



The Harmonics of Usability: A Quartet of Implications for Software Interface Design

Adele Sommers*, Ph.D.

President, Business Performance Inc. United States

*Corresponding Author

Adele Sommers, President, Business Performance Inc.
United States, E-mail: adele.sommers@gmail.com

Citation

Adele Sommers (2025) The Harmonics of Usability: A Quartet of Implications for Software Interface Design. J Artif Intell Syst Appl 1: 201

Publication Dates

Received date: August 06, 2025

Accepted date: August 23, 2025

Published date: August 30, 2025

Abstract

The Harmonics of Usability: A Quartet of Implications for Software Interface Design explores how usability in software systems can be orchestrated to better serve real-world user needs. Drawing on the foundational ideas of three masterful “conductors” — Thomas Gilbert, John Bowie, and Genichi Taguchi — this article proposes a customer-centered approach to interface design that distinguishes between primary user goals (Job 1) and secondary, system-imposed tasks (Job 2).

With the advent of AI as a fourth “conductor,” the article examines how AI can help reduce cognitive burden, surface user pain points, and support more intuitive system workflows. A set of practical recommendations and case examples illustrates how design teams can collaboratively use AI to emphasize Job 1, simplify or eliminate Job 2, and elevate user satisfaction. The article argues for a shift away from treating usability as an afterthought and toward building clarity, performance support, and contextual intelligence into systems from the outset.

Keywords: A/B testing; Artificial intelligence (AI); Context-sensitive help (or contextual guidance); Domain knowledge; Human-centered design; Job 1; Job 2; Performance support; Personas; Scenario; Usability; Use case; User experience (UX); User interface; Workflow

Introduction

In the world of usability, Thomas Gilbert, Genichi Taguchi, and John Bowie — experts in human performance, quality engineering, and information engineering, respectively — are singing three-part harmony. Despite their distinct domains and generational differences, their perspectives converge in ways that suggest they could jointly conduct the entire orchestra of software development. This article examines the contributions each individual has made, directly or indirectly, to the domain of software usability. It considers how planning factors, design criteria, and information presentation can profoundly affect the experiences that customers routinely have with software. We'll draw from their philosophies to explore how their ideas apply to user experience (UX) design. Toward the end, we'll also introduce a fourth conductor: artificial intelligence. Incorporating AI advice into interface design can

bring the entire classical experience into modernity.

We'll examine AI's role in supporting Bowie's ideas about primary and secondary user tasks when interacting with software — a structured approach that bridges the gap between user-centered and system-centered design objectives. This pragmatic, people-focused framework draws a sharper distinction between what users want to do and what systems expect them to do, and explores how AI can act as a mediator. It's a perspective that appears to be underrepresented in the current AI/UX literature. Through this lens, all members of a software team, including technical communicators, can unite around shared priorities for improving customer experiences and business outcomes. From these foundations, four specific recommendations emerge. These are practical strategies that can help teams reconcile the often-conflicting demands of complex systems and the real people who use them.



Figure 1: Our three master conductors

Gilbert's Libretto

In his masterwork, *Human Competence: Engineering Worthy Performance* (1996), Gilbert emphasizes that adapting environments to people is far less costly than trying to modify people to suit their environments. Carefully designing and op-

timizing tools and working conditions requires much less effort than asking people to learn new skills.

A pioneer in human performance, Gilbert notes that while training can be effective, it's also expensive. If tools and other environmental variables can be designed to support perfor-

mance, it reduces or eliminates the need for training. He comments, “It is well to make sure that we don’t end up training people to use tools that could be redesigned, or to memorize data they don’t need to remember, or to perform to standards they are already capable of meeting and would meet if they knew what these standards are” (p. 91). Instead, he argues, we should reduce environmental complexity so people can succeed with minimal instruction.

Training, Gilbert explains, requires deliberate structure — realistic practice, job aids such as checklists, and sustained reinforcement. Without these, learning is fragile, transient, and ephemeral. He further observes that training is best reserved for closing genuine knowledge gaps that remain after environmental changes — such as systematically removing obstacles to success — have been exhausted.

Given how quickly knowledge fades and how frequently workers turn over, organizations should focus first on improving environmental variables. Well-designed systems reduce training needs, lower costs, and boost morale.

In short: Optimize the environment first. If people can succeed without having to expand their knowledge, everyone benefits.

