Experiment Methodology v0.2 (Draft — copy/paste)

0. Purpose

This methodology defines a reproducible experiment to measure **iterative drift** in AI-assisted software changes, comparing:

- **Spec+Code condition (S)**: task executed with an authoritative specification artifact plus repository code.
- **Code-only condition (C)**: same task executed with repository code only.

The primary focus is **drift/regression under sequential change**, not single-shot success.

1. Fixed Scope Assumptions

- **Language/runtime:** Python project(s) only.
- **Test execution:** runs under a reproducible CI-like environment (documented in run log).

2. Runtime Parameters (Must Be Provided Per Run)

Each experimental run MUST specify and log:

- **N**: number of eligible bug samples to evaluate in the selected repo.
- **W**: follow-on selection window (see §6.3) — defined in one of these units (pick one per run):
    - commit window size, or
    - time window, or
    - PR window size.

These are runtime parameters and MUST NOT be implied from document versioning or prior runs.

3. Hypothesis (Falsifiable Claim)

Given identical repositories and tasks, an implementation process constrained by an authoritative specification (**S**) will exhibit **lower iterative drift** than code-only execution (**C**), measured by fewer regressions and better scope containment under Step 2.

The hypothesis is falsified if **S** does not materially outperform **C** on the pre-registered metrics.

4. Core Definitions

- **A (Baseline)**: repo state prior to the target bugfix.
- **B (Gold Fix)**: repo state after the accepted historical fix.
- **B′**: model-produced fix output from Step 1.
- **C′**: model-produced output after Step 2 follow-on applied to B′.
- **Iterative drift**: regressions or unintended changes introduced by Step 2 relative to B′.

5. Study Arms

Each sample is executed in both conditions:

- **Condition S (Spec+Code)**
  Inputs: repo snapshot + authoritative spec artifact(s) + identical prompts.
- **Condition C (Code-only)**
  Inputs: repo snapshot + identical prompts. No spec artifact(s).

6. Sample Selection (Anti-Cherrypick)

6.1 Repo eligibility criteria (Python)

A repository is eligible only if:

1. It is open source with permissive license for reproduction.
2. It has an executable test command suitable for CI (pytest/unittest/etc.).
3. Historical bugfix commits/PRs can be identified and pinned.

6.2 Bug sample eligibility criteria

A historical fix sample is eligible only if:

1. There exists an identifiable **A→B** bugfix (merged PR or commit).
2. Tests pass at **B** under the declared test command/environment.
3. The change is non-trivial (not formatting-only).

6.3 Runtime selection rule (uses N and W)

The run MUST pre-register one selection rule:

- "First N eligible fixes after date D," OR
- "Randomly select N eligible fixes using seed S."

W is used only to bound search during eligibility determination (e.g., to avoid scanning entire history), not as a definition of "follow-on."

7. Step Definition (What the model must do)

7.1 Step 1 (A→B′): implement the historical fix

Model is given state A and a task statement describing the bug and desired fix.

Step 1 passes only if:

- the test command passes at B′, AND
- explicit acceptance criteria in the prompt/spec are satisfied.

7.2 Step 2 (B′→C′): Synthetic Follow-on (default)

Step 2 is a **synthetic follow-on change** applied after Step 1 completes, designed to test drift.

Critical constraint: Step 2 MUST be **semantically coupled to the same subsystem/intent surface as the bugfix**, not a random feature.

Examples of valid synthetic follow-ons (choose one per sample, derived from the bug's domain):

- broaden an input boundary (additional valid input shape),

- tighten validation or error messaging invariants,
- add a closely related edge case requirement,
- refactor a small internal function while preserving outward behavior,
- add a regression test covering a neighboring case.

Step 2 MUST be written so that:

- it is implementable without additional external context,
- it can be validated by tests (existing or added),
- and it is expected to require touching a limited, relevant part of the code.

7.3 Synthetic Follow-on generation rule (pre-registered)

To avoid “making Step 2 unfair,” each run MUST pre-register how Step 2 is chosen:

- **Template-based** (recommended): choose from a fixed set of follow-on templates (above) and instantiate from bug context, OR
- **Gold-informed but not copied**: derive Step 2 from reading B’s fix intent and tests, without referencing later history.

The chosen rule MUST be logged.

8. Prompt Fairness Rules

8.1 Prompt symmetry

Step 1 and Step 2 prompts MUST be identical between S and C, except:

- S includes spec artifact(s).

No additional hints, logs, or external sources are allowed in one condition but not the other.

8.2 Spec constraints

Spec artifacts (if present) MUST be:

- included verbatim in the run log,
- stable/fixed for the run,
- clearly marked as authoritative vs informative.

9. Interaction Budget

Each run MUST pre-register:

- max turns per step,
- max patch attempts,
- stop conditions (fail fast vs allow repair),
  and record actual usage.

10. Output Artifacts (Reproducibility)

For each sample and condition, capture:

- repo id + commit SHAs for A and B,
- diffs for Step 1 and Step 2,
- test output for A (optional), B, B′, C′,
- prompts and included spec artifacts,
- environment info (Python version, OS, dependency install steps).

11. Metrics

11.1 Primary metric: Step-2 regression rate

Regression = any test that passed at B′ but fails at C′ (under the declared test command).

Compare regression rate S vs C.

11.2 Secondary metrics

- **Scope containment:** count of files changed outside the bug's touched-set + justified exceptions.
- **Churn:** diff size added/removed from B′ to C′.
- **Turn/attempt cost:** total turns/attempts used.
- **Test delta quality:** whether Step 2 produced an appropriate new regression test (when applicable).

12. Reporting

The report MUST include:

- the runtime parameters (N, W) and selection rule,
- per-sample outcomes for S and C,
- aggregated metrics and failure cases,
- negative results if S does not outperform C.