Communication in IT
"I Thought It Was Obvious"

Author:
Tomasz Namyslo
Date:
01.02.2026
Link:
https://tomecki.studio/communication-in-it/
Polish/Wersja polska:
https://tomecki.studio/pl/communication-in-it/

I would like to encourage you to reflect on the serious consequences of communication errors. I hope this article gives you practical, usable advice—rather than yet another collection of academic theories. The article examines the issue from the viewpoint of someone with extensive practical experience of projects, in both commissioning and management roles, as well as execution. This involves larger teams, including international ones, working on complex tasks, as well as smaller groups carrying out simple tasks. My aim is to make readers aware of where mistakes really originate and how to minimise them. I will also try to draw a few analogies to help you understand these more complex issues. I will use simple, real-life examples of mistakes that resulted in specific losses that could easily have been avoided — I will explain how. You will also find interesting facts here, such as how a certain plane crash contributed to improvements in many areas of life.

Remember, communication mistakes aren't harmless—they're expensive.

Table of contents

Introduction

It is widely believed that communication is a 'soft' skill, an addition to technical competencies [1] [2] [3] [4] [5] [6]. In this article, I will demonstrate why this approach may be a fundamental misjudgement in risk assessment. Communication is not a soft skill, but a critical operational function. Failure in this area can result in real, measurable costs, ranging from lost productivity to catastrophic security breaches. We are faced with a paradox: despite having dozens of collaboration tools at our disposal, ranging from Slack to Jira, our teams are experiencing an increasing amount of knowledge fragmentation, cognitive overload and chronic lack of context [7] [8]. The first step towards building a truly efficient and resilient organisation is to understand the hidden psychological and structural forces that quietly sabotage the flow of information [9] [10] [11] [12].

1. Why do we misjudge the effectiveness of our communication?

Fundamental communication errors do not originate in Slack channels or email content. They originate in our minds long before a word is spoken [9] [10] [11] [13]. These errors are rooted in innate cognitive biases that cause us to overestimate the clarity of our intentions and messages. For leaders who want to diagnose and solve problems in their teams at the source, understanding these universal mechanisms is crucial.

1.1. The mistaken assumption that others understand our intentions

1.1.1. The illusion of transparency

The psychological phenomenon known as the 'transparency illusion' describes our tendency to overestimate how visible our internal states — thoughts, intentions, and emotions — are to others [9]. A classic experiment illustrates this perfectly: participants were asked to tap out the melody of a well-known song on a tabletop [9]. Those tapping were convinced that the listeners would easily guess the song because the melody sounded complete in their heads. However, in reality, the listeners, hearing only a series of contextless taps, were able to identify only a few percent of the songs.

This illusion is evident in the workplace every day. In the context of a development team, for example, the illusion of transparency can manifest itself in many destructive ways:

- **Developer:** He assumes that his serious concerns about the complex technical solution are obvious to the rest of the team, even though he only hinted at them during the meeting. He can hear the alarm bells ringing loudly in his head, but outwardly, only a quiet whisper can be heard... a whisper that, after a few sprints (sprint = time period), turns into a scream during costly refactoring (code rewriting).

- **Product Owner:** He is convinced that he communicated his enthusiasm and vision for the new functionality to the developers clearly and effectively during a brief meeting. He assumes that everyone shares his perspective, unaware that, for the team, this was just one piece of information among many that day. Without clarification, this information leads to the implementation of a feature that is inconsistent with the business vision.
- **Team Leader:** He believes that his empathy and good intentions are clear when he provides difficult feedback. However, the recipient, lacking insight into his internal states, can only perceive dry criticism and an attack, which undermines trust and closes the door to honest conversation in the future.

Analogy: The illusion of transparency can be compared to speaking in a noisy room. We think we are shouting, but to someone standing a metre away, we are a barely audible whisper amid the general noise. The mistake lies in assuming that what sounds so loud in our heads must be heard just as clearly by everyone around us.

1.1.2. Jargon

Jargon is a paradoxical tool. Although it is created as a precise intellectual shortcut for experts in a narrow field, it becomes impenetrable when used in a broader organisational environment. Overusing specialised language is a manifestation of the 'curse of knowledge' – a cognitive bias that prevents us from seeing a problem from the perspective of someone without our specialist knowledge [14] [15] [16].

Research from the Journal of Language and Social Psychology [17] shows that jargon hinders understanding and actively discourages and alienates audiences, reducing their engagement. An analogy from an article in Venture in Security [3] illustrates this perfectly, comparing technical specialists to doctors. The author notes that, despite medicine being an extremely complex field, patients visiting a doctor's office expect explanations in plain language, not Latin. The modern role of an expert, whether a doctor or a security engineer, is evolving towards that of a coach and advisor who presents various options and their implications but leaves the final, informed decision to the patient (or business leader). From this perspective, business decision-makers do not require technical details about software families; rather, they need an understanding of how specific risks could impact their revenue, reputation, or business continuity.

1.1.3. The tool paradox

Modern companies are overwhelmed with collaboration tools, which has counterintuitive negative effects [18]. Employees now spend 50% more time on team activities [19] , and, according to research cited by Lokalise, they switch between applications an average of 33 times a day [8]. This phenomenon generates two hidden costs:

- Tool overload: The stress and frustration that result from having to use too many applications, many of which are redundant.
- Context switching: Each time a user switches between Slack, Jira and their email inbox, it generates cognitive costs that distract them and destroy their ability to work deeply.

The fragmentation of information means that work resembles 'archaeology' – more time is spent searching for scattered context than creating value. The findings of the Lokalise report are alarming: 60% of employees say that tool fatigue negatively affects their ability to collaborate, and 36% point to its harmful impact on mental health [8].

1.2. Psychological safety

1.2.1. What is psychological safety?

Psychological safety is defined as a shared belief among team members that they can take interpersonal risks, such as asking questions, admitting mistakes, offering new ideas, or questioning decisions, without fear of humiliation, ignorance, or punishment. While not synonymous with friendship, it is a prerequisite for open, honest and effective communication, which is essential for the early detection and repayment of technical debt. Chronic stress and pressure lead to reduced cooperation due to the activation of natural defence mechanisms. Activities that require trust and openness, such as pair programming or honest code reviews, become less common. Instead, there is a growing tendency to create information silos – isolated 'islands' of knowledge and information – that prevent the free flow of information between teams. This leads to knowledge hoarding because, in a dangerous environment, information becomes a currency that protects the individual's position.

A lack of psychological security is one of the main causes of stress and burnout in the IT industry. The data is alarming: up to 80% of developers experience burnout, and 23% to 42% of their time is spent working on low-quality code resulting from technical debt [20]. This creates a vicious circle: technical debt generates stress, and stressed developers create more technical debt [21].

1.2.2. Organizational culture and information flow

Sociologist Ron Westrum has created a model describing how information flows through an organisation depending on its culture [22]. His typology helps to understand why some organisations sweep problems under the rug while others quickly disclose them and use them as an opportunity for learning and improvement.

- **Pathological culture (power-oriented):** Information is kept secret and treated as a source of personal power. Those who deliver bad news are symbolically 'shot'. Mistakes lead to the search for culprits and punishment.
- **Bureaucratic culture (rule-oriented):** Information often flows through formal channels but is ignored. Mistakes can lead to new rules and procedures being created.
- **Generative culture (performance-oriented):** Information is actively sought. Errors are analysed to identify areas for improvement. Cooperation is highly valued.

1.3. System architecture as a reflection of communication structure

In 1968, programmer Melvin Conway made an observation which would become known as the 'Conway's Law' of software engineering: 'Organisations design systems that mirror their communication structures' [23] [24]. This means that the boundaries between teams and departments are inevitably reflected in the software's interfaces and modules. Companies such as Netflix and Amazon naturally create decentralised systems based on microservices because they have small, autonomous teams. Each team is responsible for its own part of the system and communicates with the others via clearly defined APIs. Conversely, communication between departments is often slow and formalised, resulting in suboptimal interfaces, integration problems, and enormous architectural debt [25] [21]. In response to this problem, a conscious strategy known as the 'Reverse Conway Manoeuvre' has emerged [26] [27]. This strategy involves deliberately reorganising the structure of teams to enforce the desired software architecture, rather than passively accepting the architecture imposed by the existing structure.

Analogy: The structure of an organisation is similar to the layout of pipes in a building. If the plumbers (backend) and electricians (frontend) work in separate buildings and rarely communicate, the pipes and cables in a new house will likely cross in illogical places and require complex workarounds to connect. Conversely, the reverse Conway's Law involves placing all these specialists in one room around a specific floor plan (microservice), enabling the pipes and cables to be arranged in the simplest and most efficient way.

