – it will be a continuous, iterative activity that keeps pace with product development. In agile environments and while leveraging DevOps pipelines, requirements will evolve in real time and therefore will need tools and practices to effectively accommodate continuous refinement, stakeholder consent and traceability. As system complexity increases (particularly in regulated domains such as MedTech and automotive), RE will shift further towards formalised models and Model-Based Systems Engineering (MBSE) to manage the interdependencies. Visual models will reduce communication overhead across business, technical and compliance teams.

Future RE will not only be about technical specifications and functional scope but also more broadly about understanding user needs, outlining business objectives and delivering value.

Agile ways of working, the proliferation of tools, and especially AI are reshaping the 'how' of RE. But the 'why' remains the same: building shared understanding. A frequent question we hear is whether AI will replace RE professionals. 'Replace' is not the right term; a more proper one would be 'augment'. If AI can summarise lengthy compliance documents or create draft user stories in a speedier manner, why not use it? Yet one thing AI cannot replace is the human judgment of balancing business objectives, user needs and technical realities. This remains the job of requirements engineers, now and into the future.

Conclusion

Requirements engineering is no longer just about specifications; it's about a shared vision. It represents the foundation for agile, scalable and human-centric software. Today, RE is less of a defined set of roles, and more about flexible collaboration. It is increasingly a function that embeds not just a process, but a way of thinking into product teams.

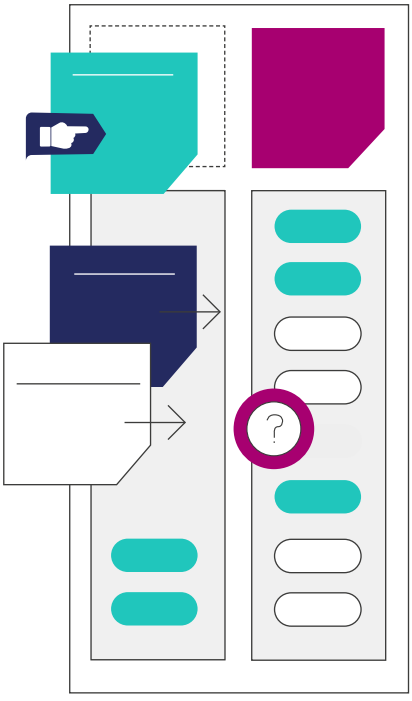


The pragmatic architect: Scaling without overengineering

Complexity is tempting, especially at the start of a project, when ideas are flowing and architectural ambitions run high. But without clear priorities and precise requirements, systems end up built for hypothetical scenarios rather than real benefits. A pragmatic, YAGNI-inspired approach helps you start lean while staying scalable.

By Mihaly Fodor, Principal Engineer, ERNI Romania

Try to imagine a situation many of us know too well. Your software engineering partner has just received your new project assignment, and the kick-off meeting is charged with enthusiasm. As the customer, you arrive with a bold idea, an ambitious delivery date, and the hope that smart technology will rise to your challenge. Within minutes, the whiteboard disappears under sticky notes and marker lines: boxes, arrows, acronyms, tool names, deployment diagrams. Voices discuss Kubernetes clusters, serverless functions and the merits of microservices versus a monolith. Passion grows, and in less than an hour, the team is deep in architectural debate while user stories still sit blank. Then someone finally asks, “What exactly should the user achieve in version one?” The room goes quiet. That silence exposes a familiar hazard: when enthusiasm outpaces focus, the team risks designing for imagined futures rather than present needs. If requirements remain vague and unprioritised, architecture becomes guesswork, and the solution ends up optimised for hypotheticals instead of real value.



The temptation of complexity

Requirements engineering is more than a checklist item; it is the foundation that supports a reliable architecture. Teams need a practical, well-defined scope that separates what must be delivered today from what can wait for tomorrow. This clarity protects the project from unnecessary complexity. Sound architecture begins with a clear purpose, not with a tool or pattern.

Yet many projects still begin with a wish list: “We need the full stack.” Microservices, serverless functions and multi-cloud setups appear before anyone writes a single user story. I once worked on a project where the customer believed an elaborate cloud design would boost B2B sales, while the solution to their challenge was simply a dependable ordering system. The gap between perceived

needs and actual needs is where complexity grows. Here the YAGNI principle – You Aren't Gonna Need It – proves its value. YAGNI keeps the team focused on delivering only what is necessary, when it is necessary – and it grounds every architectural decision in real requirements instead of assumptions.

Why over-engineering happens

Over-engineering often starts with good intentions but misplaced assumptions. Four patterns appear again and again:

Four common patterns causing complexity

1. Unrealistic forecasts

Designing for 'millions of users' when the first-year estimate is only a few hundred.

2. Imitation without context

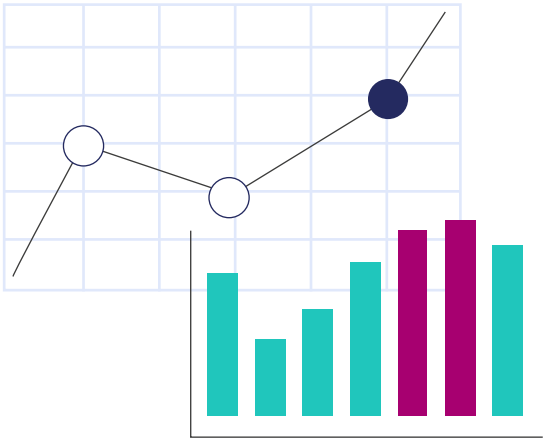
Adopting the architecture of companies such as Netflix without matching their scale or budget.

3. Hidden cost traps

Selecting a pay-per-request serverless model even though the workload is steady and always running.

4. Scope drift

Adding nice-to-have features that quietly become mandatory and inflate complexity.



These choices are usually driven by fear of future unknowns, blurred non-functional goals, or a culture that rewards complexity. The results are clear: slower delivery, fragile and complicated systems, higher operating costs, and decision fatigue that drains team energy. By the time a project team sees it has tried to do too much, the cost of change is already high.

The pragmatic process

My approach to requirements engineering is practical and tied to real-world limits and business value. It starts with orientation: a detailed review of every document, removing irrelevant claims to expose the few requirements that truly matter. Once clear, I move to sizing, checking core metrics such as expected users, data volume, service levels and release pace. Any unclear figure becomes an immediate question for the customer. I rate functional scope with T-shirt sizing (XS to XL), a quick method that flags oversized or vague demands early.

With this baseline, I sketch a minimal viable architecture – often just three boxes and two arrows. Anything that cannot fit on the whiteboard is challenged; if it does not fit, it likely does not belong in the day-one plan. Before estimates, a peer reviews the sketch and tests every assumption. The last checkpoint is to return to the business goal. If a design element does not support the main KPI, we drop it. The result is a lean, validated foundation shaped by evidence, not speculation.

A YAGNI-inspired reference case

A mid-sized manufacturer of sun-protection systems once asked us to design an AWS solution based on microservices, auto-scaling and separate teams for each service. The aim was to ‘future-proof’ a new ordering platform and keep it ready for growth. On paper, the plan looked modern and solid.

We soon learned that expected peak traffic was less than fifty concurrent users, far below the level that justifies a distributed setup. Even our first draft still contained load balancers and stateless services, assuming growth that was years away. The first AWS bills showed the flaw: the system was tackling problems that did not yet exist.

Using a YAGNI mindset, we removed the excess. We replaced the spread of microservices with a modular monolith, keeping clear internal boundaries so parts could be split later if demand rose. Load balancers went away, and deployment became straightforward.

The company met every business goal at roughly one-fifth of the former infrastructure cost. Time to market improved, maintenance became simpler and the team could focus on core features instead of cloud overhead. Just as important, the platform can still grow when real demand appears.

This case shows that sound architecture starts with what is necessary, not with what is fashionable.

Lessons learned

The lean approach works because it stops scope from expanding too early. Clear limits produce simpler designs and more accurate estimates. Most pushback comes from a fear of lock-in. I address this by showing the defined seams where parts can be swapped later, but only when a measurable KPI demands it. In highly regulated or safety-critical fields, such as real-time trading or medical devices, extra robustness may be required from day one. Even then, every added layer should have a written, testable reason to exist.

