Session 5 — Speaker Script (Debug & Enterprise Refactor)

Trainer-facing runbook with narration, demo steps, and copy-paste prompts

How to Use This Script

- **Bold** = key talking points. *Italic* = stage cues.
- Each section maps to one slide in the "10-Prompt Deck."
- Keep a **sample repo** open (small service with failing tests) and a **log file + stack trace** ready.

Slide 1 — RCA Master Prompt (Stack-trace → Root Cause)

Learning intent: Model a structured, repeatable RCA that turns noise into a fix plan + tests.
Link to earlier sessions: Applies the **5 Golden Rules** (clarity, context, constraints, role, few-shot) and the **Fragile Prompt Checker**.

Demo setup: - Open **Cursor** or **Copilot Workspace**; have a stack trace and 10-minute log slice ready. - Mention NFRs (p95 latency, error rate) and constraints (no API breaking changes).

Narration (what to say): - **"We don't ask 'fix it'. We ask for *timeline, cascade, root cause, fix plan, and tests*. This is guardrailed RCA."** - **"Note the constraints and NFRs—this prevents creative but unsafe fixes."**

Prompt (copy-paste):

```
You are a senior <RUNTIME/FRAMEWORK> engineer.
Inputs:
- Stack trace (triple-backtick block)
- Relevant logs for ±10 minutes (triple-backtick block)
- Constraints: API must remain backward-compatible; no destructive ops
- Non-functional goals: p95 latency < <TARGET_MS>, error rate < <TARGET_%>

Return a structured RCA with:
1) Timeline (timestamped bullets)
2) Cascade map (what failed → why)
3) Probable root cause (1–2 sentences)
4) Fix plan: Immediate, Short-term, Long-term
5) Regression tests to add (names + brief descriptions)
6) Risk assessment (High/Med/Low) with reasoning

Keep total output ≤ 220 lines. Use section headings and numbered lists.
```

Token/time control: Cap lines; paste only the minimal log slice.
Success criteria: Clear 5-part RCA + actionable next steps + proposed tests.
Transition: “We have a narrative. Next, compress logs into a clean **timeline + cascade**.”

Slide 2 — Timeline & Cascade Reconstructor

Intent: Extract a storyline from messy logs, separate **sequence** from **causality**.
Connect: This is the “observability filter” before we debate fixes.

Narration: - **“Timeline tells us *when*. Cascade tells us *why*.”**
- **“We also capture GAPS to drive next probes.”**

Prompt:

```
Act as an incident analyst. From the logs, reconstruct:
A) Timeline: one event per line, "HH:MM:SS — component — event — evidence"
B) Cascade map: bullet tree "trigger → downstream effect → client impact"
C) Missing evidence: list exact probes/metrics/log queries needed

Constraints:
- Focus on the 3 most consequential branches
- Ignore non-causal noise (warn if uncertain)

Return ONLY three sections: TIMELINE, CASCADE, GAPS.
```

Pitfalls & recovery: If it lists irrelevant lines, remind it to **ignore non-causal noise**.
Success: Three compact sections you can drop into the post-mortem.

Slide 3 — Regression Test Author (from RCA)

Intent: Convert RCA into **tests that lock the fix**.
Connect: Ties to earlier **SRD/PRD** discipline—acceptance criteria become tests.

Narration: - **“A fix without tests is a rumor. We turn RCA into code.”**

Prompt:

````
You are a staff SDET. From the RCA below, generate tests:

RCA:
```text
<PASTE_RCA>
````
````

Deliver: 1) Unit tests (names, arrange-act-assert steps, edge cases) 2) Integration tests (preconditions, API calls, expected status/payload) 3) Negative tests (timeouts, invalid inputs, large payloads) 4) Test data scaffolding plan

Output as a Markdown checklist + code blocks for the top 2 tests per category in <LANG/FRAMEWORK>. Limit to 150 lines of code.

```
**Token control:** Limit code lines; ask only for top 2 per category.
**Success:** Ready-to-paste tests + checklist scaffold.

---

## Slide 4 — Static-Analysis & Secrets Guardrail (Table)
**Intent:** Build a **review artifact** the team can paste into PRs.
**Connect:** Extends your **Fragile Prompt Checker**—now for infra & CI.

**Narration:**
- **“Tables reduce waffle. We want _risk, severity, evidence, fix_.”**

**Prompt:**
```text
Security architect mode. Review the repo/CI config below.

Flag and prioritize:
- Destructive commands (e.g., `rm -rf`, `DROP DATABASE`)
- Missing secret management / insecure env usage
- Dangerous permissions (over-broad tokens, sudo)
- Supply-chain risks (pinning, audits)

Return ONLY a Markdown table with columns:
| risk | severity | file:line | evidence | fix |

Repo/CI:
```text
<PASTE_GIT_TREE_SNIPPET_AND_YAML>
```

```
**Success:** One crisp table you can track in an issue.

---

## Slide 5 — CI Quality Gate Generator (Coverage + SAST + Lint)
**Intent:** Codify **quality gates** so future PRs don’t regress.
**Connect:** Aligns to your **“don’t waste tokens”** rule: fail fast in CI.

**Narration:**
- **“We standardize: lint, tests with coverage, SAST, dep audit, PR summary.”**

**Prompt:**
```

```text
Create `.github/workflows/ci.yml` for <LANG/FRAMEWORK> monorepo with jobs:
- Lint
- Unit tests with coverage gate (fail if < 80%)
- SAST (pick common action/tool for <LANG>)
- Dependency audit
- PR comment summarizing results in a Markdown table

Constraints: GitHub Actions only; cache dependencies; run matrix for
<RUNTIME_VERSIONS>.
Return ONLY the YAML.
```

Success: Valid GH Actions YAML you can commit as a PR.

