76-68-C I have Never had To Deal With Instability..

Why Stability Engineering Feels So Strange to the Computing Profession

A Personal Reflection

As a software engineer with four decades in the industry, I recently had an unsettling realization. For most of my career, I have never had to think about instability. Not in the way a chemical engineer, or a nuclear engineer, or a physician, or an air traffic controller thinks about instability.

For more than seventy years, software engineering enjoyed a remarkable luxury. The substrate beneath our systems was fundamentally deterministic.

- A program executed.
- An operating system enforced boundaries.
- A processor obeyed instructions.
- Memory remained inside its allocation.
- Processes remained inside their execution spaces.
- The system largely did what it was told.
- Certainly bugs existed.
- Failures existed.
- Security vulnerabilities existed.
- But instability itself was not the central organizing principle of the discipline.

Today, Agentic AI may be changing that.
For perhaps the first time in the history of software engineering, we are being forced to confront a phenomenon that many other disciplines have wrestled with since their inception:

> *Dynamic systems naturally generate instability.*

And that realization feels profoundly unfamiliar.

Other Disciplines Grew Up With Instability

Long before software engineering existed, other disciplines learned painful lessons about dynamic systems. Many of these lessons were written in:

- explosions,
- blackouts,
- shipwrecks,
- reactor accidents,
- epidemics,
- market crashes,
- loss of life.

Their professions evolved because instability demanded respect.
The software industry, by comparison, largely grew up in a sheltered environment.

The table below illustrates the contrast.

Discipline	Instability Present From Day One?	Typical Instability	Consequence if Ignored
Naval Architecture	Yes	Roll resonance, rogue waves, capsizing	Ship loss, deaths
Aviation	Yes	Stall, spin, oscillation, weather instability	Aircraft crashes
Chemical Engineering	Yes	Runaway reactions, thermal instability	Explosions, toxic releases
Mechanical Engineering	Yes	Vibrations, buckling, roll instability, torsional oscillations	Mechanical breakdown
Petrochemical Engineering	Yes	Pressure instability, ignition cascades	Refinery disasters
Nuclear Engineering	Yes	Reactivity excursions, neutron runaway	Meltdown, radiation release
Electrical Power Systems	Yes	Grid oscillations, cascading failures	Regional blackouts
Medicine	Yes	Homeostasis failure, infection, shock	Organ failure, death
Ecology	Yes	Population collapse, predator-prey instability	Ecosystem collapse
Economics / Finance	Yes	Bubbles, panics, liquidity crises	Financial collapse
Systems Dynamics	Yes	Feedback amplification, policy resistance	Systemic failure
Software Engineering (1945–2025)	Mostly No	Bugs, defects, vulnerabilities. At the most the program abended or system hung up.	Localized failure
Agentic AI / HCAS	Yes	Drift, coupling, synchronization cascades	Ecosystem instability

What Other Engineers Learn in Year One

A chemical engineering student quickly learns: If the reaction gets away from you, the plant explodes.
A nuclear engineering student learns: If reactivity exceeds control authority, catastrophe follows.
A medical student learns: Homeostasis is fragile and must be continuously maintained.
An aviation student learns: Airplanes are constantly trying to leave controlled flight.

A software engineering student traditionally learned:

- Here is how to write a program.

The difference is profound.

- One group learns how to prevent instability.
- The other learns how to create functionality.

Why Software Was Different

The answer lies in the substrate.

Traditional computing operated inside carefully controlled environments.
The operating system acted as an invisible governor:

- memory protection,
- process isolation,
- execution boundaries,
- deterministic scheduling.

These controls prevented many instability phenomena from emerging. The software engineer rarely had to think about:

- resonance,
- drift,
- runaway feedback,
- synchronization storms,
- ecosystem collapse.

The platform handled those concerns.

The End of the Exemption

Agentic AI may have ended that exemption.
For the first time, software systems are beginning to exhibit characteristics previously associated with living systems:

- adaptation,
- autonomy,
- coupling,
- emergence,
- self-modification,
- machine-speed feedback loops.

Suddenly concepts familiar to chemical engineers and physicians become relevant to software professionals. Concepts such as:

- stability margins,
- operating envelopes,
- drift,
- damping,
- runaway behavior,
- collapse prevention.

The profession is entering unfamiliar territory.

The Strange New Vocabulary

Many software professionals encounter words such as:

- instability,
- homeostasis,
- stabilization,
- resilience,
- damping,
- envelope management,

and instinctively feel they belong somewhere else. Perhaps in:

- aerospace,
- nuclear engineering,
- medicine,

- control theory.

Yet these concepts may soon become commonplace in computing.
Not because software changed.
But because the systems running on top of software changed.

The Great Convergence

What is happening may be viewed as a convergence of disciplines.
For decades:

Software Engineering
≠
Stability Engineering

The two worlds remained largely separate. Agentic AI may be forcing them together.
Future software professionals may need to learn lessons that other engineers have studied for generations:

- Control Theory
- Systems Dynamics
- Process Safety
- Human Factors
- Homeostasis
- Stability Analysis

The youngest engineering discipline may need to learn from some of the oldest.

The Humbling Realization

There is something humbling about this realization.
The software industry often views itself as the newest and most advanced engineering field.
Yet when it comes to instability, many other disciplines are decades ahead.

- Chemical engineers have spent generations managing runaway reactions.
- Nuclear engineers have spent generations managing criticality.
- Physicians have spent generations managing homeostasis.
- Air traffic controllers have spent generations managing dynamic traffic flows.
- The software profession may simply be arriving late to a conversation that others have been having for a very long time.

Final Insight

For seventy years, software engineering primarily asked:

How do we make the system work?

The Age of Agentic AI introduces a second question:

How do we keep the system stable while it works?

That question has existed in other disciplines for generations. It is merely new to us.
And perhaps that is why Stability Engineering feels simultaneously revolutionary and familiar.
Revolutionary to software. Familiar to almost every other engineering discipline on Earth.

The Cognitive Age may therefore mark the moment when software engineering graduates

- from the study of functionality to the study of stability.

And that may prove to be one of the most important transitions in the history of computing.


INSTABILITY IS
NEW TO THE SOFTWARE PROFESSION..
70+ YEARS OF SOFTWARE
A WORLD OF DETERMINISM
DETERMINISM
DYNAMISM
AGENTIC AI ERA
A WORLD OF DYNAMIC INSTABILITY
INPUT
CODE
EXECUTION
OUTPUT
DRIFT
EMERGENCE
RUNAWAY FEEDBACK
SYNCHRONIZATION STORMS
UNPREDICTABLE INTERACTIONS
REGIME CHANGE
COLLAPSE
WE BUILT SYSTEMS THAT DID WHAT THEY WERE TOLD.
NOW WE FACE SYSTEMS THAT CAN CHANGE AS THEY OPERATE.
NEW SUBSTRATE. NEW SPEED. NEW RISKS. NEW DISCIPLINE.
WELCOME TO STABILITY ENGINEERING.

Book Series Coming Soon..