Tesler's Law

Idea In Short

Do not assume that simplification eliminates complexity. Tesler's Law, also known as the law of conservation of complexity, argues that every system contains an irreducible amount of complexity. The real design question is not whether that complexity exists, but who has to absorb it: the user, the application, the platform, or the operational process behind the scenes. Popularized by Larry Tesler, the idea remains central to product design because it warns against shallow notions of simplicity. A cleaner interface may still rely on more complex logic, defaults, automation, data structures, support processes, or implementation work. The strategic lesson is that strong design does not magically remove difficulty. It reallocates unavoidable difficulty to the place where it creates the least harm and the most value

Simplicity is one of the most praised qualities in product design, but it is also one of the most misunderstood. Many teams talk as if complexity can be eliminated if they are clever enough. Tesler's Law says otherwise. Complexity can often be reduced for the user, but only because it is being absorbed somewhere else. The real design problem is therefore not whether complexity exists. It is where that complexity should live.

What Tesler's Law says

Tesler's Law, also called the law of conservation of complexity, states that every application has an inherent amount of irreducible complexity1. Larry Tesler's own formulation asks who will have to deal with that complexity: the user, the application developer, or the platform developer. This is what makes the law so enduring. It shifts the conversation from elimination to allocation.

That distinction matters because design discussions often confuse cleaner surfaces with simpler systems. A product may look simpler only because hidden layers have become more sophisticated. In other words, simplicity at the interface can mean greater complexity in the implementation.

This is why Tesler's Law is not anti-simplicity. It is anti-fantasy about simplicity.

Why the law matters in user design

User design is fundamentally about burden allocation. Every step that is removed from the user journey must be handled somehow. If a form autocompletes, the system must infer. If defaults are smart, someone must define them. If onboarding is reduced, the product must carry more explanatory or predictive logic. If advanced controls are hidden, the design must decide when and how to reveal them.

This is exactly the point Tesler made in his original explanation of the law. The complexity is not gone. It has moved2. The strongest product teams deliberately move it away from users when the organization can absorb it more efficiently.

That trade is often economically rational. A team may spend a week building software logic that saves millions of users repeated effort.

Why hidden complexity is often worth it

Tesler argued explicitly that it can be better for engineers to handle a unit of complexity once than for users to handle it repeatedly. The underlying principle is leverage. If a difficult step can be encoded in software, platform conventions, or backend processes, many users benefit from that decision every day. The system becomes more complicated internally, but the experience becomes easier externally.

This logic explains why great products often feel obvious only after the hard work is done. Defaults, automation, structured workflows, modeless interaction and progressive disclosure can all reduce user burden while increasing the sophistication of the underlying system3. The visible simplicity is purchased by invisible design effort.

For leaders, this is a strategic message about investment. Ease of use is rarely free. It usually requires complexity to be paid somewhere else.

Why complexity cannot simply vanish

The law is powerful because it rejects magical thinking. Systems with many capabilities,

constraints, exceptions and user needs must process that complexity somewhere. If a product promises rich functionality, adaptability and reliability, then some layer of the system must carry the burden. If the interface does not, the backend probably does. If the product logic does not, support operations may. If neither does, the user almost certainly will.

This is why the law is sometimes compared to a waterbed. Push complexity down in one place and it rises elsewhere. The design challenge is not to flatten the whole surface absolutely, but to decide where the bulge does the least damage.

Archived notes from Tesler's own writing emphasize precisely this point: every application has irreducible complexity and the meaningful design question is who must deal with it4.

Where teams place complexity badly

Teams place complexity badly when they optimize for local convenience instead of system outcomes. They may push decisions onto users because it is easier than building inference logic. They may expose too many settings because simplifying defaults feels risky. They may create short-term implementation shortcuts that leave users carrying repeated cognitive burden forever.

But teams can also fail in the opposite direction. They can oversimplify the interface so aggressively that important controls vanish, exceptional cases break, or users lose the ability to understand what the system is doing. A product that feels elegant in the demo can become brittle or opaque in real use. Tesler's Law warns against both extremes.

The objective is not to hide everything. It is to place unavoidable complexity where it is most manageable and least harmful.

What the law means for product strategy

Tesler's Law is not only a UX principle. It is a strategic principle about operating model design. When a company promises a frictionless user experience, it may be committing itself to more backend software, more platform rules, more support sophistication, more data infrastructure, or more manual exception handling. Simplicity at the customer edge often requires complexity inside the firm.

This matters because leaders sometimes demand elegant front-end experiences without funding the system changes that make them possible. They want simplicity as an output while resisting the complexity investments needed underneath. Tesler's Law exposes that contradiction.

Modern product discussions often restate the law in exactly these terms: every product has irreducible complexity and the strategic question is whether the user, the system, or the organization should bear it5.

What leaders should do with it

First, identify the irreducible complexity honestly. What must the system truly handle because of regulatory constraints, exception cases, user diversity, or functional ambition? Teams cannot allocate complexity well if they pretend it does not exist.

Second, decide who should absorb each part of it. Expert teams, automated systems and reusable platform components can often absorb complexity more efficiently than end users can. But only if the economics and reliability support that move.

Third, protect necessary transparency and control. If complexity is shifted away from the user, the user may still need confidence, override capability, or a mental model of what the system is doing. Simplification should reduce burden without creating dangerous opacity.

The deeper lesson

Tesler's Law matters because it replaces a superficial idea of simplicity with a more disciplined one. Good design is not the absence of complexity. It is the thoughtful placement of complexity. The best systems feel easy not because nothing difficult is happening, but because the difficult parts are being managed where they can be managed best.

That is the executive lesson. When a team claims to have removed complexity, ask where it went. The answer will usually reveal the real cost, quality and scalability of the design.

- 1The Law Of Conservation Of Complexity
- 2The Law Of Conservation Of Complexity
- 3Tesler's Law

- 4Tesler's Theorem And Other Adages And Coinages
- 5Tesler's Law

Summary

Tesler's Law endures because it gives leaders a more honest vocabulary for simplicity. Users often experience a product as effortless only because complexity has been shifted into software logic, platform infrastructure, internal operations, or design constraints established earlier in the process. That trade can be highly desirable, especially when expert teams can absorb complexity once instead of forcing millions of users to absorb it repeatedly. But the law also warns against oversimplification that hides necessary control, reduces transparency, or creates brittle automation. The best designers and leaders therefore ask not whether complexity can vanish, but where it should live so that the system remains usable, reliable and economically sensible