1.4. Unclear and changing requirements

Scientific research and experience both indicate that changes to requirements significantly increase the risk of delays and architectural instability. Dynamic, disorganised projects demonstrate that sudden changes introduced at a late stage have a disproportionate impact on costs [28] [29]. Problems usually do not start with poorly executed work, but with vague concepts ('fast', 'easy', 'scalable') and divergent definitions of readiness (Definition of Ready/Definition of Done). Without explicit definitions and examples, these concepts recur in tests and UAT [30]. Additionally, changing priorities do not permit an unlimited number of concept changes per sprint — constant rearrangement without closing the decision loop escalates chaos and resistance [31]. RE (Requirements Engineering) literature shows that ambiguity in natural language specifications is common and that tools for automatically detecting it are still limited and unstable. Structuring discussions about scope and priorities, and creating 'boundary objects', is a valuable investment in achieving predictability and consistency of understanding [32] [33].

1.5. Time pressure and budget

Although literature reviews confirm that time pressure can stimulate short-term productivity, it usually reduces software quality and worsens team well-being. This leads to the phenomenon of 'buying' speed at the expense of technical debt and errors [34]. Experiments demonstrate that reducing deadlines increases developers' propensity to take risky shortcuts. This can be prevented by using 'hot mediation', whereby political pressure is translated into real compromises regarding the scope and order of tasks. When functionality is added to a fixed budget — the business's response to the market — teams perceive this as a forced reduction in quality. Research on requirements volatility proves that a lack of clear compromise in this area destabilises the architecture and generates delays. Therefore, an effective mediator can separate discussions about value ('what') from technical implementation ('how/when') to reach a compromise and avoid the chaos resulting from ambiguous priorities.

1.6. Coordination and conflicts

Research by Herbsleb and Mockus on software engineering shows that tasks carried out by geographically dispersed teams take approximately 2.5 times longer than similar tasks performed by teams in a single location. This is due to the need to consult more people when solving problems, as well as the fact that communication is significantly less frequent and less effective [35]. In terms of internal conflicts, Bacchelli and Bird demonstrate that modern code review is a vital tool for knowledge transfer and fostering team awareness [36]. In practice, however, discussions often focus on style and formatting rather than significant defects. The biggest challenge for reviewers is often understanding the context of the change and the author's intentions. Neutral moderation helps close threads and shorten the question-and-answer cycle, whether it's due to delays caused by distraction or disputes over code review standards. Clear communication rules and acceptance criteria are also essential. These are investments that reduce turnaround time and improve coordination, thereby limiting costly misunderstandings resulting from a lack of shared context.

1.7. Summary: Classic communication barriers in IT

In addition to Conway's Law, development teams encounter typical communication challenges that are specific to the IT environment.

Barrier	**Implications for the development team**
Filtering information	The manager conceals genuine issues relating to the budget or deadlines from the team, fearing an adverse response. The team, unaware of the full context, makes suboptimal technical decisions in the long run; for instance, they might choose a solution that is faster to implement but more expensive.
Information overload	Developers are bombarded with hundreds of notifications from Jira, Slack and emails. Key information about changes to requirements can get lost in the noise, resulting in outdated features being implemented. This is such a common phenomenon that many employees simply stop reading notifications.
Technical jargon	Using specialised language outside a narrow group of experts creates a barrier and a sense of exclusion. In the context of the relationship between business and IT, for example, this can lead to fundamental misunderstandings about project goals and requirements.
Cultural differences	In international teams, communication style is of the utmost importance. In low-context cultures, such as Germany and the USA, directness and precision are valued. In high-context cultures, such as Japan and Arab countries, relationships are key and communication is often veiled. For example, direct feedback from a German leader may be perceived by a Japanese programmer as a brutal attack.

2. The negative effects of ineffective communication

2.1. Business consequences: From human error to financial losses

A lack of precise communication and shared understanding of knowledge (Grounding) can lead to significant financial losses. For example, the cost of rectifying an error detected in the operational phase can be 29 to 1,500 times higher than at the requirements definition stage [37]. Statistics confirm that human error is a factor in 95% of data security breaches [38] [39] , and teams that rely on the illusion of mutual understanding are vulnerable to manipulation. A dramatic example of this is the incident at Ubiquiti Networks, where $46.7 million was lost as a result of a social engineering attack [40] Posing as senior management, cybercriminals exploited employee trust and time pressure to order a series of unauthorised transfers. Employees acted under the illusion that the sender's intentions were genuine, which led to financial disaster. This demonstrates how cognitive errors in communication can be exploited to attack the entire organisation.

2.2. Technical debt

Technical debt is a powerful metaphor that perfectly illustrates the consequences of compromises made during the software development process. Borrowed from the world of finance, it describes the situation in which we choose a quick but suboptimal solution to meet a deadline, either consciously or unconsciously. In doing so, we take out a 'loan' from the future. This loan must be repaid with 'interest' in the form of the extra work required to repair, refactor or rebuild the system [41]. Gartner defines technical debt as 'the work we owe to the IT system' [42]. If ignored, it accumulates until it eventually leads to the project becoming developmentally paralysed and technologically bankrupt.

Technical debt is not a uniform phenomenon. It can be divided into several key categories, each of which has different causes and consequences:

- **Code debt:** This is the most obvious form of debt. It encompasses practices such as the use of 'workarounds', a lack of unit and integration tests, code duplication and the disregard of coding standards.
- **Architectural debt:** It is much more serious and difficult to repay. It arises from poor design decisions made early on, which lead to suboptimal component integration or the selection of outdated technology.
- **Documentation debt:** This is the result of time pressure, where documentation is treated as a low-priority task. This results in outdated, incomplete or non-existent documentation, which makes it very difficult to onboard new team members and maintain the system.
- **Process debt:** This applies to inefficient or outdated delivery processes, such as manual, time-consuming implementations; a lack of test automation; and complicated, unclear change approval procedures.

2.3. Irreversible consequences

The tragic collision of two Boeing 747s in Tenerife in 1977, in which 583 people died, is a tragic example of how a series of minor misunderstandings, exacerbated by time pressure and a rigid hierarchy, can lead to disaster [43]. The accident was not the result of a single technical error, but rather a combination of human and communication factors. In thick fog and with overloaded radio communications, a series of misunderstandings occurred: the KLM crew began take-off without receiving clearance while the Pan Am aircraft was still on the runway. Time pressure played a key role, as did the fact that the KLM captain was also the chief pilot instructor. This reinforced the hierarchy in the cockpit, making it difficult to challenge decisions effectively at a critical moment. This incident taught us a valuable lesson, revolutionising aviation and forcing the introduction of standards for unambiguous communication and procedures for mutual confirmation of understanding. This clearly shows that precision of language and verification of intent are fundamental safety mechanisms.

3. How to improve communication

3.1. Building common ground

Effective communication involves more than simply exchanging information; it requires an active process of establishing 'common ground' to ensure that the message has been not only received, but also incorporated into the team's shared knowledge. In software engineering practice, this process is often disrupted by the phenomenon of 'thin domain knowledge', which was identified by Curtis in his study of 17 large systems. In this study, it was found that a superficial understanding of business objectives among developers led to misinterpretations of requirements [12]. This competence deficit often resulted in key architectural decisions being centralised in the hands of a narrow coalition of 'design experts'. While this allowed for rapid progress, it prevented the exploration of alternative solutions and increased the risk of perpetuating errors in the system's foundations.

Analogy: A lack of shared domain knowledge within a programming team is comparable to building a modern hospital with a construction crew that has no understanding of how healthcare works. Even if the builders (developers) are perfectly capable of laying bricks (writing code), without an understanding of patient flow and the specifics of staff work (domain knowledge), they may place the operating room too far from the emergency ward. The result would be a system that is technically correct but functionally flawed and extremely expensive to fix later on.

3.2. Building an effective communication ecosystem

Transitioning from diagnosis to action demands a strategic plan. Rather than offering vague advice, the following recommendations focus on three pillars of transformation: shaping a conscious culture, engineering transparent processes and designing a clear technology stack.

Pillar I: From individual to generative responsibility

In order to neutralise the 'illusion of transparency' and build a foundation that is resistant to social engineering attacks, leaders must systematically cultivate a generative culture. Creating psychological safety starts with them. Key daily actions that model the desired behaviours include:

- **Treat work as an opportunity to learn, rather than a problem to solve:** Emphasise that every project is an experiment and that the goal is to learn and adapt as well as deliver results. This changes the focus from the pressure to perform to a sense of curiosity and discovery.

- **Recognise your own fallibility:** Admit your mistakes, lack of knowledge and uncertainty openly. By asking questions such as "What do you think about this?" or "I may be wrong, but...", a leader creates an environment in which others feel comfortable expressing their own uncertainties.
- **Model curiosity and ask questions:** Actively seek feedback and input from the whole team to show that you value their perspective. Asking questions such as "What am I not seeing?" and "What other options do we have?" encourages discussion and challenges the status quo.