Bowie's Aria

From the standpoint of information design, as Bowie asserts in his insightful article, *Customer-Driven Project Management* (2003), the singular goal of software should be helping people do their primary jobs. Primary jobs consist of the functions people were performing, or trying to perform, before turning to the software for assistance. Bowie refers to those activities as Job 1. They consist of the main roles and responsibilities that people carry out in fields such as accounting, nursing, forestry, and graphic arts, for example.

By enabling people to perform Job 1 efficiently and effectively, software can help us achieve something far better, faster, or cheaper than we can otherwise accomplish on our own.

In contrast, everything else that software requires us to do is what Bowie labels Job 2. This is new work created by the software itself, such as installing, setting up, configuring, learning, troubleshooting, and maintaining the system. In Bowie’s view, any Job 2 task represents a waste, as it distracts us from accomplishing our primary Job 1 mission.

System designers, Bowie notes, must decide which operations the hardware will perform, which operations the software will perform, and which tasks will be relegated to people — such as employees or customers. Ideally, this process should aim to reduce or eliminate the cognitive burden on humans. Too often, however, a lazy system design process hands the users unnecessary, time-wasting Job 2 tasks that could have been more elegantly incorporated elsewhere.

As Bowie posits, “Far more common, however, are products that lead their customers down a circuitous path to get to the Job 1 Result. These products construct roads full of dangerous curves, detours, dead ends, and obstacles through which the customer must navigate to reach Job 1. If the road becomes too difficult and time-consuming, the customer will simply turn back and find another road (product). Even if they stay the course and reach the Job 1 Result, chances are high that many customers will avoid using the same road again and will advise others to find an alternate route.”

In his article, *AI + UX: Design for Intelligent Interfaces*, Basu (2025) echoes Bowie’s sentiment by noting, “The average user is going to be pretty skeptical, disoriented even, when they first start using your product. We want to make sure our users trust the product and its capabilities, feel confident using it, understand how it works (to a reasonable extent), and are able to interpret the output effectively. We don’t want users to necessarily need a tutorial or crash course to use the product (unless it’s really that specialised). We also don’t want them to have to verify the results through Google, ChatGPT, or another ‘better’ tool.”

As summarized in Figure 2, Bowie’s insights underscore the need to build systems that streamline core tasks while minimizing frustrating busywork. Otherwise, the systems risk undermining the very value they were supposed to provide.

Early Rehearsals

In the early days of computing, programmers were the principal users. People with expert technical knowledge painstakingly performed the actions that today’s computers make largely transparent. Complex logic circuits required advanced knowledge and detailed instructions for every task. During that era, computer use was a daunting and highly specialized endeavor.

Over time, as hardware and software grew more sophisticated, the burden shifted from people to machines — making today's computers far easier for everyday users to operate. In any given domain, knowledge previously held by just a few specialists can be embodied directly into the system. This kind of transformation enables people with far less technical

and subject matter expertise to access a wide variety of newly available information.

Those benefits accrue, however, only when the systems that embody the now-accessible knowledge are user-friendly enough to help people fulfill their Job 1 goals.



	 JOB 1 (PRIMARY TASKS)	 JOB 2 (SYSTEM BURDENS)
Definition	The users' real-world goals and responsibilities in a given field.	Secondary, overhead tasks that a system imposes on its users.
Examples	<ul style="list-style-type: none"> ▪ Filing an insurance claim ▪ Creating a new monthly budget ▪ Planning enterprise resources 	<ul style="list-style-type: none"> ▪ Installing & setting up software ▪ Troubleshooting errors or bugs ▪ Looking up data entry codes
Value to the user	High value. Directly supports the users' actual work or learning.	Low value. Distracts and wastes users' time with busywork tasks.
Design approach	Identify, streamline, and embed all applicable domain knowledge.	Eliminate, automate, or conceal all busywork from the users.
AI's role	Model the core actions to perform a job, then optimize workflows.	Identify friction points, recommend simplifications, measure usability.

Figure 2: Job 1 vs. Job 2 – A tale of two missions

Waiting for the Curtain to Rise

Today's technological trends should continue shifting the burdens from customers to systems. Software should know how to transparently install, configure, troubleshoot, and maintain itself with little or no input from users. It should also intelligently provide us with just-in-time support for Job 1. Failure to ease customers' Job 2 burdens can result in undesirable business consequences for manufacturers.

