ElastoForge — Master Brief

For: Kailash & Saurabh **From:** Kishore (Surya AI) **Date:** 30 May 2026

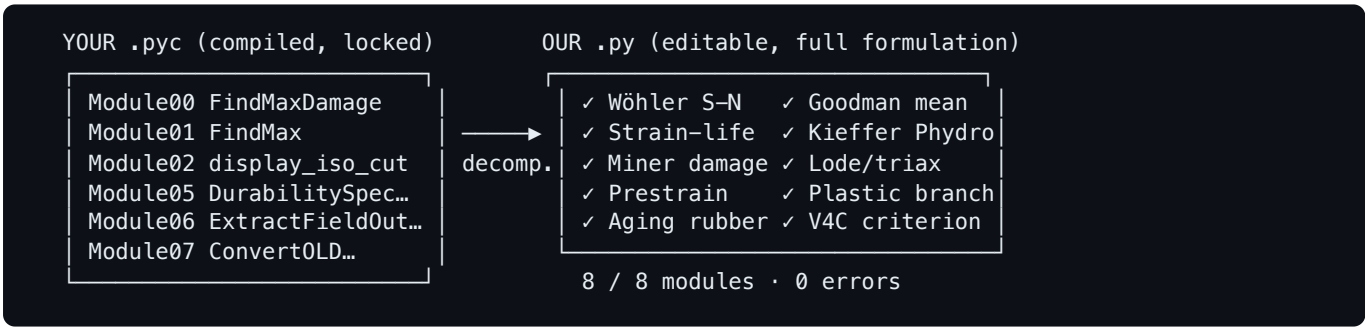
> **TL;DR Decompile done** — all 8 `.pyc` plugins recovered to editable `.py` (0 errors), full durability formulation in hand. **Engine state** — Rust core runs the rubber physics (hyperelastic + Mullins + rainflow + CED damage), **40 tests passing**. **What's next** — wire your Wöhler/Goodman/strain-life formulas + your material data into the kernel and calibrate to your numbers. **The one ask** — send us **one ground-truth (condition → expected life) pair**; that single pair unblocks the ~5% accuracy target.

How to read this: Part I is the 5-minute version (visual — ASCII boards + diagrams). **Part II** is the full detail, folded into expandable sections below it. Skim the top; open what you want to go deep on.

PART I — AT A GLANCE

1. What we just did

You asked whether the compiled `.pyc` plugins could be turned back into editable `.py`. **Yes — done.** All 8 modules recovered cleanly (Python 2.7 → `uncompyle6`, **0 errors**), every formula intact. The decompile even recovered the original source paths + 2020–2021 authoring dates, so we can trace exactly which methodology version we're matching.



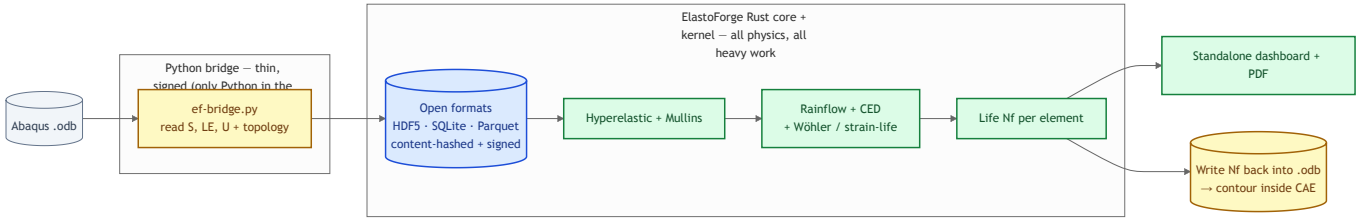
Two generations of your damage model were recovered — useful to know which is canonical:

	v02r (production)	v02i (newer)
Wöhler	Single-slope: `N = Crit^b · A`, `A = 10^Intercept`	Multi-slope (knee S1/S2/NC1)
Mean-stress	—	Goodman: $S_{eq} = S_a \cdot Rm/(Rm-S_m)$
Strain-life	LEmax / NEmax criterion	Coffin-Manson-Basquin: $NE = NE_{eq} + (\sigma f - S_m)/E \cdot N^b + \varepsilon f \cdot N^c$
Damage	Miner accumulation	Miner accumulation
Extra	Thermoplastic polynomial branch	Prestrain superposition
Shared & stable (both)	**Kieffer hydrostatic** $D \cdot 1/(1+(S_{hydro}-Trigger)^{1.5})$ **· Lode**	