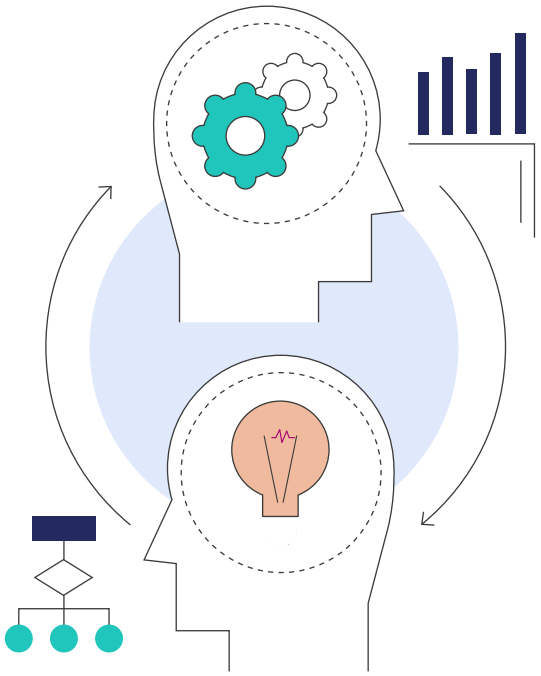
Why a designer-developer integration delivers better results

In a recent project, I managed the modernisation of a key enterprise application. The goal was clear: Update an outdated tech stack to enable scalability and maintainability. What began as a technical upgrade soon became a lesson in cross-functional collaboration, especially between software development and design.

By Nicola Reinhard, CEO Office, ERNI Switzerland

A dual focus with one timeline

At the beginning, we defined an ambitious yet achievable timeline. We aligned it carefully with development capacity and validated it with business stakeholders. We primarily focused on upgrading the outdated technologies without altering the existing functionalities. Developers were laser-focused on this, knowing that success meant delivering a state-of-the-art application on time.



However, alongside the tech upgrade, we aimed to improve the user interface and the overall experience to create a modern look and feel that would delight end users. Therefore, a UX designer joined the team. While this was a strategic decision, it added a second layer of complexity. What was the risk? A potential conflict of interest and timing.

Designers wanted to rethink interactions, layouts and flows. Developers had already started to rebuild the UI on the new stack, and they had design opinions of their own. Taking design revisions into consideration meant additional work and adjustments – none of which had

been included in our original timeline. This disjointed approach threatened to result in a fragmented experience for the user and potential friction within the team.

Inspired by factory floor thinking

A turning point came unexpectedly, while reading Walter Isaacson's biography of Elon Musk. Musk placed his design team on the factory floor, asking them to resolve breakdowns with the engineers in an iterative and collaborative way. This integration of problem solving across design and engineering disciplines inspired me.

I knew we needed the same kind of partnership. Designers and developers couldn't work in silos. We had to remove the handover mindset and foster a shared ownership model.

Real integration, not just collaboration

As soon as the design team was on board, we ensured they fully participated in all development meetings. We established a weekly design meeting to align on priorities and decisions. We captured all design decisions on a shared Miro board, making all decisions visible, organised and collaboratively maintained. Three success factors emerged:

Ownership through design authority

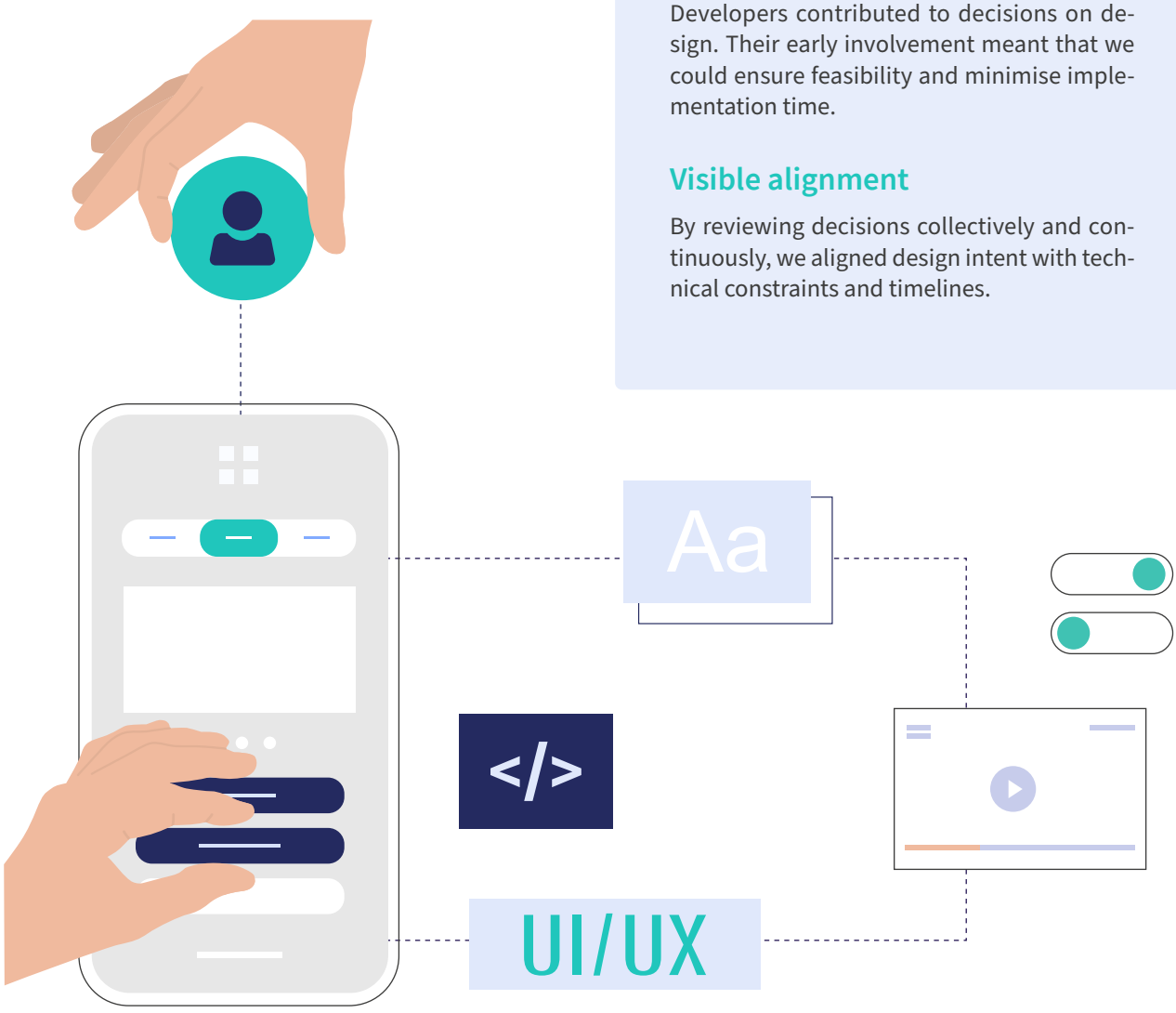
All significant design decisions were funnelled through the designers. This didn't mean blind execution – this meant a thoughtful discussion based on rationale and principles.

Shared understanding

Developers contributed to decisions on design. Their early involvement meant that we could ensure feasibility and minimise implementation time.

Visible alignment

By reviewing decisions collectively and continuously, we aligned design intent with technical constraints and timelines.



The navigation dilemma

A real test of our collaboration came early. Before the designers joined, we had finalised the application's new navigation system. Upon arrival, they flagged several concerns: it was unintuitive, inefficient due to dense screens, and failed to address critical user paths. What was the challenge then? It was already implemented, and our backlog was full.

Instead of derailing the project or forcing a compromise, we agreed to postpone changes to the navigation. We decided to focus on the technology migration first, and revisit the navigation once capacity allowed. This deliberate deferral paid off. Not only did we meet our deadline but we also successfully redesigned the navigation just ahead of go-live.

Performance, speed and quality

Initially, bringing the designers in later posed a challenge. But the way we integrated them into the team made the difference. Developers respected their decisions because they were well-grounded and included their concerns. The designers, in turn, evolved with the team – eventually even contributing directly to the codebase.