Job 2 annoyances could become all but extinct if manufacturers learn from their design mistakes. As Bowie observes (2003), "It takes great courage for project managers and company executives to make Job 1 the #1 project objective. Several years ago, Western Digital demonstrated their commitment to Job 1 by delaying shipment of a new line of consumer hard disk drives when they discovered customers were likely to fail in the installation process, a Job 2 requirement. . . . While disastrous from a revenue perspective, the decision to hold the

drives until the fatal flaw in the customer design could be fixed saved millions in support and warranty costs and avoided a customer loyalty meltdown."

Yet in reality, Job 2 burdens still exist in many products and systems. Software design has not universally evolved to eliminate Job 2. Developers often conform to existing norms instead of challenging them. Whether we're dealing with sprawling ERP (Enterprise Resource Planning) systems or small software apps, our collective mindset seems to dictate that software systems follow familiar patterns. We therefore continue to accept Job 2 as an inevitable hassle.

Octaves of Distraction

Even for experienced users, few interfaces are fully self-explanatory, so some kind of support is necessary. Figure 3 shows a spectrum of assistance strategies, from low-disruption aids like tool tips and embedded instruction to high-disruption options like classroom training.

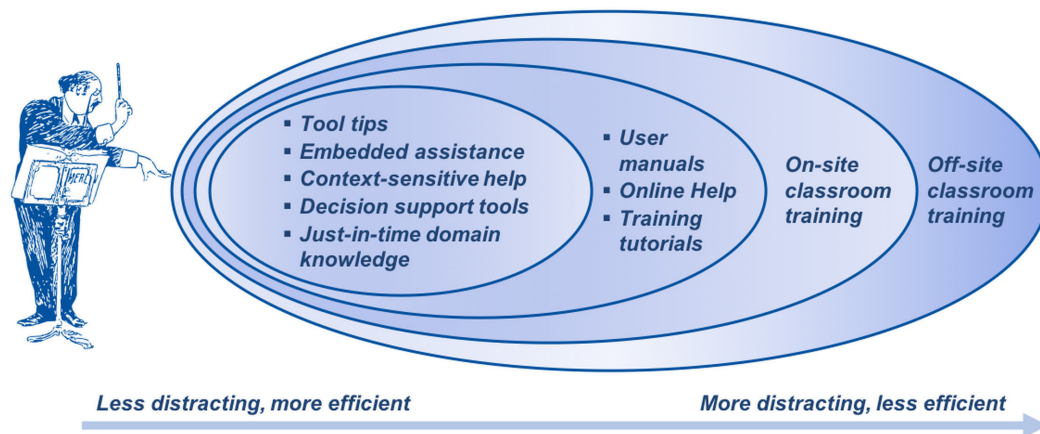


Figure 3: A spectrum of user assistance options

This figure shows that the least distracting methods are also the most efficient. The options on the left side demand far fewer interruptions of attention than those on the right. That's because of their proximity to the point of need and the degree of integration with the task at hand. For example,

- **More ideal support** includes information and devices that are highly relevant and accessible within the interface. Those types include tool tips, context-sensitive help, embedded assistance, decision support tools, and just-in-time domain knowledge.
- **Less ideal support**, even if relevant to the task, requires more steps and breaks in concentration. Those types include the guidance found in user manuals, online help, training tutorials, and classroom training (either on- or off-site). Off-site training can be especially disruptive as it usually involves some kind of logistics and scheduling, yet it doesn't necessarily improve performance.

Economic Cacophony

From a business standpoint, Job 2 is nothing but overhead. Every minute users spend on system installation, setup, troubleshooting, or training is a minute not spent carrying out their primary roles. The only financial winners are the software vendors — unless Job 2 becomes so frustrating that it affects sales — and the third parties who are traditionally relied upon to close the gap between the system and the user.

For instance, technical communicators, trainers, and support centers often serve as cognitive first responders. They patch

flawed designs with user manuals, tutorials, and troubleshooting support for inadequate features or system bugs.

Out of necessity, the training industry remedies software weaknesses by designing courses that supply missing information and teach users known workarounds for system idiosyncrasies. This constant need for help represents the cost of not designing for simplicity in the first place.