	v02r (production)	v02i (newer)
	angle `cos3θ = 27/2·J3/σ³`	

2. What ElastoForge is

A **Rust-core rubber-durability analyzer**. One thin, Ed25519-signed Python script reads `.odb` files inside `abaqus python` (the only place that API exists) — **that script is the ONLY Python in the system**. All physics, all UI, all storage live in **Rust**, shipping as **one binary in two deploy modes**: a standalone app, and an in-CAE plugin that writes the computed life field (`Nf`) back into the ODB. Damage is computed **two ways side-by-side** — **Wöhler + Miner** (the established industrial frame) and **CED** (Mars/Verron rubber-physics supplement). The customer picks one to report against, or runs both for cross-validation.

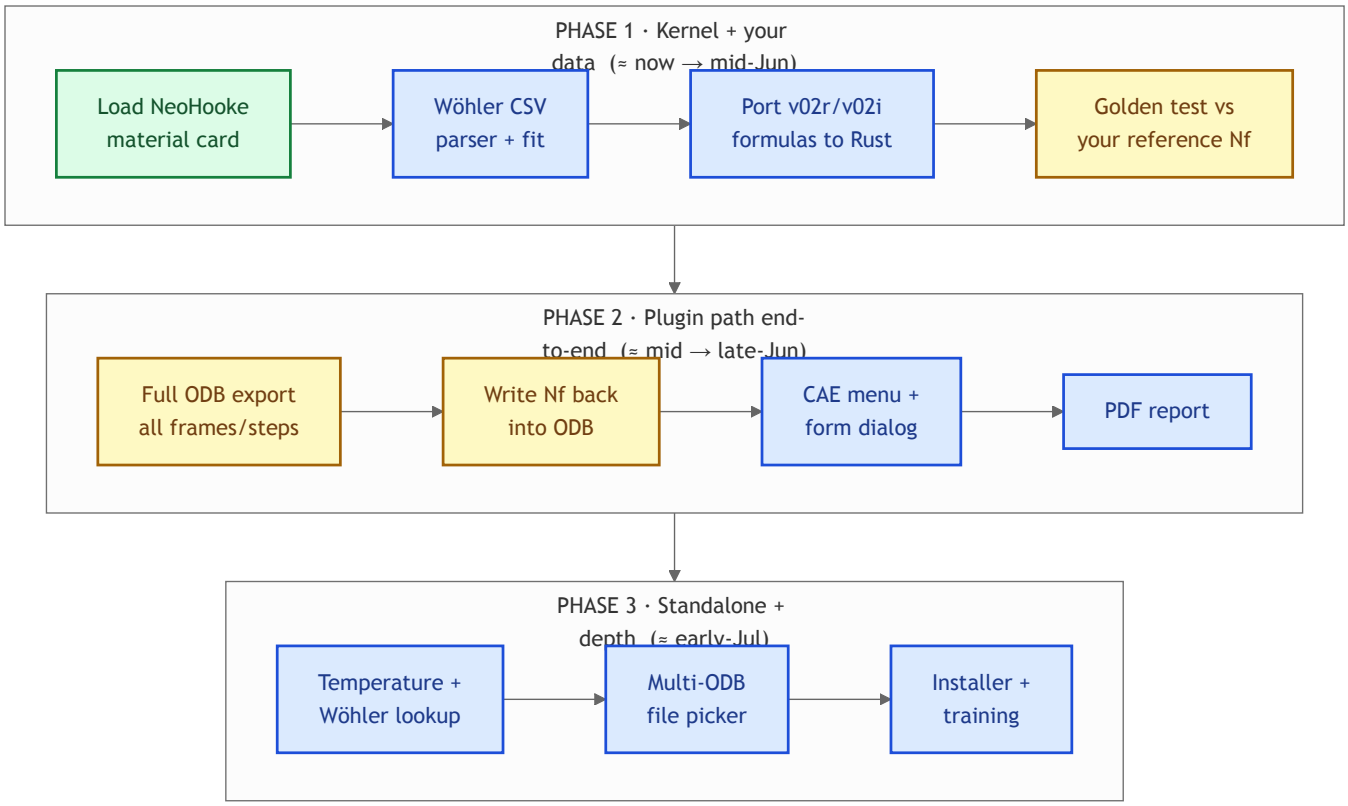


3. Status today

```
ELASTOFORGE RUST ENGINE                                  40 tests passing
═══════════════════════════════════════════════════════════════════════

RUBBER-PHYSICS CORE                    ODB DATA BRIDGE
──────────────────                     ──────────────
[✓] Hyperelastic (Yeoh/NeoHooke)       [✓] Field contract (S,LE,U+topo)
[✓] Mullins (Ogden-Roxburgh)           [✓] Signed bridge (Ed25519)
[✓] Rainflow (ASTM E1049-85)           [✓] Bridge entry script (.odb)
[✓] Energy-density damage (CED)        [~] Full per-frame export  (wip)
[✓] Prestrain / preload seed           [~] Output-ODB Nf write-back (wip)

PORTED FROM YOUR .pyc — NEXT TO WIRE IN
──────────────────────────────────────────────────────────────────────
[ ] Wöhler S-N (single+multi-slope)    [ ] Goodman mean-stress
[ ] Coffin-Manson-Basquin strain-life  [ ] Wöhler CSV material loader
[ ] Temperature / aging derating

Legend: [✓] done+tested  [~] in progress  [ ] formula in hand, not wired
```

Plain-English status: the hard, rubber-specific machinery is **built and tested**. The classical fatigue formulas (Wöhler / Goodman / strain-life) are now **in our hands from your plugins** — the remaining work is wiring them in and **calibrating against your real material data** so the numbers match Vibracoustic's. That calibration is what needs your inputs (Section 5).

4. Roadmap — three short phases



The single thing that gates **Phase 1 → "trustworthy numbers"** is **one ground-truth Nf pair from you**: one operating condition with its expected life. Without it we can produce results but can't prove the **~5% accuracy** target.

5. Open questions — quick to answer

A. Confirm in one line each:

1. **Which model is canonical — `v02r` (single-slope) or `v02i` (multi-slope + Goodman + strain-life)?** We port that one as the reference.
2. **The Wöhler curves (CTRF-003) — already corrected** for mean-stress / temperature / loading-rate, or raw uniaxial S-N? Decides whether we re-apply Goodman/Kieffer (applying twice would double-count).
