Assumptions in System Design: Avoiding the Hidden Traps

Natalia Pospieszyńska

Abstract

In system design, assumptions—whether explicit or hidden—play a crucial role in shaping the success and stability of complex systems. However, when left unchecked, especially when influenced by cognitive biases, these assumptions can lead to serious risks and system failures. This article takes a closer look at the challenge of identifying and managing hidden assumptions that can impact performance, scalability, and reliability. Traditional methods like cross-team reviews and standard testing often miss these implicit assumptions until they create critical issues. To help mitigate these risks, this article presents practical strategies such as scenario planning, designing for failure, and validating assumptions through thorough testing. A real-world example—the failure of NASA's Mars Climate Orbiter—highlights the importance of these practices. A simple, unexamined assumption about units of measurement led to a $125 million mission loss. By proactively managing assumptions, teams can reduce errors, improve communication, and build systems that are far more resilient and reliable.

Version 1, October 2024

Contents

1 Introduction

An assumption is an assertion or statement that is taken as true or supposed as a fact without proof or substantiating evidence [Cor]. In system design, assumptions play a crucial role in simplifying decision-making and enabling projects to move forward. These assumptions can span a broad range of areas—from technical limitations and system dependencies to user behaviors and environmental conditions. Teams may assume, for instance, the stability of network connections, the consistent availability of third-party services, or predictable user growth. While these assumptions reduce complexity and provide a foundation for initial design choices, they also carry significant risks if left unexamined or unvalidated.

The real danger lies in hidden or implicit assumptions—those that are taken for granted without thorough investigation. When these assumptions are incorrect, they can lead to serious system failures, affecting performance, scalability, security, and overall project success. Relying on the constant availability of a third-party API without accounting for potential disruptions can result in unexpected downtime, compromising system functionality. These kinds of oversights can range from minor issues to catastrophic failures that are costly and difficult to repair.

Standard practices, such as testing and peer reviews, are commonly employed to mitigate these risks, but they often fall short in identifying hidden assumptions or accounting for the cognitive biases that influence decision-making. Biases like confirmation bias, anchoring bias, and optimism bias can lead teams to focus on supporting evidence or overly positive outcomes while ignoring critical risks. Left unchecked, these biases further complicate the process, making it harder to surface and address hidden assumptions in system design.

This article advocates for a more comprehensive approach to assumption management, emphasizing strategies such as explicit documentation, scenario planning, and designing for failure. Documenting assumptions is particularly vital as it brings hidden risks to the forefront, making them visible, traceable, and easier to validate throughout the system's lifecycle. Without such documentation, assumptions often remain untested until they cause major, sometimes irreversible, damages. Regularly validating and communicating assumptions across all teams and stakeholders is crucial for building systems that are resilient and adaptable to change.

The effectiveness of these practices is underscored by the case study of NASA's Mars Climate Orbiter, a mission that failed due to an unexamined assumption about units of measurement, resulting in a $125 million[1] loss. By understanding how to uncover and address hidden assumptions, teams can prevent these from derailing projects and improve overall system resilience.

2 How Cognitive Biases Distort Assumptions

Cognitive biases are systematic errors in judgment that arise when individuals rely on mental shortcuts, known as heuristics, to simplify complex decision-making processes. While these shortcuts are often helpful, they can lead to predictable errors in reasoning, resulting in

[1] Value in 1999 dollars.

biases such as overestimating probabilities or misinterpreting information. These biases emerge because the brain prioritizes efficiency over accuracy when processing vast amounts of data [TK74].

In system design, these biases can distort assumptions, leading to flawed decisions that affect the functionality, scalability, and reliability of the system. These biases lead teams to base decisions on incomplete or skewed information, often resulting in flawed assumptions that affect the design and implementation of systems. Some of the most common cognitive biases impacting decision-making in software design include:

- **Confirmation Bias**: The tendency to focus on information that supports pre-existing beliefs, ignoring data that contradicts them.
- **Anchoring Bias**: The tendency to rely too heavily on the first piece of information encountered (the 'anchor') when making decisions, leading to a narrowed perspective.
- **Optimism Bias**: The inclination to overestimate the likelihood of favorable outcomes, which can result in underestimating risks and challenges.
- **Survivorship Bias**: A focus on successful cases while neglecting the failures, leading to overly optimistic conclusions based on incomplete evidence.

Confirmation Bias: Focusing on Supporting Data

Confirmation bias, as explained by Nickerson [Nic98], is the cognitive tendency for individuals to focus on information that supports their pre-existing beliefs or hypotheses while dismissing or undervaluing evidence that contradicts them. This bias influences how people gather, interpret, and recall information, often reinforcing their initial assumptions rather than encouraging objective evaluation of alternative perspectives. It is widespread and affects reasoning in everyday life as well as professional domains.

In system design, confirmation bias can manifest when a designer favors a solution or architecture they are already familiar with. If a software architect strongly believes that a specific database management system (DBMS) is the best choice for most projects, they may highlight its benefits, such as scalability and reliability, while downplaying its limitations, such as cost or incompatibility. This selective focus can prevent the team from fully exploring other options or critically assessing the system's actual requirements. As a result, confirmation bias can lead to suboptimal decisions that fail to consider more suitable alternatives.

To mitigate confirmation bias, teams should actively seek out contradictory data or play devil's advocate when evaluating assumptions. By encouraging open dialogue and involving diverse viewpoints, organizations can ensure that decisions are made based on a more comprehensive analysis, reducing the risk of critical flaws going unnoticed.

Anchoring Bias: Relying on Initial Information

Anchoring bias refers to the cognitive tendency for individuals to rely heavily on the first piece of information (the 'anchor') they receive when making decisions. This initial anchor significantly influences subsequent judgments, even when new, more relevant information

becomes available. Tversky and Kahneman [TK74] demonstrated that once an anchor is set, individuals find it difficult to adjust their evaluations away from it, which often leads to distorted decision-making. This bias can prevent teams from fully integrating updated or contradictory data into their decision process.

