📄 Phase 7.2 — Rule-Based Detection (merged with relevant content from 7.1)

7.2.1 Recap of the System Context

Our system is a **standalone telematics-style module** running on the **Arduino UNO Q**

✔ listens passively to the vehicle CAN bus
✔ decodes ECU voltage telemetry from a defined DBC
✔ applies health/diagnostic logic in real-time
✔ emits alerts if a harness degradation pattern appears

Mechanical fault being simulated:

> **Harness resistance increases → ECU sees artificially lower supply voltage → detected via cross-comparison against DCDC reference**

Diagram (from 7.1, restated for continuity):

```
[12V Battery] --> [DCDC 14.1V] --> Harness --> ECU A
                                           \--> ECU B
```

Failure progression:

Baseline → Minor Drop → Detectable → Severe → Failure

The failure manifests as **gradual voltage drops localized per ECU**.
This behavior lends itself to **rule-based detection**, because:

✔ it's deterministic
✔ it's monotonic
✔ reference voltage is known (DCDC)
✔ thresholds can be physically derived

7.2.2 Why rule-based works well here

Predictive maintenance has multiple families of techniques:

Method	Example Use	Notes
Rule-based	harness, fuses, overcurrent	best when cause-effect is known
Statistical	bearings, vibration RMS	best when historical baseline exists
ML / Model-based	RUL, Load patterns	best for noisy nonlinear systems
Hybrid	control + ML	aerospace, power systems

For our system:

✔ The physics are linear
✔ Failure modes are observable
✔ ECU voltages are directly measurable
✔ Fault progression is monotonic
✔ No ML needed for early phases

Therefore **rule-based screening** is not only sufficient, it is optimal.

7.2.3 Signal Model

Let:

- **V_DCDC** = DCDC output voltage (reference)
- **V_A** = ECU_A supply estimate
- **V_B** = ECU_B supply estimate

Define deltas:

```
ΔA = V_DCDC - V_A
ΔB = V_DCDC - V_B
```

Interpretation:

- Δ ≈ 0 → normal
- Δ > threshold → suspect harness resistance
- Δ >> threshold → confirmed failure

Typical harness degradation threshold from automotive standards: **0.5–1.5V drop range** is where diagnosis becomes meaningful.

We tested:

```
warning_threshold = 0.5V
alarm_threshold = 1.0V
```

Which matched simulated behavior well.

7.2.4 Detection Logic States

We implement **progressive diagnostic states**, not just binary faults:

State	Meaning	Action
Normal	system healthy	log
Warning	mild degradation	continue + start timer
Alarm	confirmed harness fault	log + emit alert
Severe	safety relevant	trigger event
Recovery	fault clears	hysteresis reset

Each state needs **hysteresis** to avoid flapping.

Example hysteresis:

```
enter_alarm:  Δ > 1.0V for > 3 samples
exit_alarm:   Δ < 0.8V for > 10 samples
```

Note: this matches ISO 14229 thinking (UDS DTC entry conditions).

7.2.5 Multi-Node Reasoning (A vs B vs DCDC)

Our harness case is **multi-sensor relational**, not absolute.

We detect patterns:

Pattern A – Harness_A fault

```
ΔA > alarm_threshold AND ΔB < warning_threshold
```

Pattern B – Harness_B fault

```
ΔB > alarm_threshold AND ΔA < warning_threshold
```

Pattern C – Shared fault (downstream)

```
ΔA > alarm_threshold AND ΔB > alarm_threshold
```

Pattern D – DCDC fault

```
ΔA ≈ ΔB ≈ 0 AND V_DCDC < system_min
```

This distinction is key because:

- It allows **fault localization**
- It improves diagnostic confidence
- It avoids false positives against supply sag events

7.2.6 Real-Time Execution Architecture

Executed directly inside the Linux app:

```
CAN → DBC Decoder → Voltage Extractor → Rule Engine → Alert/Event
```

No cloud, no offline replay.

The interface produced human-readable logging:

```
[ALERT][HARNESS_A] ECU_A=13.1 V ECU_B=14.1 V DCDC=14.1 V
```

And did so at ≈ 100ms sample latency.

7.2.7 Why Rule-Based Before Statistical / ML

✔ Rule-based provides immediate operational safety
✔ ML requires training corpus (we don't have real harness failures)
✔ Statistical RUL requires degradation curves (future work)
✔ OEMs generally certify rule-based conditions easier for functional safety (ASIL-B style)

Strategically:

> **Rule-based catches the fault earliest and provides explainability**
>
> Then ML can estimate **rate of change → RUL**

7.2.8 Summary of Phase 7.1 + 7.2

Topic	Outcome
Failure mechanism	Modeled as voltage deltas
Domain validation	Harness degradation detectable
Signal architecture	Multi-node relational
Diagnostic	Threshold + hysteresis + states
Performance	Real-time + stable + low CPU
Result	Correct alerts for A, B, and shared faults

What comes next (for Phase 7.3 onward)

Now that we can **detect the fault**, next steps are:

7.3 Statistical Anomaly Detection

- detect precursor deviation
- detect trending drift
- detect rate-of-change increasing
- detect failure trajectory

7.4 RUL estimation

(good question earlier — explained again here:)

> **Remaining Useful Life (RUL)** = time until failure boundary

For harness:

RUL ≈ (Δ / dΔ/dt)

Essentially:

```
if drop = 200mV/month
failure = 1.5V
RUL ≈ 6.5 months
```

Even crude estimation has value for predictive maintenance.

8.x Alerts + Logging

Once the logic works, we package:

✔ events
✔ timestamps
✔ state transitions
✔ reasons

And finally:

✔ cloud / email / UI notification