

SEMICONDUCTOR SHOWDOWN

The Strategy Behind the Chip War

A Doctrine for Decision-Critical Systems

EMAD MASOUD

CEO IN JEANS

— ABOUT THIS PREVIEW —

This is a preview edition of Semiconductor Showdown by Emad Masoud, containing the front matter, contents, doctrine framework, and preface from the forthcoming book. It is shared for the purpose of letting prospective readers evaluate the work before purchase.

The complete book — including all chapters, supplements, the Analytics Platform, the Intelligence Layer, and the full Reference materials — will be available in print and eBook editions through major retailers worldwide.

This preview is not for redistribution. © 2026 Emad Masoud. All rights reserved.

For the full edition and updates from the author, visit emadmasoud.com

Crafting the Future Interface

A Doctrine for Decision-Critical Systems

Interfaces don't present data.

They decide what becomes a decision.

PREVIEW
EMAD MASOUD · NOT FOR REDISTRIBUTION

PREVIEW
EMAD MASOUD · NOT FOR REDISTRIBUTION

— FRONT MATTER —

Acknowledgement

Prologue — The Screen Nobody Remembers

Book at a Glance

The Decision Integrity Model

The Language of This Book — Named Concepts

PREVIEW
EMAD MASOUD · NOT FOR REDISTRIBUTION

— DOCTRINE — LAYER I —

Chapter 1 — The Failures File

Story 1 — The Rainbow That Lied

Story 2 — The Argument in the Conference Room

Story 3 — The Chart That Nobody Read

Story 4 — The Ghost in the Wafer Map

Story 5 — The SPC Chart That Cried Wolf

Story 6 — The Password That Froze the Line

Story 7 — The Feature Nobody Knew Existed

Chapter 4 Supplement — Bad vs Good Visual Pattern Library

Why Engineers Resist UX — And Why They Are Right

The Cost of Misinterpretation

Designing for Wrong Decisions

The Illusion of Data Truth

Why Semiconductor UX Stays Broken

— APPLICATION — LAYER II —

Introduction — Why This Industry Breaks Designers

Chapter 1 — How Chips Are Made: A Designer's Primer

Chapter 2 — The People Who Use Semiconductor Products

Chapter 5 — Mental Models of Semiconductor Engineers

Chapter 6 — The High-Density UI Problem

Chapter 12 — The Grammar of Semiconductor Data Visualization

Chapter 13 — Wafer Maps & Die-Level Heatmaps

Chapter 14 — SPC Charts: Statistical Process Control

Chapters 15–19 — Yield, Scatter, Timing, Trend & Interaction

Chapter 20 — Dashboard Design for Real-Time Monitoring

Chapter 22 — Building the Semiconductor Design System

Chapter 24 — Bridging the Designer-Engineer Gap

Chapter 25 — Figma, Tokens & Developer Handoff

Chapter 27 — Your Career as a Semiconductor UX Designer

Chapter 28 — Case Study: Insight Builder

Chapter 29 — Case Study: Lot Disposition

Interlude — Between the Data and the Decision

THE ANALYTICS PLATFORM — PART IX

Chapter 30 — The Data Gateway: Retrieval Window Design

Chapter 31 — The Complete Visualization Stack: 14 Analysis Types

Chapter 32 — The Workflow Designer

Chapter 33 — Platform Design: What Deep Platforms Teach Us

THE INTELLIGENCE LAYER — PART X

Chapter 34 — APC: Designing the Closed-Loop Interface

Chapter 35 — Predictive Maintenance and Digital Twins

Chapter 36 — SCADA Security: Anomaly Detection Interface

PREVIEW
EMAD MASOUD · NOT FOR REDISTRIBUTION

— REFERENCE — LAYER III —

Chapter 7 — Design Tokens & Foundations

Chapter 8 — Anatomy of an EDA Application

Chapter 11 — Waveform Viewers & Debug Interfaces

Appendix B — Data Visualization Chart Selection Guide

Appendix C — Glossary & Recommended Reading

Visual Reference Library — Bad vs Good Pattern Book

PREVIEW
EMAD MASOUD · NOT FOR REDISTRIBUTION

— CLOSING —

Final Chapter — The Interface That Doesn't Exist Yet

The Permanent Things

The Letter

The Declaration

Acknowledgement

For my son

Every system in this book was designed by someone, for someone. And every failure in those systems came from decisions made under real pressure — not carelessly, but without the language to name what was truly at stake.