In system design, anchoring bias can emerge when early project estimates or assumptions become fixed reference points for future decisions. If an initial estimate projects that a system will support 1,000 concurrent users, teams might fixate on that figure when designing the system's infrastructure. As the project evolves, even if new data suggests that the user base could grow much larger or much smaller, the original estimate continues to drive decisions. This can lead to over-engineered or under-engineered systems, both of which waste resources and time.

To counteract anchoring bias, teams should revisit early assumptions frequently, especially as more data becomes available. This ensures that decisions are based on current and accurate information rather than outdated or arbitrary anchors.

Optimism Bias: Overestimating Positive Outcomes

Optimism bias refers to the cognitive tendency for individuals to overestimate the likelihood of positive outcomes while underestimating the probability of risks or setbacks. Weinstein [Wei80] described this phenomenon as *unrealistic optimism*, where people believe they are less likely than others to encounter negative events. This bias results in overly positive expectations, particularly in complex projects where unforeseen challenges are common. As a result, optimism bias often skews decision-making and risk management.

In system design, optimism bias can manifest when teams assume overly favorable outcomes, such as underestimating development challenges or assuming that systems will scale effortlessly. A development team might set an overly aggressive project timeline, assuming that all phases of development will proceed without delays. Similarly, teams may assume that the system will handle growing user loads without issues, leading to performance bottlenecks if the actual growth outpaces expectations. These assumptions can result in delayed timelines, unexpected costs, and system failures that could have been avoided with more realistic planning.

To mitigate optimism bias, teams should employ risk assessments and contingency planning to account for potential setbacks. By acknowledging that obstacles are likely to arise, teams can create more flexible and realistic project plans that minimize the impact of unforeseen events.

Survivorship Bias: Ignoring Failures

Survivorship bias refers to the cognitive tendency to focus on successful outcomes while overlooking failures, leading to skewed conclusions about the overall situation. As explained by Taleb [Tal05], this bias arises when attention is placed disproportionately on the 'survivors' or successes, while ignoring the failures that might provide critical insights. Survivorship bias can create a false narrative of success, leading teams to miss valuable lessons from less visible failure cases.

In system design, survivorship bias can manifest when teams focus solely on a system's current performance, neglecting to consider potential weaknesses or failure scenarios. A system that performs well under normal conditions may give the misleading impression that it will continue to function effectively under all circumstances. This assumption can cause teams to overlook stress-testing or planning for rare but catastrophic failure scenarios. As a result, the system may fail during unexpected events or under unusual loads, causing downtime and significant losses that could have been mitigated through more comprehensive preparation.

To counteract survivorship bias, teams should include post-mortem analysis of both successful and failed projects. This broader perspective helps identify weaknesses and areas for improvement that would otherwise be missed by focusing only on what went right.

While cognitive biases certainly distort how assumptions are formed and validated, they also have a broader, more pervasive impact: they prevent teams from critically examining these assumptions, allowing flawed ideas to persist throughout the lifecycle of a project. Whether it is confirmation bias driving teams to favor only supporting evidence, or optimism bias leading to unrealistic projections, these biases leave assumptions unchecked and incomplete, posing significant risks to a system's performance, scalability, and security.

Addressing these biases is critical, but that is only the first step. Beyond cognitive biases, assumptions themselves—whether shaped by biases or simply left unexamined—are a major source of risk in system design. These risks can remain hidden until they manifest as performance issues, security vulnerabilities, or even complete system failures. To truly understand how these assumptions influence a system, it is essential to take a step back and examine assumptions in their entirety, not just through the lens of bias.

In the following section, I will dive deeper into why assumptions matter so much in system design, exploring how they shape the decisions that determine a system's resilience and reliability. By managing these assumptions effectively, teams can minimize risks and build systems that are better equipped to handle the complexities and challenges of real-world operations.

3 Why Assumptions Matter in System Design

Assumptions are fundamental to system design because they shape the decisions that teams make throughout the design and development process. Whether assumptions are technical (e.g., about scalability or performance), business-related (e.g., project timelines), or security-focused, they serve as the foundation on which design decisions are built. However, assumptions can also introduce significant risks if they remain unvalidated or unexamined. When an assumption is incorrect or incomplete, it can lead to serious consequences, such as performance issues, security vulnerabilities, or project delays.

Cognitive biases play a major role in how assumptions are formed and left unchecked. Confirmation bias may lead teams to focus on data that supports their preconceived notions, while anchoring bias can cause teams to fixate on early estimates without considering new information. These biases prevent teams from critically assessing the assumptions they are making, which increases the likelihood of hidden risks surfacing later in the project lifecycle.

To effectively manage assumptions, teams must not only identify them but also continually validate and challenge them. By addressing cognitive biases and ensuring that assumptions are revisited throughout the project, teams can reduce the risk of hidden vulnerabilities disrupting system design. The next section explores the specific types of assumptions that frequently arise in system design, along with their consequences. By understanding the different types of assumptions, we can better assess their impact and develop strategies for managing them.

Types of Assumptions and Their Impact on Design

Teams make assumptions in several areas of system design, often as a way to simplify complex decisions and move forward with development. These assumptions may concern technical constraints, security measures, business goals, or user behavior. However, each category of assumptions comes with its own risks, and if left unexamined, they can lead to unexpected challenges during later stages of development. By identifying, documenting, and validating assumptions early, teams can mitigate potential risks and design systems that are more resilient to change and uncertainty. Below are some common types of assumptions that significantly impact system design.

Technical Assumptions

Technical assumptions include beliefs about system performance, scalability, and fault tolerance. Teams may assume that increasing hardware resources will automatically resolve performance issues, without fully considering the limitations of the underlying software architecture. But simply adding resources will not necessarily address system bottlenecks if architectural flaws remain. Regularly challenging these technical assumptions is critical, as unexamined beliefs about infrastructure and capacity can lead to unanticipated system failures or inefficiencies.