This high degree of collaboration drove faster implementation and a more cohesive user experience. The team felt ownership of both the design and the code, and the end product reflected that alignment.

Goal: Creating digital experiences

Design and development don't need to be separate tracks – they work best when they are tightly interwoven.

Integration doesn't mean all designers have to start coding. It means fostering dialogue, trust and mutual respect from the beginning.

Developers bring deep insights into feasibility and user expectations. Designers bring the vision of usability and aesthetics. Together, they can build more than just functional software; they can craft exceptional digital experiences.

My advice to fellow project managers and technology leaders: don't treat design as a phase or an add-on. Embed it. Own it. And watch your teams and your applications thrive.

From life sciences to diagnostics: Engineering requirements for a regulated release

Transforming a life science device or research-use-only (RUO) device into a medical one is a process full of opportunities and complexities. In one project, we supported a company in converting a lab diagnostics device for clinical use. This article highlights the journey in a regulated setting and how expertise drives success.

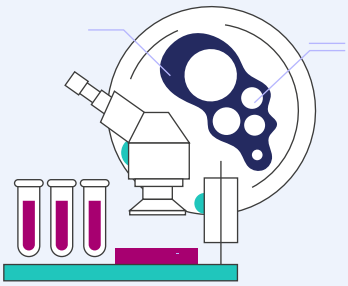
By Ares Cabó Carrera, Expert MedTech Consultant and Product Owner, and Carolina Lezama, MedTech Delivery Manager, ERNI Spain

The challenge

While the core technology appeared to be the same, patient safety, legal liability, traceability and other documentation changed almost everything. The regulatory requirements, expectations and scope of controlled processes expanded vastly – and with this, the expectations on the product developers changed, shifting from speed or flexibility of development to full accountability and validated traceability.

In this instance, the challenges became evident and were multifaceted – fragmented requirements; the ever-changing needs of stakeholders; deadlines; ISO, FDA and EU MDR compliance; and the awareness of the need to prevent costly rework and compliance failure.

Difference between a life science device and a medical product



Life science device

A life science device is a piece of equipment, tool or instrument used in biological, biochemical, pharmaceutical or laboratory research. It supports scientific discovery, drug development, diagnostics and biomanufacturing. It is not always in direct contact with patients.

Medical product

Medical products include, among other things, instruments, objects, substances and software that are used for therapeutic or diagnostic purposes in humans. They are intended for clinical use, meaning on or in patients, and regulated by health authorities like the FDA (US), EMA or MDR (EU) and classified by risk levels (e.g., Class I, II, III).

The process: A journey to regulation

Our approach to these challenges was requirements engineering (RE). RE provided a traceable and structured foundation from the start so we were able to keep technical portions of the development aligned with regulatory needs, manage change appropriately and effectively, and ensure that no critical detail was overlooked or forgotten. RE served to be the lever moving us from scientific discovery to clinical compliance.

Transforming software from a life science research tool into a regulated diagnostic medical device is not a linear upgrade; it's a fundamental shift in mindset, methodology and responsibility. This transformation journey took our teams from the relatively flexible realm of scientific software into the rigour of regulated diagnostics, where every function must be justified, every risk mitigated and every line of code traceable.

In regard to research-use-only products, teams often optimise for speed, functionality and exploration.

Documentation is less heavy, testing is pragmatic and risk management is rather informal. In contrast, a medical device – subject to FDA and IVDR requirements – demands process maturity, systematic traceability and verified quality at every stage.

That meant rethinking how we worked, from how we documented and validated user needs to how we verified software behaviour under real-world clinical conditions. Risk management was no longer a background activity; it became embedded in every backlog refinement and decision checkpoint.

Reimagining agile in a regulated world

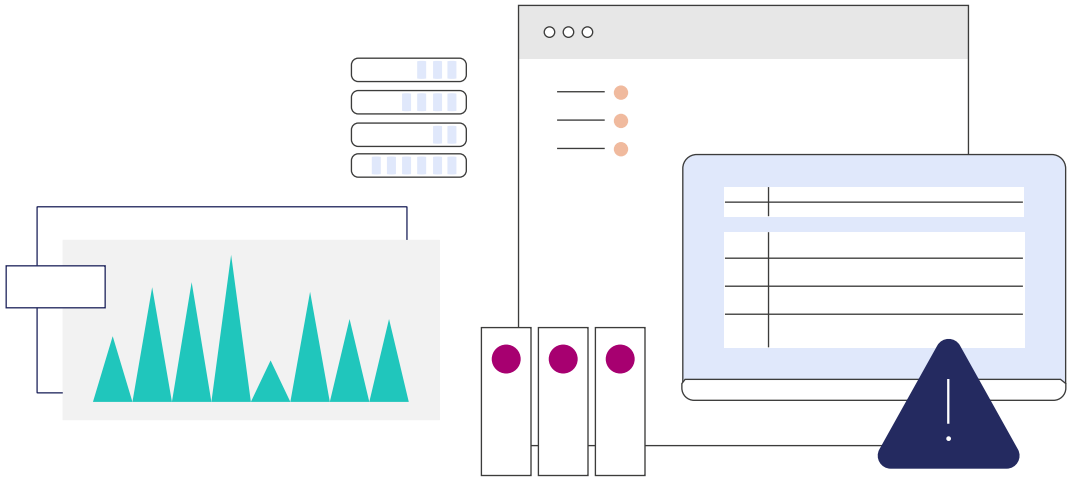
Agile principles remained at the core of our approach, but we needed to adapt them to fit a compliance-driven environment. Instead of fast iteration at the expense of traceability, we designed a hybrid agile model that preserved adaptability while ensuring full traceability, validation and compliance.

- **Creating incremental release candidates instead of one monolithic launch. The modularity in the architecture had the purpose of optimising the code and enhancing future maintenance.**
- **Defining minimum viable products (MVPs) that were not only functionally usable but also regulatory-compliant. This way, we were able to deliver viable results to the customer at an early stage.**
- **Accepting that definition of done went far beyond 'working software'. Each user story had to be linked to validated requirements, test cases, acceptance criteria and documented verification outcomes. If it wasn't testable and documented, it wasn't done.**

With retrospective, in a big project like this with a hundred team members distributed over multiple sites across Europe, we would choose a different agile framework – most likely SAFe – due to reasons like too many diverse teams or not all teams following the same iteration process.

Risk at the core

We integrated risk management directly into backlog refinement. Stories were sliced not just by business value or complexity, but by risk exposure and regulatory criticality. For example, data integrity-related features – such as audit logs and result traceability – were front-loaded in the roadmap due to their impact on patient safety and compliance. This helped the team prioritise under pressure, especially when the scope had to be negotiated due to tight timelines. The MVP was not the bare minimum – it was the minimum certifiable product.



Product ownership redefined

In this setting, product owners (POs) were more than feature gatekeepers. They became translators between stakeholders, compliance teams and developers, ensuring that evolving customer needs were interpreted correctly and delivered in line with regulatory expectations. POs and requirements engineers worked side-by-side to bridge clinical needs with development constraints, constantly verifying that each deliverable could withstand regulatory scrutiny – before it reached the end of a sprint, let alone the market.

Documentation: A living organism

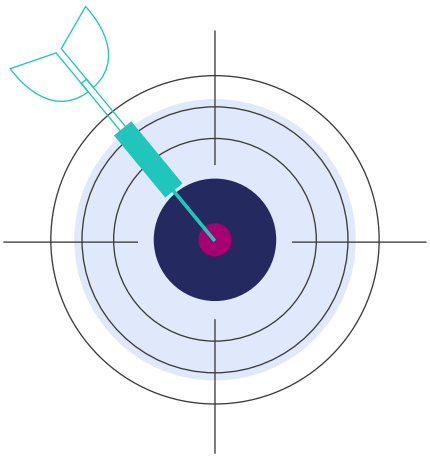
Throughout this project, we embraced documentation not as overhead, but as a living asset – a dynamic, evolving body of knowledge that enabled safe decision making, ensured regulatory compliance and unlocked scalability. Each release Candidate was treated as a self-contained milestone. Rather than front-loading all documentation or postponing it until a final release, we built it up incrementally and consistently. Each user story tied into a broader user journey. In this way, documentation became a tool for alignment and quality, not just compliance. It allowed every stakeholder, from developers to regulatory reviewers, to understand what was built, why it mattered and how it was verified.