This book is that language. I wrote it so that one day, when you are old enough to read it, you might live in a world where the systems around you are built with more care — for the people who trust them, depend on them, and make decisions through them.

But I also wrote it for a simpler reason.

I wrote it for you.

So that you know this: the work we do is never just about systems. It is about people we may never meet. Decisions we will never see. Consequences that travel further than we can follow.

You remind me of that. Not occasionally. Every day.

You remind me that what we build outlasts us. And that is the only reason that matters for building it right. When you read this — I don't just hope you understand what I built. I hope you feel why I cared enough to build it this way.

That why has your name on it.

— Baba

To my better half

Some people walk into your life and pass through it. Others become part of who you are.

You didn't stand beside me through this journey. You stood within it.

In the moments no one will ever read about, the pressure, the weight, the silence between decisions you were there. Not asking for attention. Not asking for recognition. Just there, with a strength most people will never understand.

This book carries my name. But it carries your presence. Every page holds a piece of your patience. Every idea carries the quiet stability you gave me when everything felt unstable.

The world celebrates loud voices. What you have is rarer. You don't need noise to be seen. You don't need to prove your worth. You are the kind of presence people feel, remember, and never forget.

I want you to hold this book and feel proud not because my name is on it, but because your impact lives inside it.

You are part of this story. Part of this journey. Part of what made this real. And more than anything — you are part of my life in a way that goes far beyond words, pages, or acknowledgements.

If anyone finds meaning in this work, they should know there is a woman behind it who made it possible in ways no one will ever fully see.

Be proud of yourself. Not for this book. For whom you are.

On the Practice

This book was written alongside years of practicing designing systems, building frameworks, and returning again and again to the question of why some interfaces help people think clearly and others do not.

It was also written alongside the fourth edition of UX Seeds: Design Beyond the Surface, a project I have been working on for fourteen years. That book is about design philosophy. This book is about design responsibility. They are the same argument, applied to different stakes.

To everyone who shared their failures with me. The honest ones are in Chapter 1.

And especially to the ones who never just said the words. Who showed up. Who acted. Who made it clear — not through what they said, but through what they did — that I mattered to them.

In a world full of promises, you chose proof. In a world full of noise, you chose presence. That is not common. That is not forgotten.

To the people who don't wait for permission, for the right moment, for the right audience. Who see a problem and move toward it while everyone else is still talking about it.

To the resilient ones knocked down in ways they never speak about, who got up without making it a story.

To the honest ones — who say the difficult thing when silence would have been easier, safer, and forgiven.

To the loyal ones — who show up when there is nothing to gain, who stay when staying costs something, who give without keeping score.

To the people who choose action over announcement. Who build quietly while others wait loudly. Who don't need to be seen doing the work to know the work matters.

This book is also for you.

Not because you will read it. But because without people like you — nothing worth reading ever gets built.

— Emad

Prologue

The Screen Nobody Remembers

Somewhere right now, at a semiconductor facility, a shift engineer is looking at a screen that nobody remembers designing.

It is not a bad screen in any obvious way. The data loads correctly. The charts render on time. The controls work as expected. If you ran a standard usability test on it, it would pass. Users can complete their tasks. The error rate is low. The learning curve is manageable.

But the screen has a flaw that no usability test would catch.

When the most important pattern in the data appears — the one that predicts a yield excursion forty-eight hours before it happens — the screen makes it hard to see. Not impossible. Hard. The engineer who knows exactly what to look for will find it. The engineer who is tired, or new, or distracted by the fourteen alerts that fired since her last break, might not.