Another challenge arises when technical assumptions persist without revalidation during system evolution. As new features or integrations are added, initial assumptions about system behavior may no longer hold. If teams fail to update their understanding of performance limits or fault tolerance under new conditions, the system may become increasingly fragile. Regular testing and reevaluation of technical assumptions help ensure that the system remains reliable and scalable as it grows and adapts to new demands.

Security and Privacy Assumptions

Security and privacy assumptions often rest on the belief that existing measures—such as encryption or authentication—are sufficient to protect against all threats. However, as new vulnerabilities and attack vectors emerge, relying on outdated security protocols can leave systems exposed. Assuming that once-strong encryption standards remain effective without regular updates can create blind spots in a system's defenses.

Additionally, privacy assumptions can be challenged by changes in laws and regulations. With evolving legal frameworks such as GDPR or HIPAA, what was once considered compliant may no longer suffice. If these assumptions aren't regularly revisited, teams risk failing to meet new standards, leading to potential legal and reputational consequences. Continuous audits of security and privacy protocols are essential to adapt to new threats and regulatory shifts, ensuring ongoing protection and compliance.

Business and Process Assumptions

Business assumptions often involve expectations around timelines, budgets, and stakeholder priorities, but these factors can shift rapidly, especially in volatile environments. Assuming that project timelines will remain stable despite potential resource constraints or market changes can lead to missed deadlines and cost overruns. Similarly, unexamined assumptions about stakeholder priorities may result in misaligned efforts between technical teams and business goals.

Moreover, assumptions about market conditions or regulatory requirements can introduce significant risks. A change in regulations or a sudden shift in the competitive landscape can render previous assumptions about pricing models or resource availability obsolete. If teams fail to adapt their processes in response to such changes, they risk delivering products or systems that no longer meet business objectives or user needs. Regularly revisiting and validating business and process assumptions helps ensure that projects remain aligned with evolving goals and external conditions.

User and Usage Assumptions

User and usage assumptions often hinge on predicted patterns of interaction with the system, but these assumptions can be highly unreliable. Systems may be designed based on expected user behavior, such as gradual adoption or predictable feature use, but users often act in unexpected ways. A surge in traffic due to an unanticipated viral event or the misuse of features can lead to system overload or degraded performance. When usage patterns deviate from assumptions, the system's robustness can quickly be compromised.

Another challenge is anticipating how different user groups will engage with the system. Design decisions based on assumed user preferences may not align with the actual needs or habits of end users. If assumptions about user behavior aren't regularly revisited and validated through feedback and testing, the system can fail to meet user expectations. Over time, the gap between user needs and system capabilities can grow, leading to user frustration and potential attrition. Adapting to evolving user behavior requires continuous monitoring and reassessment to ensure system flexibility and performance.

While explicit assumptions in technical, security, business, and user domains are often managed with oversight, the most dangerous risks come from those assumptions that remain hidden—unstated, undocumented, and unexamined. These hidden assumptions quietly influence decision-making, shaping system behavior in ways that often go unnoticed until they trigger failures, performance issues, or costly project delays. Managing explicit assumptions is certainly essential, but it is only part of the solution. The real challenge lies in uncovering and addressing the implicit assumptions that can silently undermine even the most well-designed systems.

The next section delves into the nature of these hidden assumptions—the silent project killers. By understanding where they come from and how they can derail a system, we can develop strategies to surface and mitigate these risks before they lead to significant damage.

4 Hidden Assumptions: Undermining System Stability

Keeling [Kee17] makes an important point when he says, 'Assumptions are truths about the system we simply take for granted. Hidden assumptions kill projects (or at least cause significant pain).' Often the assumptions we don't talk about pose the biggest risks.

In system design, hidden assumptions are tricky because they tend to be accepted as 'truths' without teams even realizing it. These assumptions often fly under the radar, either because teams are racing against deadlines, following familiar routines, or simply overlook them. They only come to light when things go wrong, leading to delays, degraded performance, or even system failure.

The Nature of Hidden Assumptions

Hidden assumptions are especially risky because they shape decisions without ever being explicitly documented or communicated. Teams might assume, for instance, that a critical third-party service will always be available or that the network infrastructure will be consistently reliable. These assumptions, which are often based on prior experience or routine operations, remain unnoticed until an outage or failure exposes them. By that point, the damage has already been done, and teams are left to scramble for reactive solutions.

Such assumptions often slip through the cracks because they seem 'too obvious' to question. For example, a team might assume that user growth will follow past trends without factoring in external changes that could dramatically shift demand. This lack of critical scrutiny allows assumptions to remain embedded in the design process, quietly undermining the system's resilience to unforeseen conditions.

Common Hidden Assumptions and Their Impact

Hidden assumptions are those that are never explicitly stated, documented, or communicated, and they often go unchallenged throughout the system design process. These assumptions are particularly dangerous because they can undermine the success of a project without being recognized until it is too late. Below are some of the most common hidden assumptions that frequently lead to system failures.

Availability of Third-Party Services

Teams often assume that third-party services or external APIs will always be available and function reliably, without considering potential disruptions. This assumption remains hidden when teams neglect to account for downtime, service outages, or changes in the third-party provider's offerings. If a system relies heavily on a payment processing API and the provider experiences an outage, the entire system could grind to a halt. This hidden assumption becomes a critical point of failure when there is no contingency plan or backup service in place. Teams should always consider service-level agreements (SLAs) and design for resilience, acknowledging that third-party dependencies are outside their direct control.

Scalability

Another common hidden assumption is that the system will scale effortlessly with increased usage. Teams often assume that as user traffic grows, the system will continue to perform without degradation, overlooking the specific design and infrastructure constraints that might limit scalability. This assumption becomes a problem when systems are not stress-tested for high loads, resulting in poor performance or failure during spikes in demand. A failure to plan for scalability can lead to bottlenecks that disrupt user experience and cause costly downtime. Addressing scalability requires proactively identifying potential choke points in system architecture and validating assumptions about how the system will behave under increased load.