Product creators ignore all of these concerns at their peril. A company's bottom line suffers if it fails to heed the lessons emerging from its customer complaint database or from increasing customer returns.

Taguchi's Tune

These costs aren't just internal. Enter Genichi Taguchi, whose timeless work on measuring product quality appears in *Taguchi Techniques for Quality Engineering* (Ross, 1996). His model illustrates how deviations from optimal quality cause economic harm — not just to individual users but to society. The mechanics of the model are most powerfully summarized in simple language: The farther product or service performance strays from perfection (where the producer's and customers' needs are balanced), the greater the negative impact on society, from inconvenience to large-scale failure.

Taguchi's insight lies in showing how flaws multiply beyond individual costs. As Ross explains, if a thief steals \$10, society's net loss is zero — the victim loses and the thief gains. But if a product's defects cost the customer more than the company saves, the result is worse than theft, as it imposes an uncompensated cost on society. These ripple effects can escalate

quickly, dramatically, and often invisibly, leading to damage far greater than a single failed transaction.

Such negative consequences vividly illustrate the underlying philosophy of the Taguchi loss function, which measures the economic losses to society caused by poor-quality products and services. To remedy this pervasive problem, Taguchi's model champions a relentless variability prevention and reduction strategy. Two key tenets of this strategy emphasize:

1. Identifying and categorizing factors that contribute to variability. One category comprises control factors that can be managed, such as material type, temperature, and pressure. In contrast, noise factors are often too difficult or expensive to control during production or operation, such as user proficiency, environmental conditions, or component inconsistencies.

2. Designing products to perform consistently and effectively even when noise factors are present. Rather than trying to eliminate all sources of variation — which may be impractical or prohibitively expensive — the design process should focus on minimizing their impact. This principle lies at the heart of robust design, which reduces the vulnerability to failure under unpredictable conditions.

To evaluate these factors systematically, Taguchi also advocated an iterative design-of-experiments (DOE) methodology. This technique uses structured experiments to test which design elements influence outcomes the most, and under what conditions.

Although Taguchi's work focused on non-software products, the ripple effects he identified are just as real in software systems

Applying his philosophy to software, a variability prevention and reduction approach should revolve around designing systems that are, once again, less vulnerable to noise factors, such as user errors, environmental challenges, or internal inconsistencies. For example, the design strategy should aim to:

- Optimize user interfaces to handle a wide range of user inputs — including incomplete, ambiguous, or extreme values — as well as different device configurations and accessibility needs.

- Design test cases and scenarios that intentionally vary conditions to surface weaknesses. These might include novice and expert users, intermittent connectivity, or entering atypical or unexpected data values. Even a simplified form of DOE can help expose which interface elements lead to user confusion or error.
- Minimize variability in performance by anticipating how different usage patterns or data flows will affect system stability or output quality.
- Build resilient system architectures that allow for graceful recovery when failures do occur, whether due to resource limitations, power outages, or corrupted inputs. This also means simplifying the architecture wherever possible and resisting the temptation to layer on complex features that are difficult to test or validate.
- **Complex systems** — especially those with bespoke components or custom configurations — can eventually reach a tipping point, where the number of variable combinations overwhelms the team's ability to test them thoroughly. As variability grows, so does the risk that defects will escape detection until they cause costly failures in the field. That's why the software development process should embody a design-for-testability philosophy — keeping testability front and center as a constraint on what can realistically be accomplished in any given system.

To apply Taguchi's model to software, consider this scenario

You're the plant controller at Acme Widget Company. Over the weekend, you're tasked with installing a business-critical productivity system. The vendor promises an easy, error-free installation if you follow the instructions. But time is tight, since your fiscal year ends in 10 days. After that, support for your current system ends, and all attention will shift to closing the books. If the installation fails, you won't have another production break for two months — missing your window to switch systems efficiently and cost-effectively.