That screen is not a technical failure. It is an integrity failure. It represents the data without representing the truth. It passes every test that was ever designed for it and fails the only test that matters: does it help the

person looking at it make the correct decision?

This book is about that screen. All of them. Everywhere they exist.

Decision Integrity Design is the doctrine for building systems where that screen is no longer acceptable. It is not a UX framework. It is not a design system. It is a set of five obligations that every interface in a decision-critical environment must fulfill.

Violate one, and the probability of a wrong decision increases. Fulfill all five, and the interface becomes what it was always supposed to be: the last reliable layer between expert judgment and irreversible action.

The failures in this book are real. The costs are real. The engineers who lived through them were not incompetent — they were working with tools that were not built to the standard this book defines.

That standard now exists. This book is it.

PREVIEW
EMAD MASOUD · NOT FOR REDISTRIBUTION

— THE BOOK AT A GLANCE —

What this doctrine contains and how to navigate it

This book is about designing interfaces

where wrong decisions have real cost.

Crafting the Future Interface

A Doctrine for Decision-Critical Systems

A Note on Framing

This Is a Doctrine, Not a Style Guide

This is a doctrine for decision-critical systems. It is written for the person who builds the interface that stands between an expert's judgment and an irreversible outcome.

In a semiconductor fab, a wrong lot disposition decision can cost \$300,000 in a single pass. Because the interface made the wrong call easy. Not a usability failure — a decision failure, caused by the interface.

Industrial software is not a usability problem. It is a decision integrity problem. A yield engineer at 2 AM is not asking if the interface is pleasant. She is asking if she can trust it fast enough to act. This industry has answered only the first question.

An interface that doesn't enable the correct decision is a failed system. No matter how fast. No matter how beautiful. No matter how scalable.

If the wrong decision is easy, the system is broken.

Most industrial software fails not because it is hard to use, but because it makes the wrong decision easier than the right one.

Decision integrity: the system shows the truth, fast enough to act on.

This book introduces Decision Integrity Design — the missing standard for industrial software. Not a framework. Not a methodology. A definition of what it means for an interface to succeed in systems where wrong decisions have real cost. The interface is the last line of defense between an expert's judgment and an irreversible outcome.

Interfaces don't present data. They decide what becomes a decision.

The screen that was never designed cost \$430,000 a year in operator time. The people who built the rest of the system were brilliant engineers. No one had told them the interface was their responsibility too.

Why Semiconductor

This Book Is About Industrial Software — Semiconductor Is the Case Study

Semiconductor manufacturing presents the hardest version of nearly every problem in industrial software design. It is not the only version.

Why Semiconductor?

Consider the design challenges this industry concentrates in one place:

Data density so extreme that conventional visualization tools fail. A single wafer lot produces millions of parametric data points from thousands of devices across hundreds of test parameters.

Expert users who have spent years developing mental models that any interface must respect — but who may have very little UX vocabulary to describe what they need.

Mission-critical decisions made on the basis of software output. A wrong yield calculation, a misread chart, a misfired disposition — these have direct consequences in product quality, customer commitments, and in automotive and medical device applications, safety.

Multi-decade legacy infrastructure. Many facilities run software that was designed in the 1990s, on hardware that predates modern display standards, by engineers who are now retired.

A culture that treats UX as a luxury. In an industry that has produced the most technically sophisticated products in human history, the argument that 'the interface matters as much as the algorithm' is still, in many organizations, a hard sell.

If you can design well for semiconductor software, you can design well for any industrial domain.

The Industrial Software Designer

This doctrine is written for the industrial software designer — one of the rarest professionals in technology: someone who understands both the domain deeply enough to know what the expert needs and the interface craft well enough to build it. Most organizations don't know this role exists. The ones that do have a structural advantage. This book is written for her — and for the organizations that are ready to take her seriously.

Industrial software designers work in energy management systems, pharmaceutical manufacturing execution systems, aerospace operations tools, rail network control systems, agricultural data platforms, and, of course, semiconductor manufacturing. What they share is a set of design problems that consumer UX education barely addresses:

How do you design for expert users without patronizing them — or overwhelming everyone else?

