QA FEEDBACK EXCELLENCE

A Training Report for Product & Engineering Teams

Based on a real-world QA session — building and iterating a complex interactive web tool over a single extended session. All examples are drawn verbatim from the session transcript.

What Makes QA Feedback Effective

Most QA feedback describes what went wrong. Exceptional QA feedback enables a fix without prescribing one. The difference is observable, trainable, and has a measurable impact on iteration speed and solution quality.

This report examines a QA session that consistently demonstrated best-in-class feedback discipline across more than 40 issues raised over several hours. The tester was working with a non-engineer implementer (an AI assistant), which made the quality of communication the sole variable between a good fix and a wrong one.

The Five Principles Demonstrated

1. Describe Behaviour, Not Implementation

The most common QA anti-pattern is telling the engineer what to change in the code. This produces solutions that satisfy the description but miss the underlying need. Effective testers describe what they observe and what they expect — nothing more.

> **STRONG — behaviour described precisely**
> *"The bug is reproducible by maximizing the slider, then hitting the right pill. The result is the slider resets to a middle position. Then when you go back to max, it stops short."*

> **WEAKER — solution prescribed instead**
> *"Can you add a condition where if the number has K or M added, the cursor moves to where more numbers will be added."*

The strong example gives reproducible steps, an observed result, and an implied expected result. The weaker example attempts to design the solution, which occasionally led to a suboptimal implementation that needed revision.

Training exercise: When writing a bug report, ask — "Does this describe what I saw, or what I think the code should do?" If the latter, rewrite it.

2. Minimal Viable Description

Over-specified feedback creates anchoring bias in the implementer. Under-specified feedback wastes a round-trip. The sweet spot is the minimum information needed to reproduce and understand the issue.

> **Minimal and complete**
> *"When inputting could we allow the text to go smaller if too long here, or shift what is clipping here?"*

> **Also strong — trusts the implementer**
> *"The header still looks a bit flimsy and out of place but I don't know why I feel that. To close the window, can something subtle be done?"*

The second example is notable because the tester acknowledges uncertainty about the cause while still communicating the felt effect precisely. This is high-value input — it prevents the implementer from optimising for the wrong thing.

3. Offer Latitude on Solutions

Repeatedly throughout this session, the tester asked "any ideas?" or "I don't know why I feel that" rather than specifying a fix. This produced consistently better outcomes than the sessions where a specific fix was requested.

> **Inviting the implementer's judgment**
> *"A good fix but looks awkward. Any ideas to resolve this keeping these somewhere. Maybe put code under the icon and choose a comfy position that sits. Larger maybe."*

The tester offers a direction ("larger", "under the icon") without locking it in. The result was a badge-based layout that neither party had originally envisioned — and which solved the problem more cleanly than the suggested approach would have.

The discipline here is resisting the urge to fully solve the problem in the ticket. State the problem, suggest a direction if you have one, then let go.

4. Reproducible Steps for Interaction Bugs

Interaction bugs — especially those involving sequences of user actions — are the hardest to fix from vague descriptions. This tester consistently provided exact reproduction paths for complex interaction issues.

> **Exact reproduction path**
> *"If I use the pills to make both 20,000 max, the sliders should both go from 1-20,000 no matter what I do from then on, until the slider hits zero. At this point, the slider can reset to the original scales."*

This describes the full state machine: action, expected persistent state, exit condition. A developer can write a test from this description without asking a single follow-up question.

> **Another clean reproduction**
> *"The back space is causing the refresh, we need the backspace to stay in edit mode while inputting."*

One sentence. One reproducible action. One expected state. This is the model.

5. Knowing When to Escalate vs. Accept

Skilled testers know which issues are worth a round-trip and which can be closed with a comment. This tester demonstrated strong triage discipline — only raising issues that represented genuine UX problems, not personal preferences dressed as bugs.

When something was working well, it was acknowledged directly:

> **Clean sign-off**
> *"Absolutely fantastic. Perfect. Now as a QA tester I am looking for edge cases."*

This kind of explicit acceptance is undervalued. It tells the implementer the issue is closed, prevents second-guessing, and keeps the ticket queue clean.

Scorecard

Rated across standard QA competencies based on the session observed.