Network Reliability

Network reliability is frequently assumed but rarely questioned, leading teams to overlook scenarios where networks may fail, slow down, or become unreliable. This hidden assumption can create vulnerabilities, particularly for distributed systems that rely on constant communication between services. Teams might assume that a cloud provider's network is always available, or that internal network infrastructure will never suffer from outages. However, network disruptions can cause cascading failures throughout the system, leading to data loss or delayed transactions. Addressing this hidden assumption requires building fault-tolerant systems that can handle intermittent connectivity or implementing retry mechanisms to ensure data consistency.

Hidden assumptions are particularly dangerous because they remain unnoticed until they lead to significant issues—often at the worst possible moments. Once these assumptions surface, the consequences can be far-reaching, affecting not just the immediate system performance but also the broader success of the project. The next step in understanding the true danger of hidden assumptions is to examine their wide-ranging impact, highlighting the severe financial, operational, and reputational consequences they can impose on projects.

The Cost of Hidden Assumptions

The cost of hidden assumptions can be enormous, ranging from technical failures and project delays to financial losses and reputational damage. When these assumptions go unchallenged or unnoticed, they often result in costly consequences that could have been avoided through better assumption management. Below are some key areas where hidden assumptions can have a significant impact.

Technical Failures and System Downtime

One of the most immediate costs of hidden assumptions is technical failure, often occurring at the worst possible times, like during high-demand periods. Such failures result in system downtime that disrupts the user experience, leads to data loss, missed transactions, and extends recovery times.

If a hidden assumption about third-party service availability goes unaddressed, an unexpected outage could leave the system unable to process payments or complete transactions. This could result in a cascading failure, where one hidden assumption about a minor dependency cripples the entire system, leaving teams scrambling to implement fixes. The cost of these technical failures is not limited to the immediate disruption—they also lead to longer-term maintenance costs as teams rush to patch vulnerabilities and prevent future occurrences.

Financial Losses and Opportunity Costs

Hidden assumptions often lead to financial losses that could have been avoided with better risk management. For example, when scalability assumptions are left unchecked, systems can fail to meet user demand, resulting in lost sales, unfulfilled customer orders, and dissatisfied users. Downtime caused by hidden assumptions about system capacity or third-party service availability can directly translate into lost revenue.

In addition to direct financial losses, there are significant opportunity costs associated with hidden assumptions. Time spent fixing unanticipated failures means less time available for developing new features or improving existing systems. Teams often find themselves in 'firefighting' mode, addressing unexpected issues caused by unexamined assumptions, rather than focusing on innovation or growth. The cumulative effect of these missed opportunities can slow down the overall progress of a project or company.

Reputational Damage and Trust Erosion

When hidden assumptions lead to public system failures or breaches, the resulting reputational damage can be even more costly than the immediate technical or financial losses. Customers and users have high expectations for system reliability and security, and failure to meet those expectations can erode trust. A hidden assumption about the robustness of security measures could result in a data breach, leading to public embarrassment, legal consequences, and loss of customer trust.

The reputational impact of these failures often has long-lasting effects. In industries like finance or healthcare, where trust is paramount, a single failure due to hidden assumptions can permanently damage relationships with clients, customers, and stakeholders. Recovery from reputational damage is often slow and costly, requiring increased investment in marketing, customer service, and security to regain lost trust.

Project Delays and Increased Costs

Hidden assumptions can also lead to significant project delays. When assumptions about timelines, resource availability, or technical feasibility are incorrect, teams often find themselves behind schedule. These delays can compound when multiple hidden assumptions are uncovered late in the project lifecycle, leading to unexpected rework and extended timelines. Assuming that a system can handle a certain volume of traffic without proper validation may lead to performance issues that require significant redesigns, delaying product launches.

Moreover, when teams are forced to address these issues reactively, project costs rise. Last-minute fixes, additional testing, or scaling infrastructure beyond initial estimates all increase the overall cost of the project. Hidden assumptions that aren't surfaced early in the design process often lead to budget overruns, as teams scramble to meet deadlines while addressing unforeseen problems.

Increased Technical Debt

Finally, hidden assumptions contribute to technical debt, which accumulates over time as systems are built on top of flawed assumptions. When assumptions about system architecture, performance, or scalability go unchallenged, teams are forced to implement short-term solutions that work around the underlying problems. These workarounds may solve the immediate issue, but they often introduce complexity and inefficiencies into the system, making future development more challenging.

As technical debt grows, it becomes harder to maintain the system, and new features or improvements become more difficult to implement. The long-term cost of hidden assumptions, therefore, extends beyond the initial failure, as teams must spend more time and resources managing the complexity introduced by patching over unexamined assumptions.

The high price of hidden assumptions—whether in the form of technical failures, financial losses, or long-term technical debt—underscores the importance of proactive assumption management. Addressing these costs early is essential to prevent them from escalating into larger, more complex issues that can cripple a project. The key to avoiding these pitfalls lies in uncovering hidden assumptions before they cause damage.

Next I will explore practical strategies for surfacing these assumptions, from scenario planning to designing for failure, so teams can mitigate risks and build more resilient systems. Understanding how to identify and challenge these hidden risks is critical to ensuring that they do not undermine the success and stability of the projects.

Uncovering Hidden Assumptions in System Design

Uncovering hidden assumptions requires a proactive and systematic approach. These assumptions are often implicit, unspoken, or unchallenged, making them particularly dangerous in system design. Hidden assumptions can remain unnoticed until they lead to costly failures, disruptions, or rework. Several key techniques can help teams surface these assumptions early, preventing them from becoming project risks.

Scenario Planning

Scenario planning encourages teams to explore potential edge cases and 'what if' scenarios. By asking questions like, 'What happens if a critical service becomes unavailable?' or 'What if user growth exceeds projections?', teams can identify assumptions about system behavior under stress. This approach reveals potential vulnerabilities—such as assumptions about scalability, third-party service availability, or network reliability—and helps teams anticipate failure modes that might otherwise be overlooked.