How do you make data-dense interfaces readable without destroying information?

How do you handle the gap between what domain experts can describe and what they actually need?

How do you design for 24/7 operations, where error means downtime?

How do you build trust in the interface itself — trust that the chart is showing the truth, that the alert is real, that the disposition result is correct?

These are not abstract questions. Every chapter in this book is an attempt to answer one of them. The semiconductor context gives them specificity. The answers hold everywhere industrial software is built.

If You Are Not a Semiconductor Designer

You do not need to understand how a transistor works. You need to understand why an engineer who does would be frustrated by the interfaces built for her. That frustration is the subject of this book.

Read the technical chapters for the principles, not the parameters. Where the book says 'parametric test result,' read 'measurement.' Where it says 'die yield,' read 'unit pass rate.' Where it says 'wafer map,' read 'spatial quality distribution.' The domain vocabulary is specific. The design vocabulary is transferable.

Complexity is not the enemy of good design. Unexplained complexity is.

— a lesson semiconductor software teaches better than any other domain

— PREFACE —

The Screen That Was Never Designed

The examples in this book are drawn from industry patterns observed across multiple organizations over many years of practice. Any resemblance to specific systems, products, or companies is incidental. All failures described are representative of documented industry patterns, not accounts of specific organizations.

The first time I walked into a semiconductor test facility, I was not there as a designer.

I was there because a yield engineer, I knew had been complaining for three months about a software tool — specifically about a screen that appeared every morning, first thing, before the day's testing could begin. She called it 'the screen nobody designed.'

The screen asked the operator to enter eight numbers. Not complicated numbers — device counts, bin totals, the same figures printed on the traveler card hanging from the lot's physical container. The operator would pick up the card, look at the first number, type it in, look at the next, type it in, look at the next.

This took about ninety seconds. Every morning. For every lot. At every test cell.

The yield engineer had calculated that across the facility, the cost of this screen in aggregate human time was roughly forty minutes per day. That was before accounting for the transcription errors — wrong numbers entered, lots dispositioned against the wrong data — that happened perhaps twice a week and required twenty minutes each to investigate and correct.

Someone had designed the rest of the system with obvious care. The yield analysis charts were sophisticated. The statistical algorithms were state-of-the-art. The database schema was meticulously normalized. But this screen — the one that every operator saw every single day, the first interaction of every shift — appeared to have been added at the end of a sprint by someone who had never met an operator.

I asked the engineer why nobody had fixed it. She looked at me the way people look at someone who has just asked why the weather is the way it is.

'There's no UX person on the team,' she said. 'There never has been.'

This book exists because of that screen.

— and every screen like it, in facilities around the world.

Every chapter in this book is written for the person who will one day stand in front of that screen — or the person who could have prevented it from being built that way.

That screen is not an anomaly. It is a pattern — and understanding why it exists is what this book is about.

Semiconductor manufacturing is one of the most technically sophisticated industries in the world. The physics is quantum. The processes are measured in atoms. And the software engineers use to control this equipment is — in many cases — designed by people who have never been trained to design software for human beings. That is the gap this book exists to close.

That gap costs the industry billions annually: in lost engineering productivity, in errors software could have prevented, in decisions made on misread charts, in analyses that take three hours because the interface didn't surface the right data first. Nobody designed it that way on purpose. Nobody designed it at all.

Who This Book Is For

This book is for senior designers, product leaders, and platform architects building decision-critical systems in complex industrial environments. If you are responsible for software where an expert's wrong call has a real operational cost, this book is for you.

She is not browsing. She is deciding. And when she decides, the system moves. The machine does not ask if she meant it. It executes.

Semiconductor manufacturing is the primary case study because it presents the hardest version of nearly every industrial design problem: extreme data density, expert users, mission-critical decisions, legacy infrastructure. Engineers will recognize their world in it. Product leaders will understand its cost. The specifics are semiconductor. The problems are universal.

You do not need to be a semiconductor engineer to read this book. You need to be willing to learn enough about the domain to design for it — which is a different thing, and a more interesting thing.