Slide 6 — Shell-Safety Review (No Foot-guns)

Intent: Prevent dangerous shell advice from making it to prod.
Connect: Your **safe-ops** trait from Session 1 comes alive here.

Prompt:

```
Shell safety review. Input is a proposed shell script:
```bash
<PASTE_SCRIPT>
```

Return: 1) Risk findings (numbered): data loss, privilege escalation, globbing pitfalls, quoting issues 2) Safer rewrite (idempotent; uses `-i` prompts or backups; no destructive defaults) 3) Rollback steps and dry-run commands

Restrict to ≤ 120 lines total; use code blocks for the rewrite.

```
**Success:** Safer script + explicit rollback.

---

## Slide 7 — Refactor Road-Map Author (YAML)
**Intent:** Produce a **phase-wise** plan with **owners** and **metrics**.
**Connect:** Mirrors your SRD/playlist method: we plan before we change.

**Prompt:**
```text
Generate `refactor_roadmap.yaml` with phases: Hygiene, Modularize, Optimize.

Schema:
phases[]:
  name: string
```

```
  tasks[]: string
  risks[]: { type, severity }
  success_metrics[]: string
owners[]:
  { area, owner }

Inputs:
- Current pain points: <BULLETS>
- Non-functional targets: <TARGETS>

Cap to ≤ 160 lines. Fill with specific examples based on the inputs.
```

Success: A YAML file you can store in the repo and track.

Slide 8 — Risk Register + Owner Matrix

Intent: Make **risk/owner** visible; enforce accountability.
Connect: This is the bridge from engineering to PM governance.

Prompt:

```
From the refactor_roadmap.yaml (below), generate a risk register matrix.

Return ONLY a Markdown table:
| phase | risk_type | severity | owner | mitigation | due_by | success_metric
|

Then propose 3 cross-phase mitigations (bullets).
```yaml
<PASTE_ROADMAP_YAML>
```

```
**Success:** One table + three mitigations for the plan of record.

---

## Slide 9 — Loop-Breaker & Handoff (Beat Token Spiral)
**Intent:** When a tool “spins,” swap context cleanly to a different agent.
**Connect:** Echoes your best practice: _summarize, constrain, handoff_.

**Prompt:**
```text
Summarize for a fellow AI coding agent:

1) Problem in 3 bullets (clear, testable)
2) What we've tried (bullets, include prompts used)
3) Hard constraints (compatibility, API surface, perf SLOs)
4) Current artifacts (paths/branches)
```

```
5) Ask for: 3 new solution plans (trade-offs), pick 1 to execute now, and 1
fast validation step

Return a compact brief ≤ 200 lines suitable for pasting into another tool.
```

Narration: - **“We don’t fight the loop. We re-brief another agent.”**

Success: Concise, tool-agnostic brief for Cursor/Augment/Copilot PR.

Slide 10 — Executive RCA Summary (Stakeholder-Ready)

Intent: Communicate outcomes **without jargon**; accelerate approvals.
Connect: Ties to Session 2’s **SRD clarity** and stakeholder language.

Prompt:

```
Create an executive RCA summary (≤ 180 words) in this structure:
- What happened (1-2 sentences, no jargon)
- Customer impact (scope, duration)
- Root cause (plain language)
- Fixes implemented (bullet list)
- Preventive actions (bullet list with owners)
- ETA / residual risk

Tone: factual, calm, accountable. Avoid blamey language.
```

Success: A concise note that can go to leadership or incident channel.

Appendix A — Live Demo Assets (prep once)

- **/logs/prod-slice.log** (10 mins around failure)
- **/traces/stack.txt** (exception excerpt)
- **/ci/ci.yml** (weak baseline to improve)
- **/scripts/cleanup.sh** (intentionally risky version for safety review)
- **/refactor/refactor_roadmap.yaml** (starter to enrich)

Appendix B — Timeboxing & Flow

- Slide 1–2 RCA: **12 min** (5 demo, 5 narration, 2 Q)
- Slide 3 Tests: **8 min**
- Slide 4–5 Guardrails/CI: **10 min**
- Slide 6 Shell safety: **6 min**
- Slide 7–8 Roadmap/Risk: **12 min**
- Slide 9 Handoff: **5 min**
- Slide 10 Executive summary: **5 min**

- **Q&A buffers:** 7–10 min

Appendix C — Troubleshooting Playbook

- **Model loops:** Use Slide 9 prompt to re-brief in another tool.
- **Overlong outputs:** Add "≤ N lines" to each section.
- **Hallucinated details:** Ask for **evidence lines** or file:line in outputs.
- **Token burn:** Trim logs, request **top-3** issues/tests only, batch long tasks offline.