Beyond identifying obvious issues, scenario planning prepares teams for less likely but high-impact events. This proactive strategy enables the discovery of assumptions that may not be apparent during routine design but could lead to catastrophic failures if left unexamined.

Design for Failure

Building systems with the expectation that failures will occur forces teams to uncover hidden assumptions about system resilience. Chaos engineering, a practice popularized by Netflix's 'Chaos Monkey,' introduces failures—such as simulated network outages or service downtimes—to test how the system responds. These controlled failures expose assumptions about fault tolerance and recovery mechanisms, revealing weak points that might otherwise go unnoticed.

By designing systems to withstand and recover from failures, teams can surface implicit assumptions about system reliability, external dependencies, and architectural weaknesses. This approach ensures that failure scenarios are not only considered but actively tested, reducing the risk of widespread system outages.

Documenting, Tracking, and Ownership of Assumptions

To effectively uncover and manage hidden assumptions, explicit documentation is essential. By recording assumptions—whether they are technical, business-related, or security-focused—teams make these assumptions visible and trackable. This prevents assumptions from remaining implicit, ensuring that they are regularly reviewed as the system evolves.

Maintaining an assumption log or including assumptions in architectural decision records (ADRs) helps ensure accountability. Assigning ownership of specific assumptions further reduces the risk of them going unexamined. By designating responsibility for validating assumptions at key milestones, teams can challenge and update them as new information becomes available, minimizing hidden risks throughout the project lifecycle.

Validating Assumptions Through Testing

Testing is critical to validating assumptions and ensuring that the system behaves as expected under real-world conditions. Different testing strategies can help challenge assumptions:

- **Stress Testing**: This method tests system performance under extreme conditions, helping teams validate assumptions about scalability and capacity. By pushing the system beyond normal operating parameters, stress testing can reveal hidden assumptions about how much load the system can handle without failure.
- **Chaos Engineering**: Similar to the practice mentioned earlier, chaos engineering introduces controlled failures into the system to test fault tolerance. By simulating outages or disruptions, teams can challenge assumptions about system resilience and validate recovery mechanisms in a live environment.

Testing reveals flaws in even the most stable assumptions. And when optimism goes unchecked, risks can easily be overlooked. Regular testing keeps these assumptions in check, ensuring the system remains resilient as it grows and adapts to new challenges.

Regular Checkpoints

Establishing regular checkpoints throughout the project lifecycle—such as sprint reviews or milestone meetings—creates opportunities to revisit and validate assumptions. During these checkpoints, teams should explicitly review documented assumptions to ensure they still hold true and align with the evolving system needs. This ongoing validation process prevents assumptions from becoming outdated or irrelevant, ensuring that they remain accurate as the project progresses.

By integrating assumption reviews into the project's workflow, teams ensure that hidden assumptions are surfaced early, before they lead to major issues. Regular checkpoints also encourage proactive communication, ensuring that assumptions are shared and validated across all relevant stakeholders.

By employing techniques such as scenario planning, designing for failure, and rigorous testing, teams can surface hidden assumptions before they have the chance to disrupt system design. These methods not only bring unexamined assumptions to light but also allow teams to proactively mitigate risks before they escalate. However, identifying these assumptions is only the first step. For them to be managed effectively, they need to be documented clearly and systematically. Without proper documentation, assumptions can quickly fall back into obscurity, leaving teams vulnerable to the same risks. Documenting assumptions is crucial not only for minimizing risks but also for improving communication, maintaining accountability, and ensuring that assumptions can be revisited and validated as the system evolves.

5 Importance of Documenting Assumptions

Keeping track of assumptions is essential in system design. Assumptions, especially the ones that are implicit or go unspoken, can lead to significant issues if they prove to be wrong. By explicitly recording them, teams create transparency and enable continuous review and validation as systems evolve. This ensures that decisions are built on solid, validated information, helping to avoid hidden risks that could otherwise undermine the integrity of the system.

Mitigating Risk

Assumptions that go unexamined or unstated introduce serious risks if they turn out to be incorrect. When teams document their assumptions, they create the opportunity to regularly review and challenge them, reducing the likelihood of unexpected system behavior or design failures. Having this clear documentation prevents teams from relying on potentially flawed assumptions and ensures the system is designed with realistic constraints in mind.

Additionally, documenting assumptions helps mitigate the risk of cascading failures—where one unchecked assumption triggers larger system-wide issues. By consistently reviewing assumptions at key project milestones, teams can prevent these minor issues from snowballing into major, costly problems. This practice builds resilience, ensuring that the system can handle changes in behavior or external factors before they become critical.

Improving Communication

Good documentation of assumptions plays a key role in improving communication across teams and stakeholders. When assumptions are clearly recorded and shared, it ensures that everyone involved—from developers and QA teams to operations and business analysts—is on the same page regarding the system's limitations, capabilities, and expected behavior. This transparency helps to avoid misunderstandings and misalignments, which are common in complex projects involving multiple teams.

Cross-team reviews also benefit from clear documentation. They help surface hidden risks that one team might miss but another might recognize. While developers may assume that scaling hardware will solve performance issues, the operations team may know of infrastructure limitations that make such scaling unfeasible. Regular assumption reviews with all relevant teams involved help challenge assumptions from different perspectives, reducing the chance of overlooked risks.

Additionally, when teams collaborate across functions, they improve communication and reduce cognitive biases like confirmation bias or overconfidence that can skew decision-making. Regularly reviewing and validating assumptions ensures that everyone has a clear, up-to-date understanding of the system's requirements and potential risks. This not only leads to better decision-making but also aligns technical and business goals, strengthening the overall system design.

Supporting Future Changes

As systems evolve, so do the environments in which they operate—whether it is user behavior, external dependencies, or regulatory factors. Documenting assumptions ensures that teams can easily revisit and reassess them when the system requires updates or when circumstances change. This documentation acts as a guide, helping teams quickly identify which assumptions are still valid and which need adjustment.