3. **Material card `NR-C023A-500`** — OK to keep as our permanent regression fixture, or pilot-only/private?
4. **Z-direction strain** — confirm you want it extracted **per load-step (all 3 steps)** through rainflow, not last-frame only.
5. **Temperature (23–70 °C envelope)** — lookup per temperature band, single worst-case, or aging slope from the CSV?

B. Please send when you can (the real accelerators):

6. **One ground-truth (operating condition → expected life) pair. This is the #1 unblocker** — it turns the engine from "produces numbers" into "validated to ~5%."
7. **The S-N / material-fitting calibration routine** — the offline code that fits the Wöhler curve from test data. It wasn't in the plugin zips, and it's the one piece the decompile didn't recover.
8. **The 5 GB Ford V801 full ODB** + (if handy) the **VC plugin walkthrough video** — both let us test export at real scale and match your in-CAE workflow.

PART II — FULL DETAIL

▼ 6. The three deployment paths (A plugin · B standalone · C today)

The word "plugin" is overloaded. We do **NOT** mean a monolithic Abaqus plugin (like Bony / the Vibracoustic internal tool) where physics + UI + IO live in one `.py` blob locked to an Abaqus version. We mean a **thin signed bridge (~100 LOC Python that only reads `.odb` → signs → writes open formats) + a Rust kernel** that holds all math, validation, and rendering — testable, fast, portable, Abaqus-version-independent, and sellable as plugin **and** standalone.

Path A — Plugin (in-CAE workflow). The engineer never leaves Abaqus. One menu button → form dialog → compute → inject `Nf` back into the ODB → contour it inside CAE. Same UX pattern as the Bony plugin Kailash already knows — but with our kernel doing **correct elastomer physics** underneath (multi-frame rainflow, hyperelastic + Mullins, calibrated against your Wöhler curves), not single-frame metal-style formulas. Primary audience: the FEA analyst.

Path B — Standalone app (manager / supplier view). Point the app at a folder of `.odb` files; the bridge runs invisibly as a subprocess; results land in our own 3D viewer + PDF report. No CAE write-back needed. Sells as a **separate SKU** (review tool / supplier portal / manager dashboard) and shares **~95% of code** with Path A — same kernel, same bridge, only the UX wrapper differs. So both ship as separate revenue channels from one codebase.

Path C — where we are today (honest snapshot). Kernel + dashboard are the strongest pieces (done + tested). The bridge wiring (full per-frame export, output-ODB write-back), the real `NR-C023A-500` material-card path, and the Wöhler/SN fatigue lookup + temperature field are the load-bearing gaps — the work mapped out in the Part I roadmap.