So, you begin the process at 6:00 p.m. Friday. By 10:30 p.m. Sunday, you've hit multiple snags: vague instructions, unex-

plained errors, no off-hours tech support, and historical data has been accidentally overwritten. Now, alone, exhausted, and under pressure, you face three options:

- Option 1: Give up.** Restore Friday night's backup and treat the effort as a total waste. Risk of loss: Most of your weekend, the time you'll ultimately spend on the phone with technical support, and the time required to attempt the installation again two months from now. There are also unforeseeable problems once your current system support ceases in 10 days, plus the disadvantages and setbacks from being unable to start the fiscal year on the new system. Those costs will be significant but cannot yet be quantified.
- Option 2: Hedge.** Wait until 8:00 a.m. Monday to call tech support, hoping to salvage the installation. Risk of loss: Most of your weekend, plus the production downtime while you're finishing the installation, if you're lucky. If you're unlucky, you'll have to proceed as explained in Option 1.
- Option 3: Gamble.** Push forward until 7:00 a.m. Monday, hoping that through trial and error, you'll figure out and correct every problem before the production staff arrives. Risk of loss: The entire weekend, plus production downtime if it appears you can't resolve the outstanding issues and need to contact technical support at 8:00 a.m. Figure 4 helps illustrate the stakes of this option.



Figure 4: The Acme Widget software installation saga

But the biggest danger lies in what happens when you think you've succeeded. Even if the gamble appears to pay off, it could nevertheless introduce subtle data corruption involving:

- Delayed detection.** New anomalies may surface in

billing records after going unnoticed for weeks. Their discovery will require you to reload the last good system backup from the Friday before your weekend installation. Then you'll have to team up with the software vendor's technical support to identify and fix the source of the problem, possibly with the help of an

outside consultant. Several of your office staff will be paid overtime for weeks or months to manually re-enter hundreds — or even thousands — of transactions from hardcopy records.

- **Customer fallout.** Acme may need to smooth out billing errors with customers. Your staff will have a hard time explaining mistakes that have been appearing on their invoices. In fact, several customers become irate about this problem and stop doing business with you. Of course, they won't be referring any new sales your way.
- **Mounting costs.** Your outside network management company will be billing you many extra hours to work alongside the paid consultant to rectify the accumulating series of errors. Ultimately, you'll need three full months to recover. The get-well effort will cost tens of thousands of dollars, not including lost business. And the kicker? The software itself cost only \$4,000 for eight licenses — yet the license agreement holds the vendor harmless for any of these losses.

This disaster shows how a poor interface design can derail projects, alienate customers, drive up costs, and even threaten a company's future. Bargain-priced software can't begin to offset the hidden costs of Job 2 fallout. These nightmare outcomes are even more likely to occur with complex systems that incorporate nonstandard, customer-specific features.

In contrast, our goal should be to eliminate Job 2 dangers and distractions altogether

Well-designed interfaces automate risky tasks like installation and setup, embed verification checks, and prevent errors before they multiply. That means understanding the situations in which users actually work. By researching the range of conditions under which customers will be using our products, we can focus on accommodating most, if not all, critical circumstances.

In short, Taguchi's ideas on reducing variation have a clear corollary in software: when we design with robustness in mind, we minimize the damage caused by unpredictable conditions. Even if we can't anticipate every scenario, we can at least strive to prevent subtle errors from snowballing into catastrophic failures — a lesson the Acme Widget saga unders-

cores all too well.

Recommendations for Transforming Interface Design

The Final Score

As we move into the grand finale, our three master conductors have composed a compelling arrangement with unforgettable lyrics. But will the audience pay to attend opening night?

Software usability — or the lack of it — carries major economic and societal consequences, far beyond the system's initial cost. While some forward-thinking companies embrace interface clarity and simplicity, too many still remain stuck in old patterns. The question is whether we'll continue accepting the dissonance, or finally rewrite the score.

When it comes to transforming software interface design, the traditional key contributors — developers, performance specialists, interface designers, and technical communicators — have long played distinct but complementary roles. Each can help move software systems closer to human-centered ideals by embedding Job 1 domain knowledge, minimizing or eliminating Job 2 burdens, and rethinking documentation.

But now, a fourth conductor has taken the podium: AI

Although not a replacement for human insight, AI offers a new kind of collaboration. For example, AI can amplify each key contributor's strengths by handling labor-intensive tasks. It can surface patterns across sprawling collections of content, and then propose options that balance design criteria with the customers' real-world goals. Whether you're drafting specifications, proposing UI refinements, or revising user guides, AI can help translate Job 1 thinking into tangible outputs — faster and more holistically than before.