Solution

Together with our customer, we delivered not only a compliant and validated device but one that is modular, scalable and ready for the global market.

We are happy the product is now on the market – feedback that we have received has been positive, not only from internal stakeholders but also from end users in clinical environments who now benefit from the device.

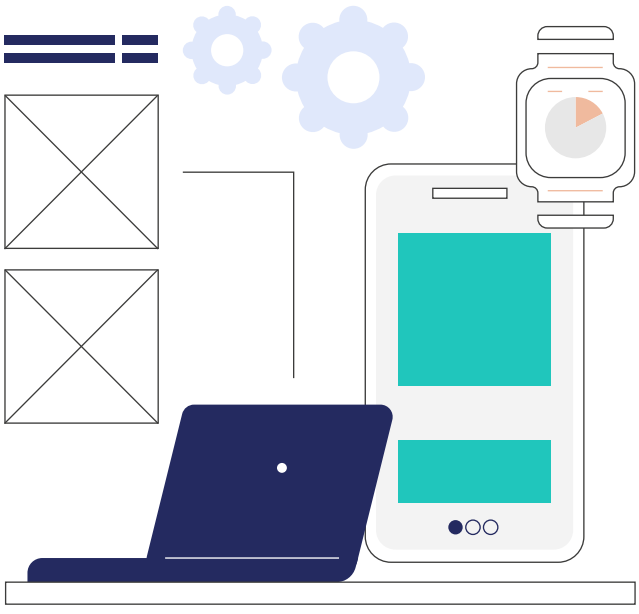
Among the key lessons we learned, we would say that early alignment on product requirements is critical; agile practices must be tailored, not transplanted, into regulated environments; and above all, requirements engineering and documentation must be treated as strategic enablers – not administrative burdens.



Real usability engineering seamlessly integrated for medical devices

In medical device development, usability engineering often plays a minor role, treated as a downstream task before regulatory submission. Yet integrating usability early creates safe products that stand out. This article shows how usability drives efficiency, quality and market success, even in regulated environments.

By Simon Brendel, Senior Consultant and Stefan Siegle, Principal Consultant, ERNI Germany



Usability – Often underestimated, rarely done right

Regulatory requirements – such as those defined in IEC 62366, MDR (Medical Device Regulation), IVDR (In Vitro Diagnostic Medical devices and Repealing Directive) or the FDA Human Factors Guidance – demand a minimum level of usability, primarily aimed at avoiding harm. However, meeting these requirements alone is not enough to create a good product. These standards require that risks are addressed – but not that a product is intuitive, efficient or even enjoyable to use. This underestimation often leads to usability activities being started too late or approached half-heartedly in the MedTech context – resulting in costly redesigns or, in the worst case, flawed products.

Start early, prioritise wisely

Whether it's chief physicians, lab workers or nurses – every system has multiple user groups with different tasks and needs. Ignoring this diversity puts the acceptance of the product by entire user groups at risk – and can ultimately lead to failure.

Another major barrier to establishing genuine usability across the MedTech sector is that the people who actually use the products are often not the ones who decide which product to buy. Instead, purchasing decisions are typically made by procurement teams or clinical leadership – individuals who rarely work directly with the devices. As a result, usability rarely becomes a decisive factor in purchasing decisions, even though it plays a critical role in daily clinical routines when it comes to safety, efficiency and acceptance.

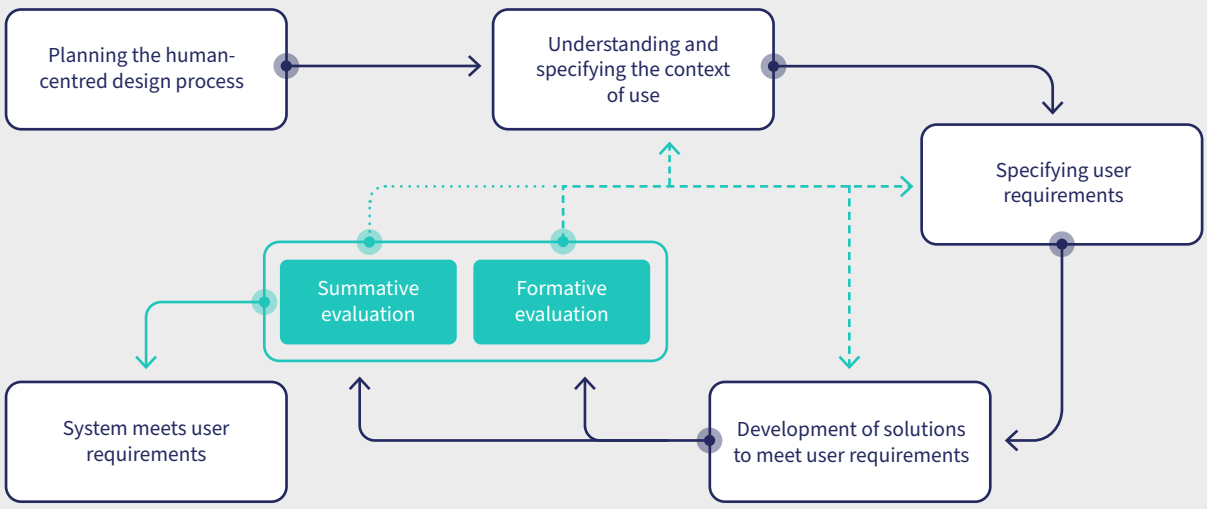
In fact, efficiency gains achieved through good usability can make a significant economic difference in the long term. So why isn't this topic given the attention it deserves?

Compliance as a byproduct of good processes

Usability engineering should be an integral part of a solid requirements process – not something treated as a nice-to-have. Teams that analyse user contexts early, develop task models, create personas, and test iteratively will naturally generate the artifacts needed for regulatory approval. These deliverables don't arise from a sense of obligation, but as a logical outcome of a sound development process – grounded in real insights, not assumptions.

From users to artifacts: A lean process

Figure: The ERNI Medical Usability Process builds on the human-centred design process defined in ISO 9241-210 and adds key elements required for the approval of medical devices.



The process we developed to prevent real usability engineering from being neglected in MedTech projects is deliberately based on the human-centred design process as defined in ISO 9241-210. This approach is widely used in industries where high-quality usability has historically been given far more weight than in the medical device field, such as e-commerce, gaming and entertainment or B2C app development.

Our process follows the classic steps of human-centred design: planning, understanding and specifying the context of use, specifying the user requirements, and producing design solutions. When evaluating prototypes and other development artifacts, we differentiate between formative and summative evaluations. Our approach works equally well within V-models, agile environments or hybrid frameworks.

If all steps are executed carefully and at the right time, the process naturally results in both widely accepted usability deliverables and all artifacts required by regulatory standards:

1. Planning the human-centred design process

- **From DIN EN ISO 9241-210**
  Human-centred quality objectives, resource plan, schedule, user group definitions

2. Understanding and specifying the context of use

- **From DIN EN ISO 9241-210**
  User group profiles, task models, personas, scenarios of use, user journey maps
- **From DIN EN 62366-1**
  Safety and use errors, use specification, hazard-related use scenarios, known and foreseeable hazards and hazardous situations
- **From FDA Human Factors Guidance**
  Description of the context of use (composed of the other artifacts in this step)

3. Specifying the user requirements

- **From DIN EN ISO 9241-210**
  User needs, user requirements
- **From DIN EN 62366-1**
  User Interface Specification (Requirements), User Interface Evaluation Plan, Critical Task Description

4. Producing design solutions to meet user requirements

- **From DIN EN ISO 9241-210**
  Prototypes, scenarios of use, storyboards, user journey maps, task models, information architecture, navigation structure, style guide