	Path A · Plugin	Path B · Standalone
Entry point	Menu button inside Abaqus CAE	Native macOS / Win / Linux app icon
Engineer location	Stays in CAE the whole time	Switches to our app
Output surface	ODB contour in CAE viewer + side-car PDF	Our 3D viewer + PDF
Primary audience	FEA analyst (Kailash, Saurabh)	Manager / VP-Eng / supplier reviewer
Sales motion	Bundled with Abaqus workflow	Separate SKU — review tool / portal
Shared code	**~95%** — kernel + bridge identical, only UX wrapper differs	

▼ 7. Your workflow flowchart, mapped to our pipeline

The two-section flowchart Kailash shared on 2026-05-26 — *"Strain Extraction from ODB"* + *"Durability Evaluation by Wöhler Curve"* — is our canonical algorithm spec. Every step maps onto something already in our stack or in active build:

Flowchart step (Kailash)	Where it lives in ElastoForge	Status
1.1 Input ODB → select step/frame/instance	Bridge (input handling)	done
1.2 Select region/location (nodes/elements/set, path/surface)	Bridge — instance subsetting today; set/path/surface in thick-extractor upgrade	partial
1.3 Select strain component (εxx/ εyy/εzz/γxy/γyz/εeq) + IP/Element/Nodal	Bridge → Rust ingest (`field_contract.rs`) — IP done; element + nodal averaging next	partial
1.4 Extract strain history (vs time/frame)	Multi-frame walk → HDF5 export; CSV available downstream	in progress
1.5 Post-process strain (filter, detrend, remove mean)	`avartan-elasto-kernel/cycle.rs` (rainflow prep)	done
2.1 Prepare strain amplitudes via rainflow → {εa, Ni}	`cycle.rs` ASTM-E1049 rainflow	done
2.2 Define material S-N (Wöhler) curve — Basquin / Coffin-Manson + mean-stress	**NEW Rust module — formulas recovered via decompile** (single+multi-slope Wöhler + Goodman + Coffin-Manson-Basquin)	**unblocked · porting 1:1**
2.3 Damage calculation (Di = ni/Nf, D = ΣDi, Miner's rule)	**NEW Rust module** (alongside existing CED)	**unblocked · porting 1:1**
2.4 Check durability (D ≤ 1 pass / D > 1 fail)	`ef_dashboard` result panel	scaffolded
2.5 Output results (D, safety factor 1/D, life, critical locations, plots)	`ef_dashboard` + PDF report	partial

The two formerly-blocked rows (**2.2 + 2.3**) are now unblocked **from the source side** — the formulas were recovered in the decompile and are porting 1:1 into Rust. Everything else is either done or actively built. The "thick generic extractor" upgrade to the bridge is the load-bearing piece for the next two weeks.

▼ 8. Ships today vs building next (reconciled to 40 tests)

Done + tested in the all-Rust engine (40 tests passing as of 2026-05-30):

Capability	Where
Yeoh / Neo-Hooke hyperelastic strain energy	`avartan-elasto-kernel/material.rs`
Ogden-Roxburgh Mullins softening	`avartan-elasto-kernel/mullins.rs`
Rainflow cycle counting (ASTM E1049-85)	`avartan-elasto-kernel/cycle.rs`
Energy-density (CED) damage accumulation	`avartan-elasto-kernel/cycle.rs`
Prestrain / preload seed	`avartan-elasto-kernel`

Capability	Where
Ed25519-signed Python bridge → Rust verifier	`scripts/elastoforge-odb-bridge.py` + `avartan-core/src/elasto/bridge_loader.rs`
Field contract (S / LE / U + topology)	`avartan-core/src/elasto/field_contract.rs`
Bridge entry script (reads `.odb`)	`scripts/elastoforge-odb-bridge.py`
Standalone GUI (egui/wgpu)	`ef_dashboard` — 6-pane shell

In progress: - **Full per-frame ODB export** — walk every frame/step and export the called fields. - **Output-ODB `Nf` write-back** — inject the computed life field back into a thin output ODB for in-CAE contour (the Path A killer-UX feature).

Next — formulas now in hand from the decompile, not yet wired: - **Wöhler S-N — single + multi-slope** (v02r single-slope `N = Crit^b·A`; v02i multi-slope/knee). - **Goodman mean-stress correction** `Seq = Sa·Rm/(Rm−Sm)`. - **Coffin-Manson-Basquin strain-life** `NE = NEeq + (σf−Sm)/E·N^b + εf·N^c`. - **Wöhler CSV material-data loader** — wire the parsed CTRF-003 curves into the damage path. - **Temperature / aging derating** — once temperature handling is confirmed (open Q5).