Pillar II: Engineering transparency in software development

The solution to the 'Language as a Wall' problem and the resulting chaos from the tool paradox is to engineer transparent processes that enforce consolidation and create a common context. Defining tasks with precision drastically reduces the cognitive load on developers and eliminates unnecessary effort. Implementing structured communication rituals is essential:

- **The '3 Amigos' sessions (Product Owner, Developer, Tester) [44] [45]:** Short meetings are held to develop a common understanding of requirements and acceptance criteria before implementation begins. This is a mechanism for establishing best practice.
- **Enforcing 'Definition of Ready' (DoR):** The development team should be able to reject tasks that are unclear or incomplete. The DoR acts as a contract that protects engineers' time.
- **Standardisation of 'Definition of Done' (DoD):** Having a clear, shared definition of what 'done' means ensures that every task is fully tested and integrated before deployment.
- **Useful ticket [5] [46]:** It contains the full context (why are we doing this?), clear acceptance criteria (what needs to be done?) and links to resources (Figma designs, documentation), to avoid 'information archaeology'.
- **Example Mapping:** This is a technique in which the team uses coloured sticky notes to visualise business rules, how they work, and any questions or ambiguities. This enables any gaps in the specification to be identified quickly.

Pillar III: Conscious architecture of cooperation

Since Conway's Law states that our architecture reflects communication, consciously designing the technology stack becomes a proactive way of shaping effective and desirable information flow paths. The aim is not to eliminate tools, but to integrate them in a considered manner.

- **Conduct a technology stack audit:** Identify any redundant tools performing the same function, and locate any information silos.

- **Establish clear usage rules:** Define the purpose of each tool, e.g. use Slack for quick, informal questions; use email for formal, external communication; and use Jira for tracking work.
- **Apply the 'Reverse Conway Manoeuvre':** Organise teams around the desired product architecture. This will ensure that their natural communication paths support, rather than hinder, technological goals.

3.3. Mediator

3.3.1 The role of mediator

A mediator's role in a team is to act as a neutral interpreter between business and technology. They use boundary objects [47], such as architectural models or requirements specifications, to synchronise meanings without imposing a uniform perspective. This is crucial for effective coordination in agile environments. Rather than asking 'who is right', the mediator helps to determine what decision needs to be made and what its architectural, cost and operational implications are. As a moderator of critical discussions, the mediator ensures psychological safety, which is a prerequisite for task-related conflict to improve decision quality rather than diminish team performance [48] [49] [50]. When faced with time pressure, which often results in quality being sacrificed for speed, the mediator creates realistic schedules by formalising compromises on scope and priorities, thereby protecting the team from decision-making chaos. The mediator pays close attention to organisational knowledge, promoting a culture of learning (for example, a blameless approach) and 'freezing' agreements in the form of explicit standards, such as DoR and DoD. This ensures transparency in the process and prevents the recurrence of issues that have already been resolved. This has a measurable effect: shorter decision lead times, less rework and fewer interruptions to work. Global Software Development shows that a significant portion of the 'distraction tax' is paid in coordination and communication costs [35] — and this is where mediation cuts losses. It clarifies expectations and definitions more quickly, limits the escalation of conflicts, minimises the cost of changing requirements through explicit compromises and protects quality under time pressure by negotiating scope instead of 'squeezing' the team. In summary, hours lost to conflict rarely create value, but well-conducted mediation can transform them into progress in the decision-making process. This approach has been shown to statistically reduce the risk of delays and costs while supporting quality and morale.

Analogy: Working with boundary artefacts is similar to how children and engineers use LEGO bricks. For children, the bricks are a toy (local perspective), whereas for engineers, they are a system of geometric connections (technical perspective). However, because the bricks themselves have a fixed shape and standard protrusions (boundary artefact), both groups can build a castle model together, even though the castle 'means' something different to each of them. The bricks enable them to collaborate without the child having to become an engineer.

3.3.2. Examples of IT mediation scenarios

Scenario: E-commerce system failure.

Situation: The e-commerce platform has stopped working, resulting in direct financial losses for a customer. There is conflict and finger-pointing within the team: the external supplier claims there is an error in the application, while the internal team blames the partner's infrastructure.

The role of mediator: The mediator takes control of the crisis situation by implementing Site Reliability Engineering procedures, beginning with the establishment of a 'blameless war room' – a space focused on facts and the chronology of events, rather than the search for culprits. The mediator's task is to analytically separate the root cause of the failure (Root Cause Analysis) from contributing factors that only increased its scale or duration. This allows the team to redirect their energy from defending their positions to solving the systemic problem. During the decision-making phase, the mediator prioritises remedial actions (Remediations) and ensures that key architectural changes are 'frozen' in the form of ADRs (Architecture Decision Records). These documents synchronise technical and business understanding. The mediator concludes the entire process with a post-incident analysis (Blameless Postmortem) to develop joint conclusions and implement preventive mechanisms for the future.

Why does it work: A non-blaming culture and psychological safety form the foundation of organisational learning. When a team is confident that admitting a mistake will not result in punishment, they are more likely to report problems, which is crucial for rapid diagnosis and repair. This approach, adopted from aviation and medicine and implemented in IT by Site Reliability Engineering (SRE), transforms failure from destructive conflict into a valuable learning opportunity, thereby increasing organisational resilience and preventing the same mistakes from being repeated in future.

Scenario: Delays and deadline pressure

Situation: A client delays the delivery of key materials or approvals while refusing to postpone the final implementation date. In an attempt to make up for lost time, the development team works overtime, which increases quality debt and the risk of burnout.

The role of mediator: The mediator steps in to transform emotional pressure into a structured, fact-based negotiation process. This begins with precisely mapping dependencies to identify which critical paths are blocked by the client, and where alternative work can be carried out in parallel. They then negotiate compromises, forcing the business to choose between reducing the scope or postponing the deadline, while ensuring the quality of key functionalities by enforcing non-negotiable technical standards. To protect the team from chaos, the mediator enforces process discipline by introducing hard limits on work in progress and rigorously applying the 'Definition of Ready', which prevents work from starting on insufficiently defined tasks. This process is complemented by the implementation of protective measures such as quality gates, which prevent the promotion of code that does not meet minimum testing requirements, regardless of external pressure.

Why does it work: Although time pressure can increase work speed in the short term, this comes at the expense of quality and leads to more errors. Mediation shifts the focus of discussions from aspirational to transactional, making stakeholders aware that inflexibility regarding input delays must be balanced with a change in scope, in order to avoid a decline in quality and the accrual of technical debt.

4. Case studies of misunderstandings

Case: 'Add search' – how misunderstandings arise.

The team is developing a product catalogue containing over 500 items. The product manager (PM) has noticed a drop in conversion rates; users scroll through the list, get lost, and give up. Someone suggests adding a search function.

Ticket in Jira (bad):
Title: Add a search function to the product catalogue.
Description: It would be useful to have a product search tool.
Deadline: Do it ASAP.

How does misunderstanding arise?
As the ticket does not describe in detail the type of search we expect, people automatically fill in the missing elements.

Developer's interpretation:
"Due to the large amount of data, I will perform the search on the server side."
"The simplest and most common option is to search by name, which is enough to get started."
"As I'm not a graphic designer and there are no mock-ups or UI designs, I'll create a simple, standard view consisting of a text field and a list of results, in line with the existing style of the website."

So, the developer implements:
- /api/search?query=… endpoint
- search by name field
- UI: search field + list of results (simple design based on the appearance of the website)

The PM's interpretation (which is equally obvious, but only to them):
- Search by name or description;
- With typo handling (fuzzy);
- Diacritical marks;
- Logical UX for mobile and desktop;
- The UI 'bells and whistles' that he believes are integral to a 'good search tool' include highlighting matches, autocomplete suggestions and hints, 'empty state' designs, tips/suggestions (most searched), nice animations and feedback when loading.

The problem:
The ticket lacks a description and additional materials. Basic information on how the search

function should look and work is also missing. As the graphic design has not been provided, the developer cannot guess the intended user experience (UX) and should not take on the role of a designer.

The moment of delivery:
Two days later, the developer delivers the 'search' feature. It works on the server side, is quick and 'production-ready', and the UI is correct, though minimalistic and 'technical' as there were no mock-ups.

PM tests and says:
"OK, but if the word is only in the description, why can't it find products?"
"Why doesn't 'sheos' (shoes) return anything?"
"The list of results on mobile devices looks different to what we expected."
"Why does it look so plain? It was supposed to have hints and highlights, as well as a cool empty state…"

Dev replies:
"The ticket only said 'Add a search tool'. I performed a standard server-side search based on the product name."
"As there were no mock-ups or UX requirements and I'm not a graphic designer, I created a simple default view to ensure the necessary functionality was delivered."

This is where the conflict begins:
PM: "This does not solve the user's problem, though."
Dev: "I couldn't have known that it was meant to be something more."