5. Formative and summative evaluations

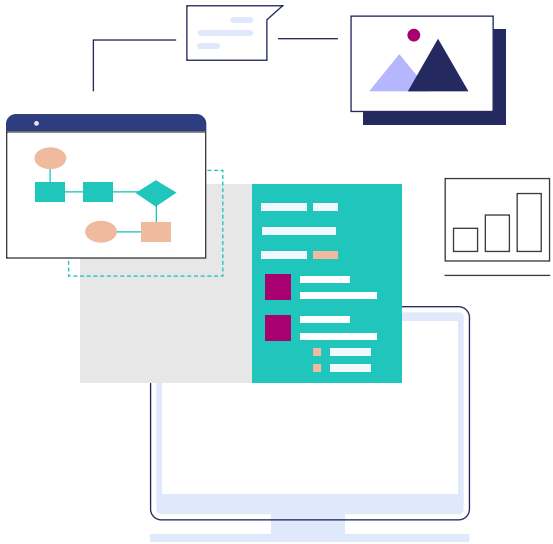
- **From DIN EN ISO 9241-210**
  Evaluation reports
- **From DIN EN 62366-1**
  No specific artifacts required – however, all evaluation activities and results must be documented in the Usability Engineering File, which serves as a structured record of all usability-related activities.
- **From FDA Human Factors Guidance**
  HFE/UE Report, Description of User Interface, Process and Interaction Documentation

Usability engineering is requirements engineering

In practice, usability engineering and requirements engineering are difficult to separate. Insights gained through human-centred design – interviews, observations and context analyses – directly feed into requirements. Many findings from usability activities become critical inputs for defining requirements. That's why these two disciplines should be closely integrated, ideally combined in a single role or a tightly aligned team. A project without a dedicated usability lead misses out on valuable potential – and risks failure.

In order to align both disciplines effectively and to ensure good product design is not lost in the face of perceived regulatory complexity, it is important to regularly reflect on the following questions:

- Do I question what others take for granted?
- Is important information being lost in communication – like a game of telephone?
- Have I truly understood the real problem?
- Could I draw a detailed picture of how and where my product is used?
- Can I describe each step of the tasks users perform with my solution as a tool?
- Do I accept a single person as a representative of an entire user group?
- Have I personally observed the users and the context in which the product is used?
- Have I considered all relevant user groups – and involved them from the start?
- Am I relying on so-called experts without having spoken to actual users?
- Am I doing usability engineering from behind a desk?



Conclusion

Usability is not an add-on for regulatory approval; it is a strategic success factor, provided it is taken seriously. In our projects, we often see the differences between formally fulfilled requirements and true usability. With the ERNI Medical Usability Process, we support our clients in asking the right questions from the very beginning, consistently involving users and viewing usability not as a duty, but as a tool. That's how we jointly create products that are not only safe but also compelling and commercially successful.

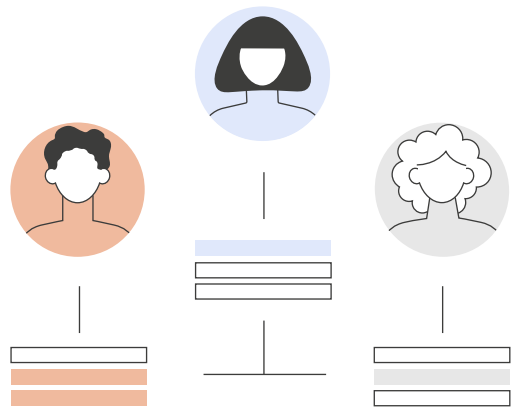
How to close the gap between end users and IT

The success of digitalisation projects depends not only on technology but also on end user acceptance, especially when users are less tech-savvy. Feeling unheard causes frustration, but it is never too late to bridge the gap between users and IT, even in later development stages.

By Caroline Badoud, Senior IT Consultant, ERNI Switzerland

When IT feels out of reach

Technically, development teams might have built a solid IT application. But there will still be this one challenge: in case the end users are resistant to adopting the new system, the finalisation of the project might leave a bad taste in developers' mouths. Bridging the gap between IT and end users is essential for a success story. It represents one of the main focuses of a requirements engineer.



In this article, I'm focusing on bringing IT teams and end users together, especially during the delicate phase of deploying a digitalisation system. This is the crucial phase where users start adopting the new system and start integrating it into their daily business. The users start loving the system or hating it. I'm convinced that it is never too late to close an existing gap between certain users and IT, even at later stages of development. Nevertheless, the positive adoption of a new system requires a structured approach to include everyone and make IT easily understandable. It requires a collaborative mindset of the development team, as well as appropriate planning, to ensure a smooth rollout for every user. In this article, I share insights on how a potential gap between users and IT can be closed – with minimal effort, cost and resistance.

How to facilitate end user acceptance, even at a later stage of development

The approach lies in creating a culture of mutual respect and continuous dialogue. Personally, I approach projects with a focus on building trust, understanding business needs as deeply as possible and developing solutions together *with* the users – not just *for* them.

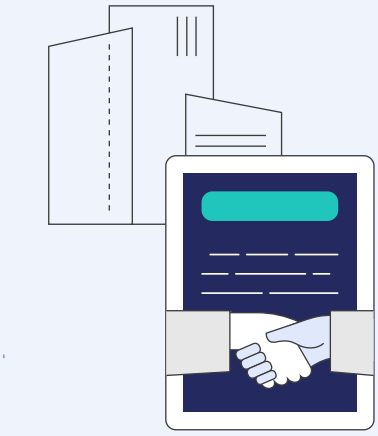
Organise focus group dialogues to identify potential concerns. Based on these insights, a tailored set of measures can be defined. I am convinced that the following six steps can significantly contribute to bridging the gap between end users and IT.

1.

Raising engagement, on site, while keeping groups small

Building trust starts the moment you show up in person where the users are located. Whether it is on a production floor, in logistics or elsewhere, IT teams need to meet users at their actual workplace. That can also mean gearing up in a safe manner and stepping out to where the IT system will later be used. Physical presence shows true interest in users' daily reality.

Engage the users of the application in conversation in focus groups. Try to keep the number of participants small. In fact, the users will be more likely to participate actively, start dialoguing, ask questions and feel engaged in a small group. It is the best way to grasp workflows, concerns and pain points first-hand far beyond remote calls or ticketing systems.



2.

Speak the user's language

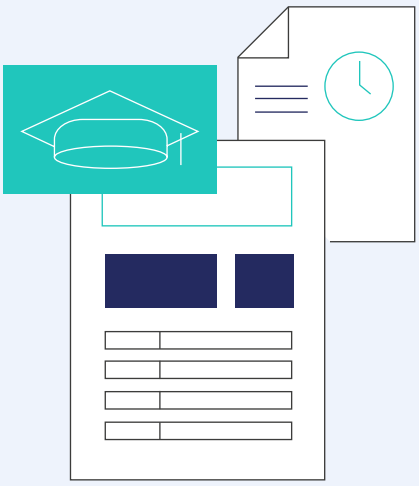
Specifically for multi-language applications, it is a big challenge to include every end user. No user should feel excluded just because he or she doesn't speak the same language as the development team. This exclusion leads to user frustration and possibly to resistance against the new system. Whenever possible, we recommend having a consultant on your side who is multilingual and an IT specialist; it can enormously improve user satisfaction and motivation.

It's not only about the spoken language but also the vocabulary used. Technical teams frequently underestimate what effects their vocabulary may have. IT jargon can be overwhelming if the end users are not tech-savvy. My motto is: clarity connects. Complexity does the opposite – it tries to impress. And that for sure is not the main goal of good communication. Using the same terms and concepts that users are familiar with helps to create a shared understanding. It signals that you're on the same level, not speaking from far away. This doesn't mean 'dumbing things down' – it means choosing empathy over ego.

3.

Trainings tailored to user needs

It is obvious that the best training approach for users is to conduct practical, hands-on sessions where users start interacting with the live system early on. This practical approach boosts motivation and comprehension by focusing on real use cases. But the success does not lie only with the content of the training; its schedule is also important. Often, training and rollout schedules are determined by IT, without fully taking users' operational constraints into account. Involving the groups of users in scheduling the rollout according to their readiness fosters goodwill and accelerates adoption, creating a user-centred and flexible implementation process.



4.