Well-documented assumptions serve as a living record, providing continuity as new team members join or the system scales. This becomes especially important when external factors like regulatory changes or new technology standards come into play, giving teams a clear basis for reassessing their design. As the system's environment shifts, these documented assumptions help guide informed decision-making during future updates.

Creating Accountability

Documenting assumptions doesn't just create transparency—it also adds accountability to the system design process. By linking assumptions to specific individuals or teams, it is clear who is responsible for validating and reassessing them. This accountability ensures that assumptions are not neglected or left to become outdated, reducing the risk of hidden assumptions lingering within the system.

When assumptions are clearly assigned to stakeholders, it is easier to hold individuals accountable for ensuring those assumptions remain valid as the project evolves. This accountability fosters a culture of vigilance, where assumptions are regularly revisited rather than left unexamined. This proactive approach strengthens the integrity of the system, lowering the chances of failure due to outdated or incorrect assumptions.

By consistently documenting assumptions, teams ensure they are equipped to adapt to evolving circumstances and changes in the system's environment. This proactive approach not only reduces risks but also creates a solid foundation for making informed decisions as new challenges arise. However, when assumptions—especially critical ones—are left undocumented or unexamined, the consequences can be severe and far-reaching.

To highlight the dangers of neglecting assumption management, I will examine a real-world example where these failures led to disastrous outcomes. The NASA Mars Climate Orbiter mission provides a stark reminder of what can happen when a single, unverified assumption goes unnoticed. This case study vividly demonstrates why properly documenting and validating assumptions is essential for the success of complex projects.

6 Case Study: The Mars Climate Orbiter

In 1999, NASA's Mars Climate Orbiter mission failed because of a simple yet devastating error: a mix-up in units of measurement. The spacecraft, which was meant to study Mars' atmosphere, entered the planet's atmosphere at the wrong altitude and burned up. An investigation found that the root cause[2] was a mismatch in units: NASA worked in metric units, while Lockheed Martin, the contractor responsible for building the spacecraft, used English units in software for trajectory calculations [Mar99]. This fundamental oversight in communication and documentation between the teams led to a fatal trajectory miscalculation.

This assumption about shared unit systems—left unverified—became a critical point of failure. The mismatch went unnoticed until it was too late, resulting in the spacecraft's destruction. The failure emphasizes the importance of thoroughly validating assumptions in system design, especially in large, multi-team projects where collaboration and communication are essential. In this case, the absence of proper assumption management, communication protocols, and validation processes directly contributed to the mission's failure.

Cognitive Biases

The failure of the Mars Climate Orbiter was not only technical but also deeply rooted in human factors, with cognitive biases playing a critical role in allowing unverified assumptions to persist. At the Jet Propulsion Laboratory (JPL)[3], a 30-year history of successful interplanetary navigation had fostered a belief that 'Orbiting Mars is routine' [Mar99]. This long-standing success led to overconfidence bias, where the navigation team assumed that their processes—proven over decades—would continue to work flawlessly without significant oversight. This belief reduced attention to risk mitigation, as the team did not critically question or validate their assumptions.

[2]According to NASA Procedures and Guidelines (NPG) 8621 Draft 1, a [dominant] *root cause* is defined as: 'Along a chain of events leading to a mishap, the first causal action or failure to act that could have been controlled systematically either by policy/practice/procedure or individual adherence to policy/practice/procedure.' [Mar99]

[3]JPL, a NASA field center, managed the Mars Climate Orbiter mission operations and navigation; Lockheed Martin designed and built the spacecraft.

Another significant factor was optimism bias, where managers clung to assumptions of mission success despite warning signs of trajectory discrepancies. Instead of rigorously questioning whether conditions had strayed into untested territory, decision-makers assumed that the mission was proceeding as expected and placed the burden on engineers to prove otherwise [Obe99]. This bias led to a distortion in risk perception, allowing flawed assumptions to persist unchecked.

The investigation report recommended a cultural shift at JPL, urging personnel to 'question and challenge everything—even those things that have always worked' [Mar99]. This recommendation underscores how cognitive biases can influence system design by allowing hidden risks and unexamined assumptions to remain unnoticed until they lead to catastrophic failure. By addressing these biases proactively, teams can reduce the likelihood of costly errors and ensure more robust system validation practices.

Cross-Team Communication and Hidden Assumptions

The failure of the Mars Climate Orbiter was exacerbated by a breakdown in cross-team communication, which allowed critical assumptions to slip through without being questioned. The investigation revealed inadequate communication between key project elements, including development, operations, navigation, and project management teams.

> [T]here is evidence of inadequate communications between the project elements, including the development and operations teams, the operations navigation and operations teams, the project management and technical teams, and the project and technical line management. [Mar99]

This lack of communication led to the implicit assumption that both teams were using the same units of measurement—an error that went unnoticed until it was too late. The failure to communicate trajectory concerns across teams left this critical assumption unchecked, contributing directly to the mission's failure. As the investigation board noted:

> [i]t was clear that the operations navigation team did not communicate their trajectory concerns effectively to the spacecraft operations team or project management. [Mar99]

A formal process for sharing and validating assumptions between teams could have caught this error early. Ensuring that key assumptions—such as unit systems—are clearly communicated, explicitly stated, and validated throughout the project lifecycle is crucial. In complex, multi-team projects, fostering robust cross-functional communication and maintaining clear documentation of assumptions are essential steps to prevent similar failures in the future.

Importance of Documentation

A key element in managing assumptions is thorough documentation. Without clear records and consistent communication of assumptions, oversights remain hidden, increasing the risk of failure. The Mars Climate Orbiter disaster starkly illustrates how the lack of proper

documentation can lead to catastrophic outcomes. Had the units of measurement been properly documented and reviewed by both NASA and Lockheed Martin, the error that led to the spacecraft's destruction could have been avoided. Instead, the team relied on informal communication methods:

> [w]hen conflicts in the data were uncovered, the team relied on e-mail to solve problems, instead of formal problem resolution processes such as the Incident, Surprise, Anomaly (ISA) reporting procedure. [Mar99]

By explicitly recording assumptions, teams ensure that these assumptions can be revisited and validated at every stage of the project lifecycle. A thorough review of such documentation could have identified the units mismatch early, preventing the fatal error. Proper documentation also helps to surface discrepancies and prevents them from being missed due to over-reliance on informal communication.