Real and painful consequences:
1) Time wasted and work interrupted – requirements were only clarified after implementation. "Let's grab 15 minutes", "Who remembers what it was about?", "Send links to the agreements". This distracted people from their current tasks.
2) Rework: extend the search logic to include 'Description', agree on the ranking of results, add scenarios for 'No results', 'Typos' and 'Diacritical marks', adjust the UI/UX, organise or redesign the graphic/UX layer, etc.
3) Frustration and low morale: the developer feels that they are 'delivering and getting criticised', the project manager feels that 'the team is not delivering value', and the team loses confidence in tickets, resulting in increased caution and a slower pace of work.
4) Communication debt: hidden work begins. This involves searching through Slack and emails, asking "What did the author mean?", carrying out 'mini-analyses' before each task and holding more meetings while producing less code.

Why did this happen:
- The developer does not know whether this is a UX improvement or a critical element of conversion due to a lack of business context (‘why?’).
- There is no goal or definition of 'done', so it is unclear what the measurable outcome should be.
- There is no scope — 'search' can mean ten different things.
- There are no acceptance criteria — there are no narrative tests that can instantly highlight differences in interpretation.

- There is no UI/UX design. The ticket does not specify whether a 'minimal functionality' or a 'refined experience' is expected.

Here's how to safeguard against this:
The ticket will not be processed if any of the following information is missing: business context, objective (what needs to be improved for the user), scope, acceptance criteria (at least two to three scenarios), Definition of Ready, mock-ups/UX decisions, links to resources.

Even when reading this example, it is possible to interpret the ticket differently. The problem is not that someone was incompetent or malicious. The problem is that language without context can be ambiguous. Take the title, 'Add search feature to the product catalogue'. Even after reading it, different people may imagine different things (quite sensibly!):
Reader A (UI/UX):
"A search tool is primarily about UX: suggestions, highlights, empty states, animations — without mock-ups, it is impossible to do it well."
-> This is also a reasonable expectation when considering the quality of the experience.
Reader B (Tester):
"Without acceptance criteria, I don't know what to test. Are there any hints? What about when there are no results? What does the empty state look like? What are the matching rules?"
-> No 'Definition of Done'.
Reader C (Klient):
"I really dislike unnecessary animations. Above all else, searching should be fast and effective."
-> The PM's vision does not align with actual customer expectations.

This illustrates why 'obvious' requirements are the most costly: until they are written down, they only exist in people's heads, and everyone's idea of what is obvious can differ.

This is what a good ticket should look like:
Title: A product search tool based on UI design
Task description:
We have over 500 products in our catalogue. Analytics and feedback indicate that conversion rates are declining: users scroll through the list, become overwhelmed and abandon the process. The search function is designed to reduce the time it takes to find the right product, thereby decreasing the number of users who abandon the process at the product selection stage.
Objective:
Users should be able to quickly find products by name or description in the catalogue, whether they are using a mobile or desktop device.
KPI (Key Performance Indicators) - after implementation, for verification by PM:
- decrease in the percentage of sessions terminated on the catalogue page,
- an increase in the click-through rate of products after using the search engine,
- reduction in the average time taken for the first click on a product.
Scope:

- search by product name or description,
- support for diacritical marks (e.g. 'café' == 'cafe'),
- basic tolerance of typos (fuzzy at the level of 'one or two errors'),
- the UI is based on graphic design (<see link to ticket with graphic design>),
- ranking of results: prefer matches in name > description.

Not applicable (outside the scope of this feature):
- personalisation of results,
- advanced filters,
- 'most searched' and extensive marketing suggestions.

Functional description:
- Search field at the top of the catalogue list.
- Enter at least two characters to start the search.
- Results refresh the product list.
- Clear search → standard list returns.

Technical requirements:
- Backend: API endpoint or better method (to be chosen by dev)
- Search by: product name or description.
- Case-insensitive.
- Tolerance for typos, so: 'sheos' returns 'shoes' and 'cafe' returns 'café'.
- Ranking: name match (highest), description match, shorter matches/greater consistency - higher.
- Performance: response < 200 ms.
- Security: no risk of query modification through transmitted search text.

Acceptance criteria:
A) Name + description:
Given: a product only has the word in its description
When: type this word into the search tool
Then: the product appears in the results.
B) Diacritical marks:
When: type in 'cafe'
Then: finds products with 'café'.
C) Typos:
When: type 'sheos'
Then: the results show 'shoes' (or products with 'shoes' in the name/description).
D) Empty state:
When: no results
Then: a message appears + a button to clear and return to the list.
E) Performance:
A typical query returns results quickly (within the limit / without visible UI lag).

Definition of Done:
The code has been reviewed and tested (with a minimum of unit tests for the search function).
The changes have been implemented in production and the project manager has confirmed acceptance of the criteria. Notes have been added to the release/changelog.

Priority / Deadline:
Priority: High (impact on conversion).

Deadline: e.g. until the end of sprint X (because of the campaign/release).
Assigned:
PM: <name> (acceptance of UX and criteria)
Dev: <name>
QA: <name>

Case: 'You have a 24/7 service with a response time of 15 minutes' → followed by drama at 10.30pm.

An IT company sells a software-as-a-service (SaaS) platform to business-to-business (B2B) customers. One such customer, a large organisation, insists on 'certainty' because the system is crucial to sales.

During a sales conversation:
Customer: "We need 24/7 support, with responses within 15 minutes."
Salesperson: "Yes, we're available 24/7 and we respond within 15 minutes."
That sounds great! Everyone is happy that they managed to reach an agreement. Nobody is asking what 'response', '24/7' and 'failure' mean.

How misunderstanding arises:
Both sides interpret things differently.

What the customer thinks:
'Response within 15 minutes' means that the problem must either be resolved or its functionality restored within 15–30 minutes.
'24/7' means 'we can call at any time and someone will respond immediately'.
'Failure' is defined as 'anything that does not work as it should', such as report export, slow loading or no emails.

What does the IT company think?:
'Response within 15 minutes' refers to the time taken to acknowledge the report or provide an initial response, not the time taken to carry out a repair.
'24/7' means being on call for P0/P1 incidents (help desk levels), but not having a full team available for all issues.
'Failure' means 'system unavailable' or 'critical business path down'; the rest is standard support during business hours.

Everyone thinks their idea of what it means is normal and clear, because no one has given any examples or definitions.

The moment of occurrence:
It's Friday, 10:30 p.m. The customer can't generate a monthly report (the function works, but is interrupted by a long response time).

The customer calls ‘24/7’:
"Our report isn't working — this is critical! You have 15 minutes to respond. Please fix it."

On-call replies after 10 minutes:
"I can confirm that the incident occurred and that we are analysing it. We will provide an update."

The repair process takes two hours and includes diagnosis, service restart and load analysis.

The customer on Monday:
"You promised 15 minutes, but we ended up waiting for two hours. This violates the SLA."
(SLA = Service Level Agreement, a contract that guarantees a specific level and quality of service.)

IT team:
"We responded within ten minutes. The SLA was met."

And so begins a conflict that is not about the system itself, but about words.

Consequences:
- Loss of trust: the customer starts to see promises as just marketing.
- Escalations and politics: emails to directors asking "Who signed this?", and crisis meetings.
- Costs: discounts, free months and additional services to 'keep customers happy'.
- On-call burnout: people feel that they are 'on call for everything'.
- The divide between sales and programmers: sales 'sold' it, and programmers 'cleaned up' after it.

Why did this happen:
- No common definition: response vs. resolution, 24/7 for what, what constitutes an incident, how we measure time.
- No examples: no one said, "Let's give three situations and determine whether it's P1 or P3."
- No handoff: sales did not convey (or did not have) 'how support actually works'.

How to prevent this:
1) SLA Cheat Sheet (for the customer and internally)
Definitions: Response time vs Resolution time
Hours: what does 24/7 mean
Channels: telephone/email/portal + what constitutes an official report
Severity matrix: P0/P1/P2/P3 with examples

2) Rule: 'A promise without definition = risk'
If the phrase '15 minutes' comes up in conversation, clarify it immediately:
"Is it 15 minutes for the first response or to restore service?"
"Does this apply to all tickets or just to P0/P1 tickets?"

3) Pre-sales → delivery handoff, short checklist:
What does the customer mean by 'critical'?
What are the support hours?
What is absolutely P0 (e.g. login, checkout)?
What are the communication expectations, such as updates every 15, 30 or 60 minutes?

4) Summary after the conversation, e.g. by email:
24/7 support applies to P0/P1 incidents.
Response time: P0/P1: up to 15 minutes (24/7), P2/P3: between 9 a.m. and 5 p.m. on working days

Repair time: P0: 2 hours, P1: 6 hours
Examples:
P0: system unavailable / login not working / checkout not working
P1: the key function is significantly limited for most users.
P2: the workaround exists, affects some users
Method of communication during an incident: updates every 30 minutes for P0, every 60 minutes for P1.