An overview of some of the primary AI tools recommended by the UX Design Institute for achieving these goals appears below. AI tools for user experience design typically fall into several useful categories, such as streamlining design, anticipating user needs, or reducing users' cognitive effort. Taken

together, they can help teams produce much more satisfying and productive user experiences. For example:

- Generative design assistants (such as Galileo AI or Uizard) can take written descriptions of a screen or feature and instantly generate visual mockups. This speeds up the process of trying out different design ideas without needing to build each one from scratch.
- User behavior analytics platforms can predict which parts of a screen are likely to draw attention or cause confusion. This helps teams adjust layouts, wording, or placement of key elements so users don't get stuck or distracted. Examples include Neurons or Attention Insight, which analyze user behavior and can also detect user sentiment.
- Large language models (LLMs), such as ChatGPT, can process large amounts of messy or unstructured input — like user interviews, training transcripts, or meeting notes — and turn it into something more usable. Examples include step-by-step task descriptions or working outlines for help content.
- Recommendation engines can personalize user interfaces based on past behavior or peer patterns, helping users reach Job 1 outcomes more intuitively.

Now that we've explored what AI can do, teams of developers, performance specialists, interface designers, and technical communicators may wonder: How can we make a meaningful impact? Below are four practical recommendations that involve the use of AI.

Recommendation #1: Eliminate Job 2 tasks and refocus on Job 1 goals.

If your customers experience grief or dissatisfaction when using your products or systems, your team will want to run down every possible clue. The candid voice of your customer typically resides in your survey data, product reviews, support databases, social media, customer emails, and other communications. AI can help pinpoint user frustrations and then suggest how to resolve them in your next system update.

The UX Design Institute recommends AI tools like MonkeyLearn for sentiment analysis, and Neurons for behavioral

analytics to identify areas where users become bogged down. Those tools can process large volumes of user feedback to surface patterns where Job 2 tasks are overshadowing Job 1. Your team can use that analysis to pursue a more user-prioritized and methodical approach to interface design, which is further explained in the recommendations below. For example, AI might propose additional personas (fictional characters who reflect your actual customers), as well as draft case studies and improved workflows to better address those personas' needs.

Recommendation #2: Rethink and relocate information to where it matters.

If your customers are wading through irrelevant, outdated, or overwhelming system instructions, AI can analyze where users are getting lost, and then recommend whether and how to reformat or relocate that guidance. For instance, AI might propose eliminating any ineffective information altogether, such as by fully automating or integrating it more elegantly into the interface.

Wherever guidance is still necessary, the UX Design Institute suggests using AI-driven analytics platforms like Attention Insight to assess where users are encountering friction within the interface. These tools can generate heatmaps and attention maps, revealing which areas of the UI are overlooked or misunderstood. Using those revelations, the team can relocate critical information closer to the point of need. For example, user documentation could be converted into tooltips or just-in-time, step-by-step, on-screen guidance. This type of transformation reduces the cognitive burden on users, allowing them to become more efficient.

Recommendation #3: Write specifications that reflect real-world scenarios

How well do your products or systems respond to and support your customers' actual circumstances of use? These are the conditions that pertain to when, where, and how your audiences engage with your information, systems, websites, products, or services. For example...

- Routine circumstances comprise the normal or typical modes in which customers consume what you offer, such as at home, in the office, at school, while

exercising, or on the go. Those uses frequently occur in perfectly sunny, non-stressful conditions, with plenty of access to customer support in case anything goes wrong.

- Non-routine circumstances, on the other hand, are atypical, unusual, or even extreme situations in which people might need to engage with your products, services, systems, or documentation. That's when people might be working off hours; in remote locations; outside of wi-fi range; during power outages or bad weather; without sufficient tools, training, or resources; or relying on erratic energy sources. As illustrated above by the fictional story of Acme Widget Company, circumstances like these can cascade into catastrophic economic losses — or even life-threatening outcomes — if system failures occur. As Taguchi (Ross, 1996) noted above, the ripple effects of a system breakdown could result in damage far more devastating than any single failed transaction. Those are the types of legal and reputational liabilities any manufacturer should want to avoid.

Even if your customer feedback databases don't reflect specific examples of extreme circumstances of use, AI assistants such as ChatGPT can examine raw data from user interviews, logs, or brainstorming sessions and surface potential problems that users might encounter with your system. AI can then draft use cases, specifications, and design documents that take those situations into account, as well as ensure consistency across requirements, definitions, and naming conventions.