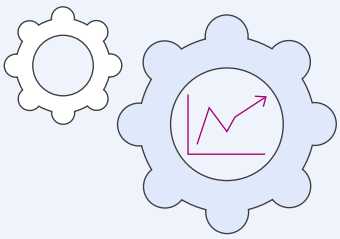
A smooth rollout instead of a Big Bang

Performance and load tests of the IT system are probably the most challenging to simulate in a development environment; sometimes, the practical experience in live operation is different from the simulation. No developer wants to experience a system overload in production. The recommended approach is a gradual rollout, adding users incrementally week by week. This allows close monitoring of system performance and proactive handling of issues. Early adopters help identify bugs so that by the time more hesitant users join, the system runs smoothly, and adoption proceeds with minimal disruption.

5.

Make 'operations' an integral part of the development team

Often, developers prefer developing new features rather than maintaining an existing system. As a facilitator, you should encourage your development team to love user feedback from operations. Make feedback from the users visible, organise a regular operations meeting within the team, prioritise the feedback and let it flow into the development plan. Ensure that each development ticket includes steps for testing, user documentation or communication. Over time, this approach not only improves system stability but also helps teams see the value of user input, fostering a collaborative mindset.



6.

Maintain close collaboration after the rollout

The work doesn't end once the system goes live – in fact, that's often when the most valuable dialogue begins. Make it a priority to maintain strong collaboration over time by holding regular user meetings where experiences can be shared openly. Users talk about what's working and where challenges remain, while the development team presents current and upcoming features and actively seeks feedback.

It is especially effective to involve enthusiastic users as pioneers, giving them recognition and a voice in shaping future improvements. This ongoing engagement builds a sense of ownership, helps developers understand the 'why' behind requests, and turns users into ambassadors for the system. The pioneer users become motivators for the potentially more hesitant colleagues in their team.

Conclusion

Mainly through these six simple measures, the possibility of a divide between IT teams and end users can be reduced – leading to higher satisfaction and stronger system adoption. Repeatedly, I've learned that trust and open communication establish the foundation of any successful project. When users feel seen, heard and genuinely involved, they are far more likely to embrace and support the end result. This might sound self-evident, but in practice, it takes consistent effort, planning and commitment from everyone involved. By fostering such an environment, users become motivated partners in digital transformation, laying a strong foundation for future innovation within organisations.

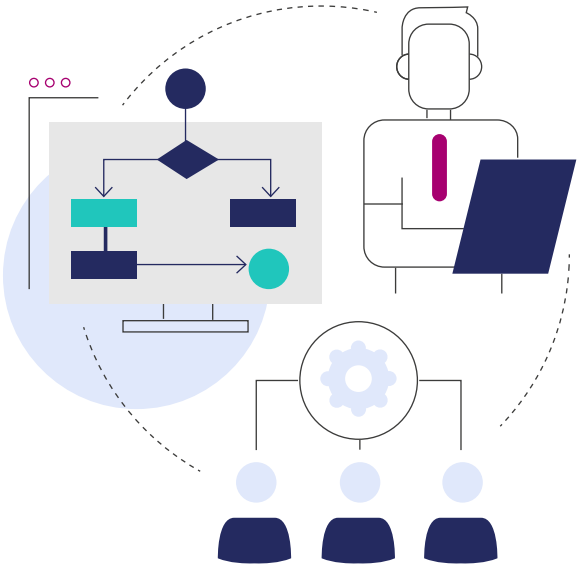
Managing requirements engineering in complex projects with a digital system model

In developing complex medical systems like automated diagnostic devices or patient monitoring platforms, precise requirements capture is both a regulatory and design necessity. Document-driven approaches fall short. We show how requirements engineering as a strategic modelling discipline lays the foundation for digital system models.

By Thomas Ruckstuhl, Requirements Engineer, ERNI Switzerland

The impact of complexity

Each regulated area has its own set of specific requirements that demand intense attention to detail. Especially in heavily regulated domains, but also in others, it is most often mandatory to carefully document the work performed, how the system should operate, what it can and cannot do, and how it needs to be tested. Having experience with complex projects in any regulated area can be of huge benefit for any other regulated domain.



Aviation and aerospace are just two more examples of complex branches where the development environment is characterised by rigorous regulatory requirements, a focus on quality and risk management, interdisciplinary collaboration and the integration of advanced technologies. Companies must remain agile and proactive in addressing these complexities to successfully bring safe and effective medical devices to market.

Five causes of project complexity (not only in regulated sectors)

Interconnectivity

Systems are no longer isolated – they connect with apps, cloud platforms and other devices, increasing interface and integration complexity.

More functionality

More functionality increases complexity because each additional feature introduces new interactions, dependencies and potential points of failure, making the system harder to understand, maintain and manage.

Shorter development cycles

Shorter development cycles create pressure to deliver high-quality project results quickly.

Cybersecurity

In connected environments, systems face constant threats.

Evolving regulations

The project must adapt to changing rules and compliance requirements.

Requirements engineering as part of systems engineering

A defined systems engineering approach as described by INCOSE (International Council on Systems Engineering) is becoming more and more important because the industries are increasingly confronted with complex projects. For companies acting in heavily regulated environments which develop complex smart devices, systems engineering is an effective approach to address their challenges. As a transdisciplinary and integrative approach to enable the successful realisation, use and retirement of engineered systems, it uses systems principles and concepts, as well as scientific, technological and management methods.

The V-model: Demonstration of converting stakeholder needs into capabilities

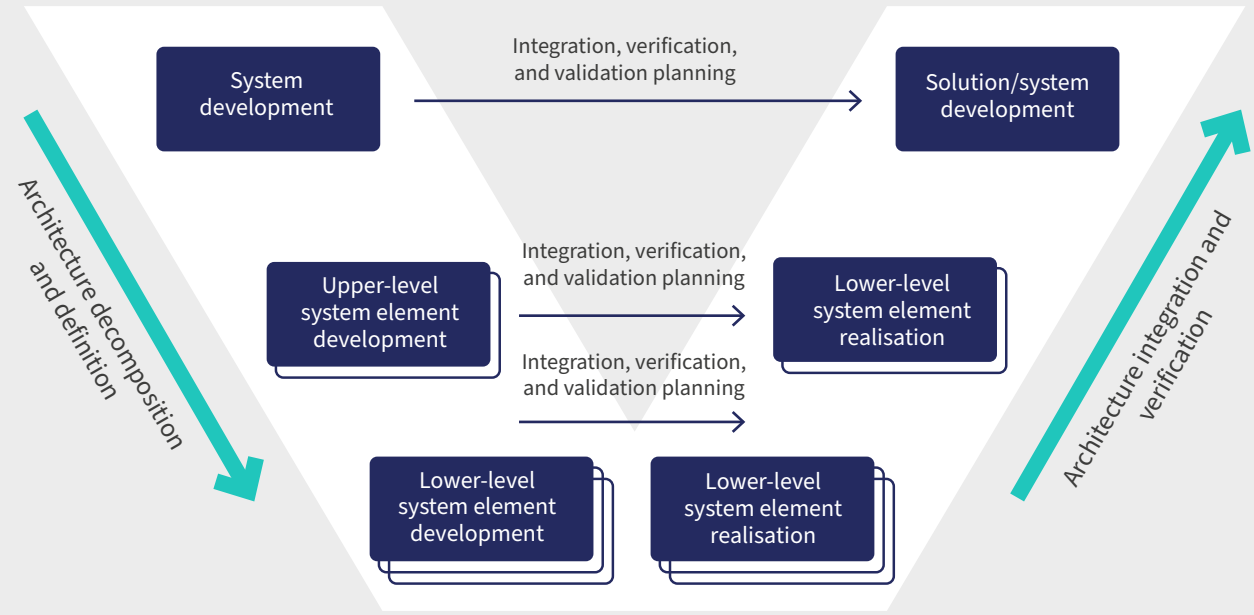
The V-model (or Vee model) in systems engineering is a visual framework that illustrates the systems development lifecycle in a structured, sequential manner. It emphasises the relationship between each development phase on the left side of the 'V' and its corresponding validation or verification activity on the right side.