There is no 'guilty person' here — it is ambiguity that is to blame. This is why such statements should be accompanied by definitions and examples; otherwise, everyone will interpret them differently. Even when reading this example, a person may automatically interpret it differently:
'Response in 15 minutes' means 'repair in 15 minutes' (a very common business mindset).
'Response in 15 minutes' means that someone will respond and confirm (this is a common way of thinking in IT/operations).
'24/7' means providing a full service at all times vs being on call only for critical issues.
'Failure' refers to anything that does not function perfectly vs the system being down.

Cooperation is not about competition or 'blame games', but about understanding and synergy. In order to achieve the set goals and the success of the project, both parties should work together and build trust. They should both be equally committed to these processes and, consequently, to achieving the goal. This is in their mutual interest.

5. Helpful materials

The materials below can help improve communication and save the Reader from unpleasant situations. Please note that these materials are based on scientific research and experience, but Readers should adapt them to their own situations and contexts.

5.1. Mediation checklists for IT projects

A) Warning signs: arrange mediation if any of the following occur:

☐ The review of the prepared functionality has been pending for more than three days. There has been no substantive review.

☐ Critical tasks have been blocked for more than 24 hours by another role or team.

☐ The proportion of reopened tasks in a sprint is greater than 10% (or the next sprint is increasing).

☐ There are two competing architectural visions, but no common decision-making model.

☐ Changes to the scope without adjusting the budget or deadline.

☐ Implementation surprises: what was delivered was different to what was expected.

☐ The tension surrounding the incident/failure.

☐ Symptoms of burnout include working overtime and increasing quality debt.

B) Preparing for the meeting:

☐ Collect facts and artefacts such as the backlog, architectural decisions, incident logs and metrics.

☐ Agree on the purpose of the meeting, i.e. which decision or problem needs to be removed from the critical path (in 1–2 sentences).

☐ Define the success criteria (e.g. after the session, there should be an architectural decision record, a sprint re-plan and action owners).

☐ Select a secure platform that does not record and enables waiting rooms to ensure confidentiality.

☐ Establish the rules for the meeting. For example, agree that there will be no interrupting, shouting or blaming.

C) Post-session artefacts (no 'meeting notes' in a disorganised format):

☐ **One-page brief:** problem, context, data, decisions made, criteria used, options considered, option selected, consequences.

☐ **A record of architectural decisions**, including context, decision, consequences, alternatives considered and status.

☐ **The Definition of 'Ready/Done'** has been updated and includes a glossary of ambiguous terms.

☐ **Action tracking:** owner, deadline, success metric and review date.

☐ **Feedback** to participants should be brief and factual, and should not introduce new topics.

5.2. 'Useful ticket' checklist:

- **Title:** It should be concise and specific, in the format: Verb + object + context (e.g. 'Create an export button in the admin panel').
- **Description of the issue:** A general description of the problem, its current status and the desired outcome, as well as the scope of work.
- **Steps to reproduce (for bugs):** The environment, application version, browser/device, exact steps, expectations vs. current situation, frequency of occurrence?
- **Context (Why):** A brief description of the business problem or user pain point that this task addresses. It answers the question, 'Why are we doing this?'
- **Objective (What):** A clearly defined expected outcome. How should things change once the task is complete?
- **Definition of Done (DoD):** A link to the team's definition of completion, which constitutes a quality contract (e.g. code that has been tested, reviewed, integrated and documented). This protects against the 'it works for me' syndrome.
- **Priority and deadline.**
- **Persons responsible:** Who is responsible for implementation, testing, acceptance, etc.
- **Additional materials:** For example, include a link to a mock-up of the project and screenshots of the problem.

5.3. Communication Manifesto (to be added to the team's README), example:

Objective: Establish a single, standardised 'communication contract' for the team to reduce misunderstandings and coordination costs.
1. The ‘obvious’ is the most expensive – without context, everyone adds their own meaning.
2. Agree on the definitions and examples before making any promises or writing any code. One promise without a definition equals risk.
3. A ticket is a contract, not a note marked 'ASAP'. Coding will not begin until the ticket is ready (DoR protects time and quality).
4. Avoid 'information archaeology' – keep decisions and context together in one place rather than spread across five tools.
5. Make sure that psychological safety is ensured – without it, people will stop talking about risks.
6. Design communication in the same way as architecture: the structure of the team is reflected in the system.

5.4. Channel policy (end of 'context switching'), example:

Objective: Reduce coordination costs and the time spent searching for information. A simple policy to adopt is that every type of information has its place, and conversations in ephemeral channels should always end with a link to the source of truth.
Slack/Teams is used for quick questions and operational arrangements (e.g. 'Who?', 'When?', 'What is blocking?'). However, arrangements that are to last longer than the thread should be linked to a ticket or document.
Jira is a tool for managing the scope of work, including problem descriptions, objectives, acceptance criteria, DoR/DoD, priorities, dependencies and implementation decisions.
Confluence/documentation stores long-term knowledge, such as architectural decisions and their reasoning, a glossary of terms, standards, runbooks and operating procedures. In short, it stores everything that should be easy to reproduce after a month.
Email is used for formal communication with customers and for summarising agreements, particularly when an audit trail or an unambiguous record of what was agreed is required, including links to tickets and documents.

5.5. 'Follow-up after the sale conversation' – email/note template, example:

Subject: Summary of support arrangements
Thank you for the conversation. To avoid any confusion, I am setting out the arrangements in writing so that we all understand them in the same way:
1) 24/7 coverage:
- 24/7 refers to: […]
- Outside the scope of 24/7: […]
2) 'Response within 15 minutes' means:
- [first response] within […]
- and not [repair time] (this is described in point 3)
3) Repair goals:
- P0: […]
- P1: […]
4) Importance levels + examples (P0–P3):
- […]
5) Communication during an incident:
- updates: P0 every […], P1 every […]
If you have a different understanding of something, please let us know and we will clarify it.

5.6. Leader card - behaviours that build psychological safety on a daily basis:

Objective: model curiosity, humility and shared responsibility. A leader is not always right; a leader creates an environment in which the team is more likely to speak up, report risks earlier and think more clearly.

Signals in language (ready-made sentences for use):
"I don't know. What do you think?"
"I could be wrong - what other options do we have?"
"What is unclear or risky about this?"
"Can anyone spot any weaknesses in this plan?"

Micro-behaviours (lasting 1–2 minutes but highly effective):
Normalise mistakes by starting with your own: "Yesterday, I did X wrong — let's correct it."
Separate the person from the problem. Criticise the idea, not the person.
Appreciate risk reports: publicly thank people for sharing 'bad news'.
Summarise the meaning: "In summary: decision A, reason B, risk C and next step D."
Allow for questions: "If something is unclear, it is better to ask questions now than to spend a week doing rework."

Things to avoid because they compromise safety:
Irony and embarrassment ("are you seriously asking that?").
Punishment for escalation ("why are you raising the alarm?").
Interrupting and ending discussions with authority ("because I said so").
Publicly holding people accountable for mistakes without context (this teaches them to hide problems).

5.7. 'Communication debt' — a measurable indicator

Objective: capture the cost of 'hidden work' associated with unclear arrangements, scattered knowledge, and differing interpretations of requirements in a simple, repeatable, and comparable way, from sprint to sprint.

Simple tracker for one sprint, ten minutes per day:
We bring together three counters in one place (e.g. Confluence, Google Sheet or sprint comment). The aim is to measure problems, not to blame someone.

"Does anyone remember what it was about?" (Recall count)
The number of times the team had to recreate the context because it had not been recorded. The counting rule is as follows: each 'context reproduction' counts as one, regardless of the length of the discussion.

Returns from QA (Quality Assurance) and UAT (User Acceptance Testing) due to different interpretations of the acceptance criteria (AC mismatch).
The number of tickets that came back not because 'something wasn't working', but because the scope and acceptance criteria were understood differently by other people.

The counting rule is as follows: if it returns because 'it wasn't supposed to be like that', then it counts as 1.

Searching for evidence through channels (Evidence hunt).
The number of times someone had to search for links, decisions or 'who decided what' in Slack, emails or documents because there was no central record.
The counting rule is as follows: each 'search for the source of the decision' counts as 1.

Reporting at the end of the sprint:
Sprint result: Recall + AC mismatch + Evidence hunt = X
In 2–3 sentences, explain where it hurts the most and why. Reasons could include a lack of mock-ups, architectural decisions or DoR/DoD, or decisions that are scattered.

Interpretation:
An increase in the index indicates rising costs of coordination and risk of rework, i.e. 'communication debt'. A decrease indicates that decisions are being finalised more effectively, with the team wasting less time recreating context.

There are more helpful materials on the next page.

5.8. Incident communication package

A) Slack — start of incident (internal)

Objective: establish a common context, owner, update rhythm and single source of truth in two minutes.