> [T]he discipline of documenting all concerns on problem reports is paramount for spacecraft teams in flight operations—to make sure nothing 'falls through the cracks'. This was not rigorously applied on Mars '98. [EJC01]

Clear, structured documentation is essential for managing assumptions effectively. It provides a safety net by revealing potential inconsistencies early in the process, preventing hidden assumptions from leading to significant system failures later. Ensuring that all assumptions are formally documented allows teams to maintain accountability, track risks, and ensure that no critical details are overlooked.

Validating Assumptions Through Testing

Even when assumptions are thoroughly documented, they must still be rigorously tested to ensure their validity. One of the key factors behind the Mars Climate Orbiter failure was insufficient verification and testing of critical assumptions. The investigative report highlights:

> End-to-end testing to validate the small forces ground software performance and its applicability to the specification did not appear to be accomplished. […] The interface control process and the verification of specific ground system interfaces was not completed or was completed with insufficient rigor. [Mar99]

This lack of comprehensive testing meant that key assumptions—such as the use of metric units in trajectory calculations—were never identified or corrected. Without rigorous testing, the assumptions embedded in the system's design remained unvalidated, leading to catastrophic failure. Testing, especially under real-world conditions, is critical to ensure that all components interact correctly and that assumptions about inputs, outputs, and system behavior hold true.

Comprehensive testing methods, such as end-to-end testing and interface validation, are necessary to verify that assumptions align with the actual system design. Skipping or inadequately conducting these tests allows dangerous assumptions to persist, as demonstrated by the Mars Climate Orbiter failure. For assumption management to be effective, validation through testing is as crucial as the documentation itself. Without this step, even well-documented assumptions can undermine system reliability if they are not properly verified.

Scenario Planning

In addition to testing, proactive planning for potential failures is essential. The Mars Climate Orbiter mission lacked comprehensive scenario planning, which left the team unprepared for unexpected events. The investigation observed[4] that the mission team did not systematically analyze what could go wrong and failed to employ standard techniques such as fault tree analysis to evaluate potential failure points [Mar99]. This lack of analysis contributed to the mission's downfall, as the team did not anticipate or prepare for critical failure modes.

Had fault tree analysis been performed, the units mismatch could have been flagged as a potential failure point, underscoring the importance of thorough assumption verification. Systematically analyzing potential risks allows teams to prepare for unexpected events and ensures that hidden assumptions are surfaced and addressed before they become project risks.

Effective scenario planning requires anticipating potential issues and failure modes well in advance of mission-critical events, ensuring that the entire team is prepared for unexpected circumstances. The investigation noted that the operational team lacked clear, well-defined contingency plans, and not all members fully understood the decision criteria required to execute critical maneuvers.

> Inadequate contingency planning for TCM-5[5] was observed to play a part in the MCO failure. The MCO operational contingency plans for TCM-5 were not well-defined and or[6] completely understood by all team members on the MCO operational team. [Mar99]

The failure to establish a clear set of Go/No-Go criteria or review the evaluation and decision-making processes before committing to TCM-5 demonstrates how the absence of systematic planning can lead to disastrous results. Scenario planning is crucial for surfacing hidden assumptions about system behavior and ensuring that teams are prepared to make informed decisions when faced with unexpected challenges.

[4]According to NASA Procedures and Guidelines (NPG) 8621 Draft 1, a [significant] *observation* is defined as: 'A factor, event or circumstance identified during the investigation which was not contributing to the mishap, but if left uncorrected, has the potential to cause a mishap [...] or increase the severity should a mishap occur.' [Mar99]

[5]Trajectory Correction Maneuver 5 was a planned adjustment to the Mars Climate Orbiter's trajectory intended to refine its course as it approached Mars. It was meant to ensure that the spacecraft would enter the correct orbit around Mars. However, TCM-5 was never executed, as the mission team deemed it unnecessary.

[6]Original language preserved.

Designing for Failure

Scenario planning should always be paired with robust system designs that can withstand failure. This approach emphasizes building systems and processes that are resilient, even when faced with unexpected challenges. In the Mars Climate Orbiter mission, the lack of end-to-end contingency testing and insufficient preparation for potential failures played a significant role in the mission's failure.

The NASA investigation stressed the importance of not only developing contingency plans but also rigorously testing these plans and training operational teams on their execution. As the report noted:

> Contingency plans need to be defined, the products associated with the contingencies fully developed, the contingency products tested and the operational team trained on the use of the contingency plans and on the use of the products. [Mar99]

By testing contingency plans and training teams to respond effectively to failures, organizations can reduce the impact of unexpected events. A design-for-failure mindset ensures that systems are built to recover swiftly, even in the face of failures, and helps surface hidden assumptions about system reliability and recovery mechanisms.

Incorporating this mindset allows system designers to build resilience directly into their processes, mitigating the risks associated with unexamined assumptions. Without proper scenario planning and design for failure, teams miss critical opportunities to validate assumptions under stress, leaving systems vulnerable to failures—such as the Mars Climate Orbiter disaster.

The Cost of Unexamined Assumptions

The Mars Climate Orbiter disaster starkly illustrates the high cost of failing to systematically manage assumptions in complex systems. Even minor, unchecked assumptions can lead to catastrophic consequences. In this case, NASA lost a $125 million spacecraft [Llo99] and years of valuable scientific research. As Thomas Gavin, deputy director for space and earth science at NASA's Jet Propulsion Laboratory, noted, 'A single error should not bring down a $125 million mission' [Obe99]. Yet, the destruction of the Mars Climate Orbiter was ultimately preventable and resulted from a failure to document, communicate, and validate a basic assumption about the units of measurement.