▼ 9. Material model & data facts on hand

NeoHooke material card `NR-C023A-500`:

Parameter	Value
C10	0.313 MPa
D1	0.000732611 1/MPa
Hardness	50 Shore A
Cure	175 °C
Derived shear modulus G	≈ 0.626 MPa
Derived bulk modulus K	≈ 2730 MPa
Derived Young's modulus E	≈ 1.88 MPa

Neo-Hooke is **Yeoh with C20 = C30 = 0** — so it drops straight into the existing kernel hyperelastic path, no new code.

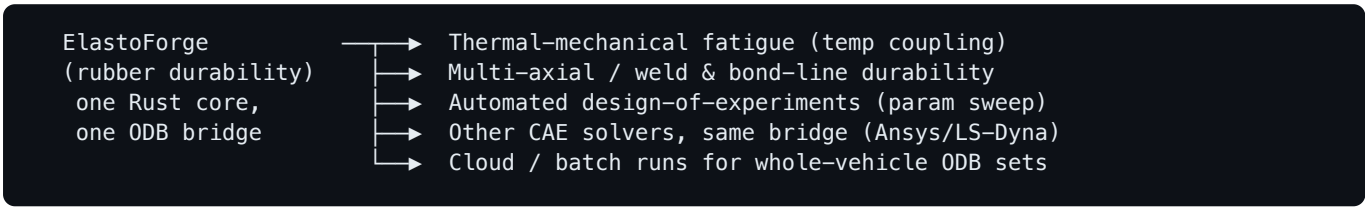
Wöhler CSV `CTRF-003` (NR-C023A-500):

Parameter	Value
Slope b	−3.2668
Intercept A	11.644
Valid temperature range	23–70 °C
Aging slope	−4.7449

The whole **C023A family shares the slope** (per-compound intercepts differ).

Damage conventions — two frames, side-by-side: - **CED (Mars/Verron)** + Mullins + rainflow — the rubber-physics supplement that catches energy-dissipation Wöhler can't model. - **Basquin HCF + Coffin-Manson LCF + Miner** — the established industrial strain-life + cumulative-damage frame (same family that fe-safe ships), now with **measured rubber Wöhler curves** (exactly what you sent — CTRF-003, 128 rows × 31 compounds), making it rubber-correct for the industrial use-case.

▼ 10. Future capability branches (after the pilot lands)



Every branch reuses the **thin-Python-bridge + all-Rust-engine** pattern — no rewrite, just new modules on the same core. Each new CAE vendor gets its own thin bridge; the Rust core is unchanged, so customers running multiple solvers (most Tier-1 OEMs) get one tool that ingests from all of them. A vendor-supplied material library (ship the validated Vibracoustic dataset as a starter, version cards in the SQLite catalog) and a hosted, hash-chained reproducibility tier sit on the same foundation.

▼ 11. Architecture lock (2026-05-28 record)

The thick-generic-extractor decision. The Python bridge makes **no analytical decisions** — it is a thick *generic* extractor that dumps **everything** from the ODB to open formats (**HDF5** numeric arrays + **SQLite** catalog/metadata + **Parquet** columnar slices), each **content-hashed and Ed25519-signed**. The bridge decides nothing; the Rust kernel decides everything. This also moves the reproducibility primitive one level down: reproducibility is **content-hashed, not `.pyc`-tied** — the same content-hashed extraction can be fed to your existing VC plugin **and** our Rust kernel, with a log-diff between the two outputs (stronger than PYC + logs alone, and it works across organisations).

Output-ODB write-back of `Nf`. Per Kailash's 2026-05-26 21:47 note — *"the funfact is output results is also will be odb. but less size. It consist only those strains which we called"* — the Abaqus durability output is itself an ODB, holding only the fields we called. So our pipeline writes the computed life field (`Nf`) back into a **thin output ODB (~50–200 MB)** for in-CAE contour, in addition to handling the (4.8 GB-class) input ODB. That round-trip is the Path A in-CAE killer-UX feature.

Two deployment modes from one binary. The same Rust artefact ships as a standalone app and as a plugin installed alongside Abaqus — `KERNEL → DASH → standalone` and `KERNEL → Nf contour write-back → plugin`. Both algorithms (CED + Wöhler-Miner) ship side-by-side; the strain-life model carries the originating method's name in the dashboard UI.

CONFIDENTIAL — prepared for Kailash & Saurabh / Vibracoustic India pilot. Not for distribution.