Starting message template:

```
[INCIDENT] P{1-4} — {short description, max. 1 sentence}

Impact:
- Who: {e.g. all users / some customers / region}
- What: {e.g. checkout does not work / login returns 500 / delays > 30 seconds}
- Scale: {~X% traffic / X customers / X orders}

Start: {HH:MM} (local time)
Coordinator: @{name}
Documentation: @{name}
Communication: @{name}
Status: Verification

Important links:
- Panel: {link}
- Logs / tracking: {link}
- Ticket: {link}
- Operating manual: {link}
- Timeline: {link}

Update frequency:
- Next update: {HH:MM}
- Then after {30 / 60} minutes or after a significant change in status

Rules:
- Facts > opinions. We mark hypotheses as 'H:'.
- All decisions and changes are posted here.
- If you are making a change: write what, where, when, who.
```

B) Slack — internal update (recurring)

Objective: prevent chaos and duplication of work.

```
Template:

[INCIDENT UPDATE] #{N} — {HH:MM}

Status: Investigation / Mitigation / Monitoring / Resolved
Impact (currently): {unchanged / increased / decreased} — {specific details}

What we know (facts):
- {F1}
- {F2}

Hypotheses (H):
- {H1} (owner @{...})
- {H2} (owner @{...})

What are we doing now:
- {task} — owner @{...} — ETA {~min}
- {task} — owner @{...}

Changes since the last update:
- {what, where, who, HH:MM} (e.g. rollback of service X to v1.2)

Risks / obstacles:
- {e.g. no access to partner logs, waiting for...}

Next update: {HH:MM}
```

C) To the customer — recurring update (externally)

Objective: even if there is no reason for it yet, the customer should feel that the situation is under control. The minimum required is clear information on the situation, its status, what we know, what we are doing and when the next update will be provided.

General template:

Subject: Incident {P1/P2} — {brief description} — Update #{N} ({HH:MM})

Good morning,
Status: Investigation / Mitigation / Monitoring / Resolution

Impact:
- {who and what} (e.g. some users cannot complete their orders)
- {since when} (e.g. since 12:40)

What we know at this point:
- {1–3 facts, no speculation}

What are we doing now:
- {1–3 actions in progress}

Workaround (if applicable):
- {e.g. we recommend X / we have implemented workaround Y}

Next update:
- No later than {HH:MM} or earlier if the status changes.

Best regards,
{First name and surname}
{Role / Team}

D) 'After the incident' — a brief summary that does not assign blame.

Objective: close the subject, leave no 'unfinished business' and prevent recurrence. Be brief, specific and systematic.
When: preferably within 24–72 hours.

Template (internal + optional for customers in a limited version):

[AFTER THE INCIDENT] { name / incident } — summary

1) Summary (1–2 sentences)
- What happened and what was the impact on users.

2) Impact
- Scope: {who / how many}
- Duration: {from – to}
- The greatest damage: {e.g. loss of revenue, delays, unavailability}

3) Timeline (key moments)
- HH:MM — detection / alert
- HH:MM — escalation / war room
- HH:MM — mitigation / rollback
- HH:MM — stabilisation
- HH:MM — closure

4) Root cause + contributing factors
- Root cause: {briefly and concisely}
- Contributing factors: {e.g. no circuit breaker, overly aggressive retries, no alarm, no test}

5) What worked well
- {e.g. rapid escalation, efficient rollback}

6) What we are improving (Remediations)
- {action} — owner @{...} — deadline {date}
- {action} — owner @{...} — deadline {date}

7) Changes to rules/definitions
- {e.g. runbook update, SLO/SLA change, alarm thresholds, deployment procedure}

Summary

In IT projects, imprecise communication at the outset often leads to costly changes or errors that are only discovered later. Research by NASA and others has shown that the cost of fixing errors resulting from misunderstandings in requirements increases rapidly in subsequent project phases, reaching a level in the operational phase that is usually tens of times higher than in the planning phase. In extreme cases, it can be thousands of times higher. Therefore, combatting the 'illusion of transparency' — the belief that our intentions are as clear to others as they are to us — is a priority for every team.

The key to success lies in shifting from passive information transfer to an active process of 'grounding', whereby the sender and receiver build a shared understanding by exchanging and verifying data. In practice, this involves implementing precise standards, such as the 'Definition of Ready' and the 'Definition of Done'. These act as 'quality gates', preventing unclear tasks from destabilising the system architecture or generating a false sense of progress.

Ultimately, however, no tool or process will be effective without the right organisational culture. Psychological safety is the foundation of effective communication, allowing employees to report concerns and mistakes without fear of punishment. In a world where time pressure often leads to risky shortcuts being taken, leaders must turn emotional conflicts into transactional compromises on scope and quality to protect the team from burnout and decision-making chaos. According to Conway's Law, the systems we create reflect the structure of our communication. If we want to build consistent software, we must first build a consistent team.

Tomasz Namyslo — I'm a senior web/mobile developer and IT consultant with many years of experience building and improving e-commerce stores, as well as mobile and web applications. I help companies deliver software predictably: less chaos, fewer reworks, shorter pipelines, and cleaner, more efficient code that doesn't come back to bite you a quarter later.

I treat cooperation as a partnership. I take responsibility for engineering and delivery, and in return, I expect the same level of commitment, decisiveness, and professionalism. Together, we set the goals, rules and pace of work, and we adhere to the agreement.

I am not the right choice if:
your priority is 'cheap and fast', and you are willing to accept that technological and architectural debt will accumulate along the way. I work to save time and money over months and years, not just to 'deliver the sprint'.

Google
★★★★★

Maciej Zawieja

"I highly recommend 'tomeckiStudio'!!! Mr Tomasz created a new website for my company (sitandtalk.pl). The entire process was carried out very professionally, with a thorough analysis of my needs and expectations. The website is exactly what I ordered. The service was carried out efficiently, with no unnecessary delays or problems."

Patrycja Kubacz

"I have the pleasure of regularly collaborating with Tomek on various website and online store projects for my clients. He is an excellent programmer. When I entrust him with a project, I know it will be completed to the highest standard. He is always able to implement a variety of ideas and needs, regardless of what I present him with. He provides valuable guidance at every stage of the project and keeps everyone involved informed. He also collaborates seamlessly with graphic designers to bring the project vision to life. Tomek's expertise is an invaluable asset to any company. I highly recommend him!"

12+
years of experience

20+
certificates, e.g. from Cisco and Securitum

130+
projects including 70% for agencies

tomek@tomecki.studio
https://tomecki.studio/
https://linkedin.com/in/tomasz-namyslo/

References

[1] K. Wnuk and B. Regnell, Requirements are slipping through the gaps — A case study on causes & effects of communication gaps in large-scale software development. 2011, pp. 37-46.

[2] B. Wróbel, "Rola komunikacji w zarządzaniu projektami," (in pol), The role of communication in project management, no. 3, pp. 119-129, 2007. [Online]. Available: http://www.ejournals.eu/sj/index.php/ZP/article/view/1023.

[3] R. Haleliuk. "Cybersecurity has a communication problem." Venture in Security. https://ventureinsecurity.net/p/cybersecurity-has-a-communication (accessed 12, 2025).

[4] Kaspersky, "Fluent in Infosec: Are c-level executives and IT security managers on the same page?," 2023. [Online]. Available: https://www.kaspersky.com/blog/speak-fluent-infosec-2023/

[5] M. Gorin, "Writing Engineering Tasks as a Team Process." [Online]. Available: https://maxim-gorin.medium.com/writing-engineering-tasks-as-a-team-process-afccc9767eb4

[6] M. Watson, "Top 7 Communication Problems in Software Development and How to Solve Them," ed: Full Scale, 2025.

[7] H. Khanna, "The Collaboration Paradox: Why Productivity Tools Made Work Harder." [Online]. Available: https://foresight.sparklin.com/collaboration-paradox-productivity-tools-failing-knowledge-work

[8] B. Wolfe, "Too Many Tools, Too Little Time: How Context Switching Is Killing Team Flow," 2025. [Online]. Available: https://lokalise.com/blog/blog-tool-fatigue-productivity-report/.

[9] T. Gilovich, K. Savitsky, and V. H. Medvec, "The illusion of transparency: biased assessments of others' ability to read one's emotional states," (in eng), J Pers Soc Psychol, vol. 75, no. 2, pp. 332-46, Aug 1998, doi: 10.1037//0022-3514.75.2.332.

[10] Anonymous. "Communication Barriers." LibreTexts Business. https://biz.libretexts.org/Bookshelves/Management/Principles_of_Management/12%3A_Communication_in_Organizations/12.4%3A_Communication_Barriers (accessed 12, 2025).

[11] C. D. Cramton, "The Mutual Knowledge Problem and Its Consequences for Dispersed Collaboration," 2001. [Online]. Available: https://doi.org/10.1287/orsc.12.3.346.10098.