Recommendation #4: Pursue an iterative interface design process

Designing an interface to minimize Job 2 tasks while supporting Job 1 goals requires a thoughtful, iterative approach. Yet whenever the schedule driver is an aggressive product release date, pushing the system out the door on time often takes top priority, even if it compromises the end result. Before AI, product teams constantly grappled with this tension. Today, however, AI advancements can facilitate repeating development cycles, enabling teams to test, refine, and improve interfaces with much greater agility and clarity.

The UX Design Institute recommends integrating AI-powered Uizard or Galileo AI to rapidly generate multiple UI variants based on textual prompts. Tools like these help teams explore diverse design options quickly, facilitating a more agile cyclical process. AI can also create test cases and procedures for usability testing and beta testing (derived from the use cases and design specifications in #3 above). Then AI can summarize testing feedback, highlight relevant pain points, and propose targeted improvements for the next iteration.

In short: AI doesn't replace human expertise. It helps human experts shape system interfaces that truly support Job 1 performance. And when everyone on the team — human and AI alike — is attuned to the same performance goals, the resulting system can do more than just work. It can meet or exceed the Job 1 expectations of the people who use the final product, instead of dragging them down a rabbit hole of frustration and inefficiency.

Case study #1: Detecting Usability Problems with AI Sentiment Analysis

A research team in Italy (Desolda et al., 2022) developed a lightweight AI tool called SERENE to help design teams uncover usability problems on websites. Instead of relying on lengthy surveys or formal usability testing, the tool silently tracked how visitors interacted with the site, such as what they clicked on and how long they stayed on each section.

Then, using AI trained on emotional cues from text and behavior, SERENE estimated how satisfied or frustrated users likely felt during those interactions. It presented the results as color-coded heatmaps, allowing designers to quickly identify "hot spots" of confusion or disengagement on the page.

Even though SERENE was still in an early research phase, it showed how AI could offer real-time insights into user sentiment, without interrupting the user experience. This case illustrates how AI can help teams quickly zero in on trouble areas and make targeted improvements — all while supporting a Job 1 focus on intuitive, frustration-free interaction.

Case study #2: Reducing the Job 2 burden with Job 1 insight (and AI)

Imagine a community clinic that uses an ancient reporting system to log the services provided to the local neighborhood. The system was originally designed around cryptic database

structures, and it requires users to look up arcane service codes, navigate multi-tab forms, and validate cross-field data — all classic Job 2 tasks. The user manuals and help information reinforce those requirements, as they focus on correct data-entry rules rather than the real-world responsibilities of the clinic staff. Since training takes weeks and data-entry errors are common, the system is sorely in need of an overhaul.

Although a full system redesign isn't in the budget, even modest improvements can yield measurable benefits. To get the ball rolling, a performance specialist proposes an initial shift toward Job 1 thinking. Instead of focusing on how to merely plod through each data entry screen, the team starts exploring why users are entering the data in the first place — what they're really trying to accomplish. The team begins reworking the system with that larger picture in mind by:

- Introducing a persona-driven case study to profile a new outreach worker who is logging her first week of services. This narrative helps ground her data entry tasks in real-world decisions such as, “How do I document multiple services I've provided during a single outreach event?”
- Adding overview diagrams to show how the services she enters lead to local and national impact metrics — making the “why” behind her data entry clearer to her.
- Embedding contextual glossaries and inline prompts to reduce her need to hunt for terms or eligibility rules.

Meanwhile, AI helps accelerate the transformation by:

- Scanning thousands of support tickets and training transcripts using AI-powered text mining tools to flag recurring friction points.
- Generating an initial glossary of confusing system terms with the help of large language models (LLMs) trained on internal documentation.

- Drafting alternate versions of the input screens, each with slightly different wording and labels, using an AI-driven design assistant. The team will test each variation with users to see which version is easier to understand and use (A/B testing).

This composite case, reflecting patterns common to health-care data systems, illustrates how even a modest, AI-supported shift toward Job 1 thinking can produce verifiable improvements. In this case, onboarding was faster and data entry errors dropped by over 30% — all without requiring a full system overhaul.