There are three specific SE technical processes where the requirements engineer is strongly involved:

- Business or mission analysis
- Definition of stakeholder needs and requirements
- Definition of system requirements

These steps define the stakeholder needs and requirements and further convert the stakeholder, user-oriented view of desired capabilities into a technical view of a solution that meets the operational needs of the user. The system requirements become the foundation for architecture, design, implementation and verification. By establishing RE in this segment of the SE lifecycle, we keep the requirements visible, verifiable and aligned throughout the entire lifecycle.

Figure 1: The V-model



Technical processes for:

- Business or mission analysis
- Stakeholder needs and requirements definition
- System requirements definition
- System architecture definition
- Design definition
- System analysis

Technical processes for:

- Operation
- Transition
- Validation
- Verification
- Integration
- Implementation

Adapted from INCOSE SEH, 2023 (Forsberg et al., 2005)

Complexity demands more than knowing the craft

Experienced requirements engineers know the craft and the essentials – all the context, use case and activity diagrams and how to elaborate requirements based on these fundaments. However, the challenge only starts when the requirements turn from static artifacts to living interconnected assets that need management across tools, teams and time. It requires systems thinking, collaboration and adaptability.

ALM tools: A good start, but not the finish line

ALM (Application Lifecycle Management) tools are crucial in regulated domains like medical devices, finance and aerospace because they help manage software development from start to finish, ensuring compliance and quality.

Requirements that were elicited and elaborated out of the requirements engineering artifacts are stored and managed in text form in an ALM tool – which is, of course, already a great start. However, the era of storing requirements in spreadsheets is behind us.

Moreover, requirements engineering artifacts are stored in different documents or different tools, and it is challenging to keep these diagrams updated and aligned during the project and especially also during the operation and maintenance phase. Sometimes, it can happen that tools like Microsoft Visio have been already decommissioned before the project is finished, and thus the diagrams are lost or can no longer be maintained. And in which project does one not wish for a single source of truth?

Design traceability and impact awareness – A missing link

Another big challenge in complex projects arises if the Requirements Engineers do not know which system functions or components are impacted by their requirements. Or, the case may be that they do know, but it is not properly documented and transparent for the project. Each requirements engineer should be able to track the key requirements against the design (know the delta). Therefore, in addition to traces being maintained between requirements and from requirements to test cases, the connection between requirements and their associated design and architecture elements needs to be maintained as well.

Our experience has shown that requirements engineering is not only a key component of the systems engineering process. RE itself also benefits from the systems engineering approach, especially when it comes to Model-Based Systems Engineering (MBSE).

MBSE arrives on the scene

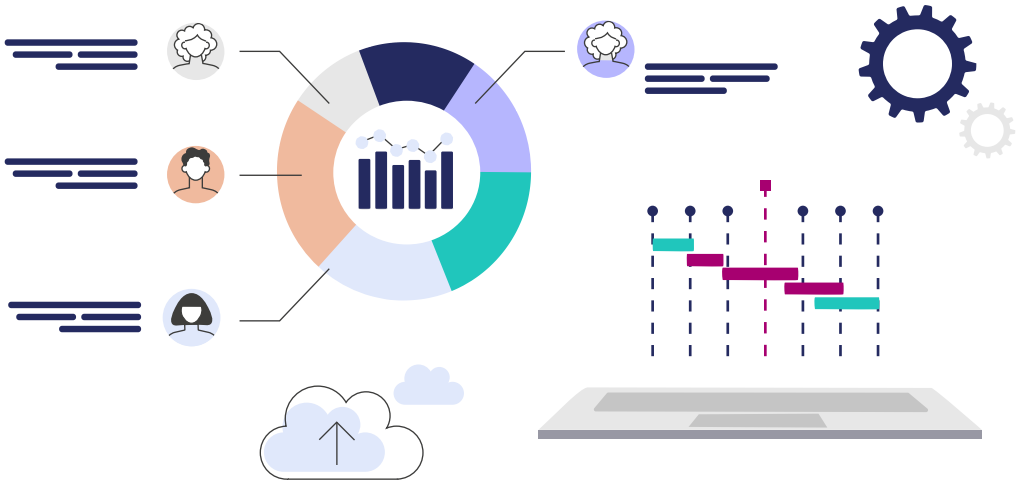
Model-Based Systems Engineering (MBSE) is a standardised methodology used to facilitate requirements, design, analysis, verification and validation in regard to the development of complex systems. Unlike document-centric engineering, MBSE centres around models of system design. The increase in digital modelling environments within the industry over the last couple of years has directly impacted the pace of the MBSE uptake. In January 2020, NASA observed this trend and reported that MBSE "has been increasingly embraced by both industry and government as a means to keep track of system complexity." As far as the methodology is concerned, MBSE represents a collection of related processes, methods and tools.

The INCOSE Systems Engineering Vision 2020 (2007) defines MBSE (Model-Based Systems Engineering) as:

The formalised application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the concept stage and continuing throughout development and later life cycle stages.

From a requirements engineering point of view, it is important to contribute to the model by creating artifacts like activity and sequence diagrams and – of course – elaborated requirements. As soon as the system design and architecture are available, the requirements can be traced to the corresponding elements of the model. This way, a relationship between the text-based requirements and the model elements is established and maintained.

Model-Based Systems Engineering (MBSE) plays an important role because it enhances clarity and communication among stakeholders through visual representations, facilitates early validation and verification of requirements, and supports better risk management by allowing for simulation and analysis of complex systems. MBSE promotes traceability, ensuring that all system elements are aligned with requirements, and enables iterative development, making it easier to adapt to changes. Additionally, it improves collaboration among cross-functional teams and provides a structured approach to managing complexity, ultimately leading to more efficient and successful system development.