[12] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," Commun. ACM, vol. 31, pp. 1268-1287, 11/01 1988, doi: 10.1145/50087.50089.

[13] H. C. Herbert and B. Susan, "Grounding in communication," 1991. [Online]. Available: https://api.semanticscholar.org/CorpusID:153811205.

[14] M. Fairlie, "Why Speaking in Jargon Doesn't Make You Look Smarter." [Online]. Available: https://www.business.com/articles/cut-the-code-why-speaking-in-technical-jargon-is-not-making-you-look-smarter/

[15] K. Alexander, "Jargon Conundrum: Its Effect on Workplace Communication," ed: The Survey Initiative, 2024.

[16] N. Staff, "Why Corporate Jargon Is Bad for Business," ed: NeuroLeadership Institute, 2023.

[17] H. C. Shulman, G. N. Dixon, O. M. Bullock, and D. Colón Amill, "The effects of jargon on processing fluency, self-perceptions, and scientific engagement," Journal of Language and Social Psychology, vol. 39, no. 5-6, pp. 579-597, 2020, doi: 10.1177/0261927X20902177.

[18] C. s. E. Team, "Too many Slack channels? Here's how to manage tech anxiety at work," D. C. Mosunic, Ed., ed: Calm, 2025.

[19] R. R. a. A. G. Rob Cross. (2016) Collaborative Overload. Available: https://www.robcross.org/wp-content/uploads/2021/04/HBR-Collaborative-Overload.pdf

[20] P.-L. Rodrigue, "How Team Stress Hurts Delivery + What to Do," ed: Axify, 2025.

[21] V. Terrón, J. Mejia, and M. Terron-Hernández, "A Systematic Literature Review on Technical Debt in Software Development: Types, Tools, and Concerns," International Journal of Combinatorial Optimization Problems and Informatics, vol. 16, pp. 288-302, 10/12 2025, doi: 10.61467/2007.1558.2025.v16i4.1150.

[22] R. Westrum, "A typology of organisational cultures," Quality and Safety in Health Care, vol. 13, no. suppl 2, p. ii22, 2004, doi: 10.1136/qshc.2003.009522.

[23] Microsoft, "What is Conway's Law?," ed: Microsoft, 2024.

[24] M. Odusola, "Conway's Law Explained," ed: splunk, 2023.

[25] M. P. Matthew Skelton, "Conway's Law: Critical for Efficient Team Design in Tech," ed: IT Revolution, 2020.

[26] A. Franzen, "Team Topologies - The Reverse Conway Manoeuvre," ed: AGILE Analytics, 2022.

[27] Graph, "Reverse Conway Maneuver," ed: Graph, 2025.

[28] S. Ferreira, J. Collofello, D. Shunk, and G. Mackulak, "Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation," Journal of Systems and Software, vol. 82, no. 10, pp. 1568-1577, 2009/10/01/ 2009, doi: https://doi.org/10.1016/j.jss.2009.03.014.

[29] Z. D. a. N. Nur, "A Study of the Impact of Requirements Volatility on Software Project Performance," 2002. [Online]. Available: https://opus.lib.uts.edu.au/bitstream/10453/2350/3/2004003534.pdf.

[30] A. F. Vincenzo Gervasi, Didar Zowghi, and Paola Spoletini, "Ambiguity in Requirements Engineering: Towards a Unifying Framework." [Online]. Available: https://arpi.unipi.it/bitstream/11568/1029138/1/Ambiguity_in_RE.pdf.

[31] M. Kuutila, M. Mäntylä, U. Farooq, and M. Claes, "Time pressure in software engineering: A systematic review," Information and Software Technology, vol. 121, p. 106257, 2020/05/01/ 2020, doi: https://doi.org/10.1016/j.infsof.2020.106257.

[32] C. Ribeiro and D. Berry, "The prevalence and severity of persistent ambiguity in software requirements specifications: Is a special effort needed to find them?," Science of Computer Programming, vol. 195, p. 102472, 2020/09/01/ 2020, doi: https://doi.org/10.1016/j.scico.2020.102472.

[33] M. L. Aleksandar Bajceta, Wasif Afzal, Pernilla Lindberg and Markus Bohlin, "Using NLP tools to detect ambiguities in system requirements - A comparison study." [Online]. Available: https://www.es.mdu.se/pdf_publications/6390.pdf.

[34] D. Basten, M. Müller, M. Ott, O. Pankratz, and C. Rosenkranz, "Impact of time pressure on software quality: A laboratory experiment on a game-theoretical model," PLOS ONE, vol. 16, no. 1, p. e0245599, 2021, doi: 10.1371/journal.pone.0245599.

[35] J. D. H. a. A. Mockus, "An Empirical Study of Speed and Communication in Globally Distributed Software Development," 2003. [Online]. Available: https://www.st.cs.uni-saarland.de/edu/empirical-se/2006/PDFs/herbsleb03.pdf.

[36] C. B. Alberto Bacchelli, "Expectations, Outcomes, and Challenges Of Modern Code Review." [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICSE202013-codereview.pdf.

[37] J. D. Jonette M. Stecklein, Brandon Dick, Bill Haskins, Randy Lovell, Gregory Moroney, "Error Cost Escalation Through the Project Life Cycle," 2004. [Online]. Available: https://ntrs.nasa.gov/citations/20100036670.

[38] M. Buenning, "How Human Error Relates to Cybersecurity Risks," ed: ninjaOne, 2025.

[39] J. Gregory, "Communication platforms play a major role in data breach risks," ed: IBM, 2025.

[40] krebsonsecurity, "Tech Firm Ubiquiti Suffers $46M Cyberheist," ed, 2015.

[41] erp-view, "Dług technologiczny w systemach ERP – ukryte wyzwanie dla organizacji," ed: erp-view, 2025.

[42] R. Williams, "Reduce and Manage Technical Debt," ed: Gartner, 2025.

[43] F. A. Administration. "Boeing 747-206B and Boeing 747-121." Federal Aviation Administration. https://www.faa.gov/lessons_learned/transport_airplane/accidents/PH-BUF (accessed.

[44] A. Alliance, "Three Amigos," ed. Agile Alliance, 2016.

[45] N. Reddy, "What are the Three Amigos in Agile? A Complete Guide," ed: Star Agile, 2025.

[46] M. Gorin, "Ready vs Done: The Underrated Process That Keeps Work Clean." [Online]. Available: https://maxim-gorin.medium.com/tired-of-tasks-starting-half-baked-or-closing-half-done-learn-how-dor-and-dod-keep-your-team-aligne-b912dfdf828c

[47] R. Wohlrab, P. Pelliccione, E. Knauss, and M. Larsson, "Boundary objects and their use in agile systems engineering," Journal of Software: Evolution and Process, vol. 31, no. 5, p. e2166, 2019/05/01 2019, doi: https://doi.org/10.1002/smr.2166.

[48] C. K. De Dreu and L. R. Weingart, "Task versus relationship conflict, team performance, and team member satisfaction: a meta-analysis," (in eng), J Appl Psychol, vol. 88, no. 4, pp. 741-9, Aug 2003, doi: 10.1037/0021-9010.88.4.741.

[49] J. L. a. S. Lueder, "Postmortem Culture: Learning from Failure," in Site Reliability Engineering, G. O. Connor Ed.: Google, 2016.

[50] A. Edmondson, "Psychological Safety and Learning Behavior in Work Teams," Administrative Science Quarterly, vol. 44, no. 2, pp. 350-383, 1999, doi: 10.2307/2666999.