How to Read This Book

Read it as a doctrine: front to back, letting the argument build from failure to principle to standard. Read it as a reference: open to the chapter that addresses your current problem. Read it as a field guide: use the named concepts and model tables when you are in the middle of a design decision and need the framework fast. All three are valid. The doctrine holds in all three directions.

One recommendation: start with the failures. They are more instructive than the solutions, and they will tell you whether this book is addressing the problems you actually live with.

— Emad Masoud

How This Book Is Organized

This book operates on three levels. The first is doctrine: the argument for why industrial software fails and what Decision Integrity Design means as a standard. The second is application: how the doctrine maps to specific design problems in semiconductor software. The third is reference: the implementation specifications, token systems, and rendering details that support the application layer. Read the doctrine first. The application without the doctrine is a collection of patterns. The doctrine without the application is a philosophy without evidence. Both together are the argument.

A reader who wants the argument reads the doctrine layer. A practitioner who wants to apply it reads the application layer. An engineer who is building it reads the reference layer. The book works at any entry point.

Every argument in this book is grounded in what follows. Every failure traces back to a violation of one of these five obligations. This is the doctrine's foundation.

The Decision Integrity Model

The Decision Integrity Model is not a rubric. It is a diagnostic — a way to locate exactly which obligation a failing system violates. Name the violation, and the remedy becomes clear.

The Tensions Are Not Bugs

The five elements of the Decision Integrity Model create genuine tension with each other. This is not a design flaw. It is the defining challenge of decision-critical interface work.

Chapter Map — where each argument lives

Use this to find the chapter that answers your specific problem.

Every chapter maps to at least one obligation. If you know which obligation your current problem violates, you know which chapters to read first.

T = Truth S = Speed TR = Trust R = Recovery D = Discoverability

This is where systems fail. Not at the technology layer. At the obligation layer.

The Five Obligations — Detail Reference

This is where millions get lost quietly.

Five obligations. Here is what each one means when it fails.

Here is the one that breaks first.

The Obligation You Most Often Violate

Trust is the most commonly violated obligation, and it is violated not by malicious design but by optimism.

Designers build systems they believe in. The engineer who ignores the fifty-first alert because the first fifty were false positives is not irrational. She stopped trusting the system because the system stopped deserving trust.

This is what nobody wants to admit about the systems they build.

Designing for Trust means asking, before you build any feedback mechanism: what does this system need to do to deserve belief? Not what does it need to show — what does it need to do.

That question was asked at 2 AM by an engineer looking at an alert she wasn't sure she trusted. She acted on it anyway. She was right. Not because the system earned it. Because she was good at her job despite it.

The model is an abstraction. The failures that follow are not. Someone made each of these calls. Some of them were wrong. This book exists to make the next person right.

The Language of This Book

Named Concepts and Where to Find Them

This is what nobody writes down.

Industrial interface design has a vocabulary problem. The same failures recur under different names, making them harder to recognize, harder to discuss, and harder to prevent. This book names them.

When you encounter one of these names in the text, the definition here is the one the book is using. When you use one of these names in your own work, you now have a shared language to argue from.

The following concepts are used throughout this book. They are not jargon — they are handles for ideas that are specific enough to be useful and common enough to be worth naming. When a reviewer quotes this book, they will quote these names. When a practitioner applies this book, they will reach for these handles.

These concepts appear in bold on first use throughout the book. They are also defined in context — the glossary here is a reference, not a prerequisite. If you encounter a term you don't recognize, this is the page

to turn to.

PREVIEW
EMAD MASOUD · NOT FOR REDISTRIBUTION

— THIS IS WHERE THE PREVIEW ENDS —

The complete book begins on the next page with Chapter 1.

For the full edition — including the complete doctrine, all named anchor cases, the Analytics Platform architecture, and the Intelligence Layer reference — visit:

emadmasoud.com

If something in these pages landed for you, write back. A single line means more than a review ever could.

— Emad Masoud, Toronto, 2026