Now imagine the possibilities if AI were involved from the very beginning, guiding the design of a brand new interface unencumbered by outdated architecture. With a clean slate, the team could engage a focus group of both experienced and novice users to help them reimagine the entire process, specifically by rethinking both Job 1 and Job 2:

- Job 2: A "Job 2 hunt" could surface dozens of recurring pain points — gathered from trouble tickets, support logs, and user feedback — enabling the team to eliminate or automate many setup and troubleshooting tasks.
- Job 1: Core user guidance could be embedded directly into the interface through intuitive procedures, contextual task guides, smart checklists, and interactive simulation tools that support real-world responsibilities from the start.

The Audience Response

The clinic scenario is just one proof point. Across industries, similar transformations are within reach. By steadfastly following these suggestions — and heeding the guidance of our three master conductors, enhanced by AI — we fast-forward to the final bows. As momentum builds and our work takes center stage, the suspense gives way to long-awaited gratification. Our audiences, who for years have remained silent or offered only mixed reviews, reward our efforts with thunderous applause and sold-out performances.

References

1. Basu A (2025) AI + UX: Design for intelligent interfaces.
2. Bowie J (2003) Customer-driven project management.
3. Desolda G, Esposito A, Lanzilotti R, Costabile MF (2022) Interplay between AI and HCI for UX evaluation: The SERENE case study. In Proceedings of CoPDA 2022: Cultures of Participation in the Digital Age – AI for Humans or Humans for AI? (pp. 55–59). CEUR Workshop Proceedings, Vol. 3136.
4. Gilbert TF (1996) Human competence: Engineering worthy performance (tribute edition). Amherst, MA: HRD Press.
5. Ross PJ (1996) Taguchi techniques for quality engineering: Loss function, orthogonal experiments, parameter and tolerance design (2nd ed.). New York, NY: McGraw Hill.

Glossary

A/B testing: A method of comparing two or more versions of a design element to determine which one performs better in terms of usability or user preference.

Artificial intelligence (AI): The use of machine learning and language models to analyze data, generate content, or assist in task automation, which is now emerging as a design collaborator.

Attention map: A visualization technique that helps illustrate which parts of a page users spend the most time on (such as an image or text box).

Context-sensitive help (or contextual guidance): Software features, such as prompts, glossaries, or tooltips, that help related to the specific program, command, or dialog box that is currently open.

Design of experiments (DOE): A structured methodology used to determine the relationship between factors affecting a process and the output of that process. In UX, it can be used to test which design elements impact user behavior most effectively under various conditions.

Domain knowledge: Knowledge required to carry out typical responsibilities in a particular field, such as accounting,

forestry, or nursing.

Enterprise Resource Planning (ERP): A type of software system that integrates various business processes, such as finance, human resources, manufacturing, and supply chain management, into a single platform.

Heatmap: A representation of data in the form of a map or diagram in which data values are represented as colors.

Human-centered design: A design philosophy focused on creating systems that align with user needs, limitations, and goals.

Job 1: The primary, real-world goals that the user wants to achieve with the system that directly relate to his or her actual professional responsibilities or educational activities.

Job 2: System-imposed tasks usually required to install, set up, or troubleshoot software, which often involve distracting make-work and are not aligned with the user's Job 1 goals.

Microcopy: Small snippets of instructional or interface text (e.g., labels, buttons, messages) that guide users through a task.

Performance support: Tools or features embedded in the system that help users' complete tasks more efficiently and accurately.

Personas: Fictional characters who reflect real-world customers or users, who can be profiled based on available user research, survey data, support logs, or social media.

Scenario: A narrative or story that describes the activities of one or more persons, including information about goals, expectations, actions, and reactions.

Taguchi loss function: A concept from quality engineering that quantifies the societal and economic loss incurred when a product's performance deviates from its optimal target.

Usability: The ease, speed, and pleasantness with which intended people can use a product.

Use case: A set of scenarios tied together by a common user goal. A use case specifies all possible interactions between the user and the system with respect to those scenarios.

User experience (UX): The overall experience users have when interacting with a product or system, encompassing usability, satisfaction, and task success.

User interface: Physical representations and procedures, integral to a software system, that enable users to view and interact with the system functionality.

Variability prevention and reduction: A proactive design strategy aimed at minimizing inconsistencies in product performance, especially in the presence of uncontrollable or unpredictable factors.

Workflow: A sequence of steps that move from the beginning to the end of a process.