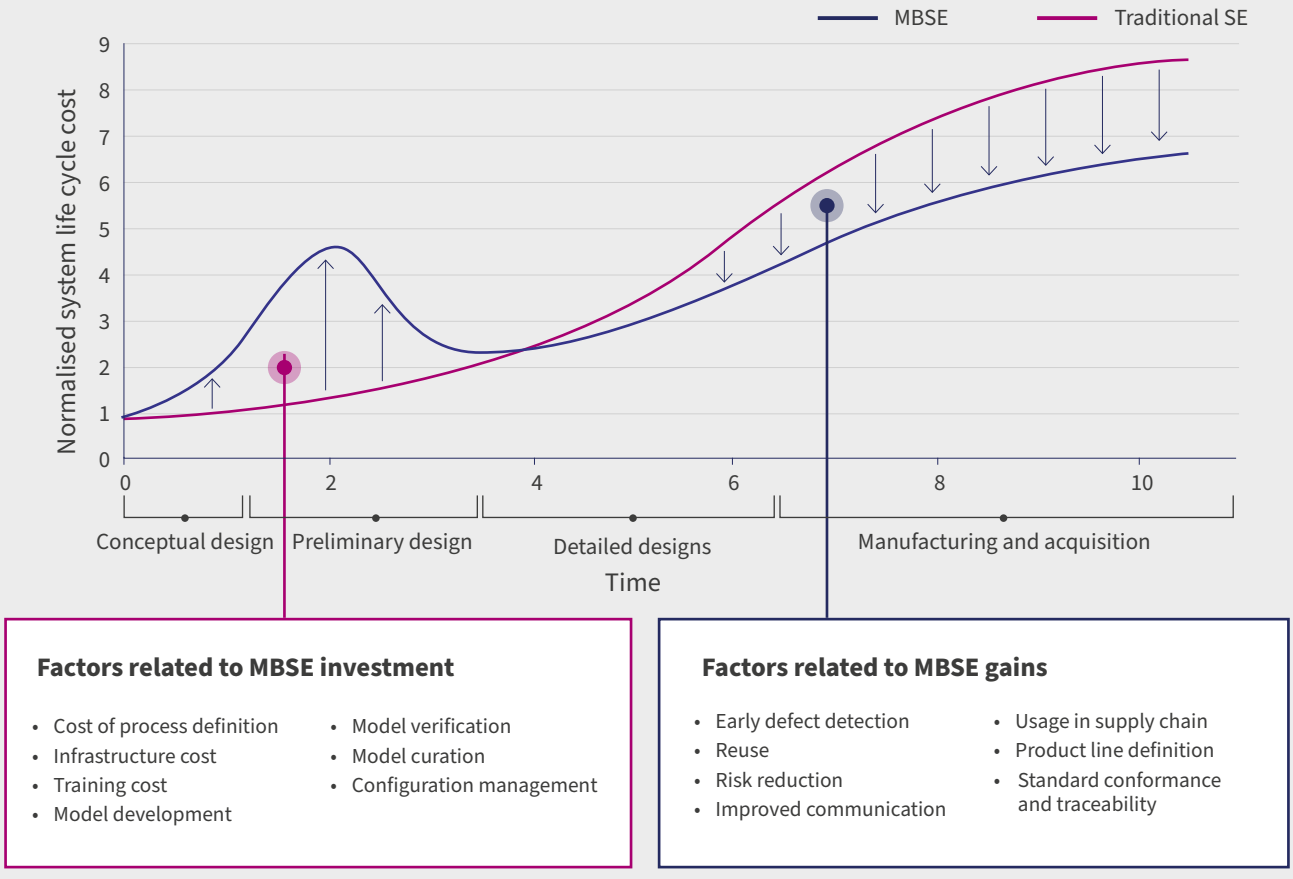
Cost-benefit relationship

Implementing MBSE requires bigger upfront investments compared to traditional approaches. Time and resources are needed early in the lifecycle – for tool integration, modelling, training, and adapting workflows. However, these initial efforts are strategic investments. Projects that adopt MBSE realise significant returns in the later stages of development, especially during verification, integration and compliance. Understanding this relationship is key. Analysing both the cost drivers of early MBSE adoption and the value levers in later phases helps build a strong business case for change. Common early investments include process alignment, tool configuration and model creation; downstream gains include faster traceability, reduced rework, easier change management and smoother regulatory audits. When viewed across the full lifecycle, the economics of MBSE clearly favour long-term efficiency and product quality.

As soon as change management is required and a function needs to be changed, the requirements engineer can easily identify the corresponding requirements. Conversely, when a requirement changes, we can see which functions or parts of the model are impacted. All diagrams are part of the model representing a single source of truth. This applies not only to the development phase but also to operation and maintenance, when documentation becomes even more important.

MBSE offers a range of significant advantages. It ensures that the model is inherently consistent, which supports full traceability through semantic relationships. This enables clear links between requirements, design elements and system behaviour. MBSE also supports high levels of reuse, improving efficiency and reducing error rates across the development lifecycle. Moreover, Requirements Engineers can derive requirements systematically from the model itself, providing grounds for automated document generation.

Figure 2: Key factors related to MBSE investments and gains



*Source: Azad M. Madni * and Shatad Purohit, 2019, researchgate.net*

Practical case: A complex diagnostic system

In our wide experience in the MedTech area, we once accompanied the development of a fully automated diagnostic system capable of measuring different parameters – all in a compact form factor.

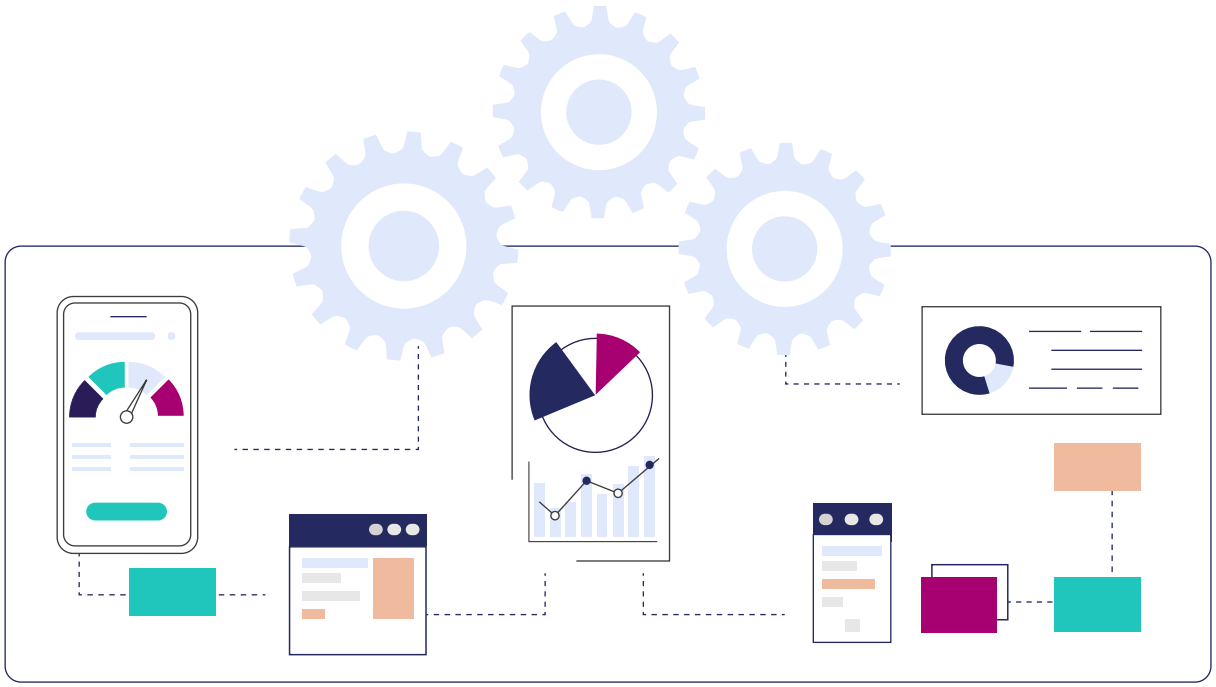
In such a system, several factors play a crucial role:

- Ambitious product vision and scope
- Market competitiveness and cost sensitivity
- Advanced technological development
- Extensive integration of hardware and software
- Rigorous regulatory and compliance framework
- Complex project management

Using an MBSE approach, the development team linked each stakeholder requirement to a system function, component and test case. Changes in one area – meaning a new regulatory requirement for bilirubin measurement – automatically propagated through the model, revealing the impact on architecture and verification plans. This traceability ensured confidence in compliance, minimised late-stage rework and supported faster documentation generation for regulatory submissions.

Conclusion

Integrating systems engineering and MBSE does not only mean using new tools or methods. It means making a company-wide shift in how the organisation thinks and works across disciplines. Making that step requires both technical expertise and knowledge of change management. Having an experienced partner by your side helps in figuring out complex projects both in regulated and non-regulated areas. The difference does not simply lie in implementing frameworks, but in tailoring them to the industry-specific context and internal capabilities, and ensuring that every step on the way is aligned with the overall strategic goals.



About ERNI

ERNI stands for Swiss Software Engineering. What are we really interested in? How we can support you and your employees better than any other company in developing and marketing software-based products and services. Our global platform for software development, in combination with a sound understanding of the market, forms the framework for our customers' success. Our team also implements complex projects, empowers people and delivers outstanding customer solutions in the shortest time. We apply the Swiss mentality with behaviours such as consensus building, pragmatism, integration, reliability and transparency on a global scale – and have done so since our foundation in 1994 together with our great team, which is the basis for successful software projects. Today, the ERNI Group employs more than 800 people worldwide.

About .experience

In this magazine, which is published a couple of times per year by ERNI, we provide information about important learning experiences that we have had in our daily work in the areas of collaboration, processes and technology.

Imprint

Issue 1/2025

ERNI
Swiss Software Engineering
betterask.erni

Publisher
ERNI Management Services AG

ERNI Locations

ERNI Schweiz AG
Bern
Zurich
Lucerne
Lausanne
Basel

ERNI Consulting España S.L.U.
Barcelona
Madrid
Sant C. del Vallès

ERNI Development Center Spain, S.L.
Valencia

ERNI (Germany) GmbH
Frankfurt
Munich
Berlin
Stuttgart

ERNI Development Centre Philippines Inc.
Manila

ERNI Development Centre Romania S.R.L.
Cluj-Napoca

ERNI Singapore Pte Ltd.
Singapore

ERNI (Slovakia) s.r.o.
Bratislava

ERNI USA
New York

Contact

ERNI Management Services AG
Löwenstrasse 11 I 8001 Zurich
Email: marketing@betterask.erni
Phone: +41 58 268 12 00
Web: www.betterask.erni

ERNI on the social networks