Bibliography

[51] S. Alliance, "Everything You Need to Know About Acceptance Criteria." [Online]. Available: https://resources.scrumalliance.org/Article/need-know-acceptance-criteria

[52] S. Alliance, "Definition of Done vs. Definition of Ready." [Online]. Available: https://resources.scrumalliance.org/Article/definition-vs-ready

[53] T. Asana, "Jak utworzyć brief projektu w 7 krokach," vol. 2026, ed: Asana, 2026. [Online]. Available: https://asana.com/pl/resources/design-brief

[54] Atlassian, "Acceptance criteria: examples and best practices," vol. 2025, ed: Atlassian. [Online]. Available: https://www.atlassian.com/work-management/project-management/acceptance-criteria

[55] Atlassian, "Definition of Ready (DoR) Explained & Key Components," vol. 2025, ed: Atlassian, 2024. [Online]. Available: https://www.atlassian.com/agile/project-management/definition-of-ready

[56] Atlassian, "How to master backlog refinement meetings," vol. 2025, ed: Atlassian, 2025. [Online]. Available: https://www.atlassian.com/agile/project-management/backlog-refinement-meeting

[57] AVH, "How to write a useful Jira ticket," ed: Atlassian, 2022. [Online]. Available: https://community.atlassian.com/forums/Jira-articles/How-to-write-a-useful-Jira-ticket/ba-p/2147004

[58] J. Bancroft-Connors, "The Keys to Effective Product Backlog Refinement," vol. 2025, ed: Scrum Alliance, 2021. [Online]. Available: https://resources.scrumalliance.org/Article/keys-effective-product-backlog-refinement

[59] A. Borcz, "Techniki efektywnej komunikacji z klientem: kluczowe umiejętności komunikacyjne w sprzedaży," vol. 2025, ed: Enterprise Advisors Polska, 2025. [Online]. Available: https://enterpriseadvisors.pl/blog-o-sprzedazy/techniki-efektywnej-komunikacji-z-klientem-kluczowe-umiejetnosci-komunikacyjne-w-sprzedazy/

[60] E. Brooks, "Mastering the Definition of Done for User Stories: 5 Key Elements for Success," vol. 2025, ed: ONES, 2025. [Online]. Available: https://ones.com/blog/mastering-definition-of-done-user-stories/

[61] M. Cataldo, P. Wagstrom, J. Herbsleb, and K. Carley, Identification of coordination requirements: Implications for the Design of collaboration and awareness tools. 2006, pp. 353-362. DOI: 10.1145/1180875.1180929

[62] A. Clarke, "High vs low-context: rethinking team communication," vol. 2025, ed: Vestd, 2025. [Online]. Available: https://www.vestd.com/blog/high-context-vs-low-context-cultures-how-communication-style-shapes-global-teams
[63] CLAYHR, "Embracing Conway's Law: How Your Organization's Structure Shapes Your Software," vol. 2025, ed: CLAYHR, 2025. [Online]. Available: https://www.clayhr.com/blog/embracing-conways-law-how-your-organizations-structure-shapes-your-software
[64] F. Corp, "Bridging the Language Barrier: Enhancing Cross-Cultural Communication in International Teams," vol. 2025, ed: Fluency Corp, 2025. [Online]. Available: https://fluencycorp.com/enhancing-cross-cultural-communication-in-international-teams/
[65] Dawid, "Dług technologiczny w przedsiębiorstwie: Jak rozpoznać, zmierzyć i skutecznie zredukować ukryte koszty przestarzałych systemów," vol. 2025, ed: Aveneo, 2025. [Online]. Available: https://aveneo.pl/pl/blog/dlug-technologiczny-w-przedsiebiorstwie-jak-rozpoznac-zmierzyc-i-zredukowac-ukryte-koszty
[66] Z. P. Expert, "Dług technologiczny – co to jest i dlaczego może zrujnować projekt IT," vol. 2025, ed: play.pl, 2025. [Online]. Available: https://www.play.pl/play-expert/porady/dlug-technologiczny---co-to-jest-i-dlaczego-moze-zrujnowac-projekt-it
[67] K. Fedko, "Silos informacyjny w firmie: jak go rozwiązać?," vol. 2025, ed: Thulium, 2021. [Online]. Available: https://thulium.com/pl/blog/silos-informacyjny-w-firmie-jak-go-rozwiazac
[68] S. M. Galloway, "Nurturing Psychological Safety while Upholding Accountability," vol. 2025, ed: ProAct Safety, 2024. [Online]. Available: https://proactsafety.com/articles/nurturing-psychological-safety-while-upholding-accountability
[69] Ideo, "Dług technologiczny – co to jest i jakie są jego skutki?," vol. 2025, ed: Ideo, 2024. [Online]. Available: https://www.ideo.pl/firma/o-nas/nasze-publikacje/dlug-technologiczny-skutki,210.html
[70] Ironhack, "Functional vs Non-Functional Requirements: Understanding the Core Differences and Examples," vol. 2025, ed: Ironhack, 2024. [Online]. Available: https://www.ironhack.com/us/blog/functional-vs-non-functional-requirements-understanding-the-core-differences-and
[71] J. Janik, I. Drobina, and R. Drobina, "EFEKTYWNOŚĆ KOMUNIKACJI W ZESPOŁACH PROJEKTOWYCH ROZPROSZONYCH GEOGRAFICZNIE," 2023, pp. 93-108. DOI 10.53052/9788367652124.10
[72] Kaspersky, "Miscommunications in IT security lead to cybersecurity incidents in 62% of companies," vol. 2025, ed: Kaspersky, 2023. [Online]. Available: https://www.kaspersky.com/about/press-releases/miscommunications-in-it-security-lead-to-cybersecurity-incidents-in-62-of-companies
[73] J. Kłosińska, "Nowe formy budowania relacji z klientem w internecie za pomocą takich narzędzi jak content marketing, real-time marketing, aplikacje mobilne, portale społecznościowe, komunikacja video," NSZ, vol. 13, no. 3, pp. 15-27, 2018 2018, doi: 10.37055/nsz/129488. [Online]. Available: https://doi.org/10.37055/nsz/129488
[74] G. Krüger, "Non-Functional Requirements: Tips, Tools, and Examples," vol. 2025, ed: Perforce, 2025. [Online]. Available: https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples
[75] LeaderFactor, "Culture of Psychological Safety," vol. 2025, ed: LeaderFactor. [Online]. Available: https://www.leaderfactor.com/learn/culture-of-psychological-safety
[76] J. L. a. S. Lueder, "Postmortem Culture: Learning from Failure," in Site Reliability Engineering, G. O. Connor Ed., 2016. [Online]. Available: https://sre.google/sre-book/postmortem-culture/
[77] S. Nikonenko, "Effective Brief for Software Development + Free Template," vol. 2025, ed: Purrweb, 2024. [Online]. Available: https://www.purrweb.com/blog/brief-for-software-development/
[78] S. Overflow, "A practical guide to writing technical specs," vol. 2025, ed: Stack Overflow, 2020. [Online]. Available: https://stackoverflow.blog/2020/04/06/a-practical-guide-to-writing-technical-specs/
[79] S. Petrova, "How to Write a Brief for a Software Project," vol. 2025, ed: Adeva, 2020. [Online]. Available: https://adevait.com/blog/startups/write-brief-for-software-project

[80] J. Probst, "How to Tackle Technical Debt and Maintain High Software Quality," vol. 2025, ed: Qt, 2025. [Online]. Available: https://www.qt.io/quality-assurance/blog/how-to-tackle-technical-debt
[81] ProductPlan, "What is Acceptance Criteria?," vol. 2025, ed: ProductPlan, 2019. [Online]. Available: https://www.productplan.com/glossary/acceptance-criteria/
[82] P. Safety, "What is Psychological Safety?," vol. 2025, ed: Iterum Ltd, 2010. [Online]. Available: https://psychsafety.com/about-psychological-safety/
[83] S. Salimi, "What is the Definition of Done (DOD) in Agile?," vol. 2026, ed: Agile Academy. [Online]. Available: https://www.agile-academy.com/en/scrum-master/what-is-the-definition-of-done-dod-in-agile/
[84] Smartling, "Low-context culture and high-context culture: Communication styles in global business," vol. 2025, ed: Smartling, 2025. [Online]. Available: https://www.smartling.com/blog/high-vs-low-context-culture
[85] V. Solutions, "Functional VS Non-functional Requirements," vol. 2025, ed: Visure Solutions, 2025. [Online]. Available: https://visuresolutions.com/alm-guide/functional-vs-non-functional-requirements/
[86] StoriesOnBoard, "Understanding Acceptance Criteria: Examples and Best Practices," vol. 2025, ed: StoriesOnBoard, 2023. [Online]. Available: https://storiesonboard.com/blog/acceptance-criteria-examples
[87] K. S. J. Sutherland, "The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game." [Online]. Available: [Online]. Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf
[88] K. Taylor, "What does psychological safety mean, anyway?," vol. 2025, ed: Atlassian, 2024. [Online]. Available: https://www.atlassian.com/blog/teamwork/what-does-psychological-safety-mean-anyway
[89] P. L. L. Team, "Impact of poor communication in dev team productivity: 7 proven solutions," vol. 2025, ed: Preply Inc, 2025. [Online]. Available: https://preply.com/en/blog/b2b-impact-of-poor-communication-in-dev-team-productivity/
[90] M. Tisinger, "How not to handle a cyberattack: The worst crisis communications failures," vol. 2025, ed: CyberRisk Alliance, 2025. [Online]. Available: https://www.scworld.com/perspective/how-not-to-handle-a-cyberattack-the-worst-crisis-communications-failures
[91] P. Weichbroth, M. Lotysz, and M. Wróbel, A survey on the impact of emotions on the productivity among software developers. 2025. DOI: 10.48550/arXiv.2510.04611
[92] B. Wróbel, "Rola komunikacji w zarządzaniu projektami," (in pol), The role of communication in project management, no. 3, pp. 119-129, 2007. [Online]. Available: http://www.ejournals.eu/sj/index.php/ZP/article/view/1023