According to project manager John Stephenson, the rush to get the small forces model operational led to abbreviated testing. 'Had we done end-to-end testing [...] we believe this error would have been caught,' he admitted [Obe99]. This highlights how even seemingly minor assumptions can go unvalidated under tight deadlines, leading to severe consequences.

The disaster serves as a powerful reminder that no assumption is too small to verify, and no potential failure is too unlikely to plan for. A simple mismatch in measurement units led to the loss of a high-profile mission, showing that without proper communication and

documentation, even minor assumptions can escalate into catastrophic outcomes. The failure to validate such assumptions wasted not only financial resources but also time, scientific opportunities, and public confidence.

Effective management of assumptions at every stage of system design is critical. Documenting assumptions, facilitating clear cross-team communication, and conducting rigorous testing must be foundational practices in any complex system. Without this level of diligence, even small oversights can lead to devastating outcomes, as demonstrated by the Mars Climate Orbiter failure.

The Mars Climate Orbiter disaster is a vivid example of the profound consequences that can result from a failure to document, communicate, and validate assumptions. A seemingly minor oversight—left unexamined—escalated into a catastrophic failure, highlighting the critical importance of proactive assumption management. This case underscores the need for clear documentation, rigorous testing, and effective communication at every stage of system design.

The lessons from this failure tie directly back to the broader challenge of building resilient systems. Managing assumptions more effectively can enhance system resilience, reduce failures, and ensure that teams are better equipped to handle uncertainty in complex projects.

7 Conclusions: Assumptions and System Resilience

NASA's Mars Climate Orbiter failure is a striking example of how unchecked assumptions can derail even the most advanced systems. Throughout this article I have explored how proactive assumption management—through documentation, testing, and communication—can significantly reduce the risks posed by hidden assumptions. The downfall of the Mars Climate Orbiter, caused by a simple, unverified assumption about measurement units, shows just how easily small, unnoticed errors can snowball into catastrophic consequences. This case reinforces the need for diligence in managing assumptions at every stage of system design.

Cognitive biases, such as overconfidence and optimism, complicate assumption management further by distorting decision-making. These biases can lead teams to overlook critical risks, miss early warning signs, or fail to communicate effectively. To manage assumptions effectively, teams need more than just technical solutions—they must also address the human factors that influence decision-making. If left unchecked, flawed assumptions can persist and result in costly mistakes down the line.

As Jesse Robbins insightfully said in his talk *GameDay: Creating Resiliency Through Destruction*: 'You don't choose the moment, the moment chooses you. You only choose how prepared you are when it does' [Robi1]. This captures the heart of system resilience: while failures are often unpredictable, teams can control their level of preparedness. By rigorously testing assumptions, building fault-tolerant systems, and fostering open communication, teams can better handle challenges when they arise.

The key takeaway is that assumption management requires a systematic, ongoing effort. Even with thorough documentation and testing, some assumptions may still go unnoticed—especially in complex systems with many interdependencies. As systems evolve, new hidden assumptions will inevitably emerge, meaning that assumption management practices must continuously adapt.

Looking ahead, there is significant potential for improving assumption management with advances in system monitoring and AI-driven diagnostic tools. Additionally, organizations can foster a culture that encourages team members to challenge assumptions and openly discuss concerns. By continuously refining these practices, teams can build systems that are not only reliable but also resilient in the face of uncertainty.

Managing assumptions is essential for building systems that can withstand unforeseen challenges. Regular assumption check-ins, clear documentation, and proactive planning—such as scenario planning and designing for failure—are critical strategies for minimizing the risks that hidden assumptions pose. By adopting these practices, system designers can create resilient, scalable systems that are better prepared for both expected and unexpected events, ultimately improving performance and reliability.

References

[Cor] Cornell Law School, Legal Information Institute. *Assumption*. URL: https://www.law.cornell.edu/wex/assumption (visited on 07/10/2024).

[EJC01] Euler, Edward A., Jolly, Steven D. and Curtis, H. H. 'Lad'. 'The Failures of the Mars Climate Orbiter and Mars Polar Lander: A Perspective from the People Involved'. In: 24th Annual AAS Guidance and Control Conference. American Astronautical Society, 2001.

[Kee17] Keeling, Michael. *Design It!: From Programmer to Software Architect*. The Pragmatic Bookshelf, 2017.

[Llo99] Lloyd, Robin. *Metric Mishap Caused Loss of NASA Orbiter*. CNN. 30th Sept. 1999. URL: https://web.archive.org/web/20191024152139/http://www.cnn.com/TECH/space/9909/30/mars.metric.02/index.html (visited on 02/10/2024).

[Mar99] Mars Climate Orbiter Mishap Investigation Board. *Mars Climate Orbiter Mishap Investigation Board: Phase I Report*. NASA, 10th Nov. 1999.

[Nic98] Nickerson, Raymond S. 'Confirmation Bias: A Ubiquitous Phenomenon in Many Guises'. In: *Review of General Psychology* 2.2 (June 1998).

[Obe99] Oberg, James. 'Why the Mars Probe Went off Course [Accident Investigation]'. In: *IEEE Spectrum* 36.12 (Dec. 1999).

[Rob11] Robbins, Jesse. 'GameDay: Creating Resiliency Through Destruction'. LISA '11: 25th Large Installation System Administration Conference (Boston, MA). 7th Dec. 2011.

[Tal05] Taleb, Nassim. *Fooled by Randomness: The Hidden Role of Chance in Life and in the Markets*. Random House, 2005.

[TK74] Tversky, Amos and Kahneman, Daniel. 'Judgment under Uncertainty: Heuristics and Biases'. In: *Science* 185.4157 (27th Sept. 1974).

[Wei80] Weinstein, Neil D. 'Unrealistic Optimism about Future Life Events.' In: *Journal of Personality and Social Psychology* 39.5 (Nov. 1980).