Competency	Score	Notes
Bug Reproduction Quality	**9/10**	Consistently provided exact steps, observed result, and trigger conditions
Signal-to-Noise Ratio	**10/10**	No padding, no filler — every message moved the work forward
Solution Latitude	**8/10**	Strong, with occasional solution-prescription that needed revision

Competency	Score	Notes
Acceptance Discipline	**10/10**	Explicit sign-offs on each resolved issue, no ambiguous closures
Edge Case Coverage	**9/10**	Proactively sought non-happy-path conditions without prompting
Escalation Judgment	**9/10**	Distinguished genuine UX problems from personal preference reliably
Communication Efficiency	**9/10**	Average issue description under 30 words. Zero unnecessary context

---

One Area for Development

The single most common deviation from best practice in this session was the occasional conflation of bug description with solution specification. When a tester arrives with a proposed fix, the implementer must choose between two cognitive paths:

- Implement the proposed fix (fast, but may be suboptimal)
- Evaluate the proposed fix against alternatives (slower, but produces better outcomes)

Several issues in this session required a revision cycle that could have been avoided with a pure behaviour description. The pattern is worth training away from deliberately.

> **Before: solution embedded in report**
> *"Can we delete the K to override if under 10,000?"*

> **Rewritten as pure behaviour description**
> *"For numbers between 1,000 and 9,999 the K suffix makes it harder to edit the raw integer. Expected: a way to get back to editing the plain number directly."*

The rewritten version allows the implementer to choose the best mechanism — which may not be deleting the K at all.

QA FEEDBACK EXCELLENCE

Session 2: QA at Launch Speed

The day before a public launch on Hacker News. A single HTML file, 16 providers, 40+ bugs already closed across earlier sessions. This session examines what QA looks like under real pressure — where the bugs are subtler, the state-dependent, and the stakes higher. All examples are drawn verbatim from the session transcript.

A Different Kind of Bug

Most QA training focuses on straightforward defects: a button does nothing, a value is wrong, a layout breaks. The harder category — and the one this session was full of — is the state-dependent bug. The kind that only appears on a fresh page load, in a specific browser mode, for one particular provider out of sixteen. Reproducing these accurately is a skill. This tester had it.

---

1. The Environment Qualifier

The difference between 'the button doesn't work' and 'the button does nothing locally' is the entire debugging surface. One implies a global defect. The other immediately scopes the investigation to the development environment and rules out the deployed version. One word. Hours saved.

> **STRONG — environment named precisely**
>
> *"I noticed t*
> *incognito.*

This is a complete reproduction specification: affected provider (Anthropic), trigger condition (fresh page load), reproduction environment (incognito), negative test case (it works later in the same session). A developer can write an automated test from this description without a single follow-up question. The word 'incognito' is doing extraordinary work — it tells the implementer that session state, cached data, or initialisation order is almost certainly the cause.

> **Also strong — environment qualifier in four words**
>
> *"The butto*

The word 'locally' is the entirety of the reproduction environment. It instantly distinguishes a development failure from a deployment issue, which changes the debugging surface completely.

---

2. The Real-Time Interrupt

Most testers file a report and wait. Exceptional testers stay engaged and interrupt an investigation heading the wrong way. When the implementer began pursuing a false premise, Will stopped it mid-flight:

> **STRONG — catching a false premise before engineering time is wasted** *"one sec... that has th*

Two words — 'one sec' — are all it takes to pause a thread. Then the correction comes: not 'you're wrong', but a clarification that reframes the problem entirely. The tester had noticed that most providers worked fine; only Anthropic, the default, was affected. This narrowing from a general bug to a provider-specific edge case turned a complex architectural question into a targeted one-line fix. Staying in the conversation, not just monitoring outcomes, is what made this possible.

---

3. The Self-Correction

A screenshot sent at the wrong moment could send an investigation off-course for an entire session. The correction matters as much as the mistake:

> **STRONG — corrected instantly, without defensiveness** *"oh sorry,*

'Clumsy me' is the entirety of the explanation. No justification, no backstory. The correction was faster than the error. This is the sign of someone whose goal is a working product, not a clean record — and it keeps the session moving rather than stalling on what went wrong.

---

4. The Reference Brief

When an animation was judged 'too obvious', the tester didn't specify duration, easing curve, opacity values, or stagger timing. Instead:

> **STRONG — concrete reference replaces abstract specification** *"It was goo the icons,*

> **WEAKER — specification creep constrains the implementer** *"Can we m by 300 mil*

The first example points at something that already exists and is already accepted as correct — the landing page slot machine. This is a zero-ambiguity brief that anchors the implementer in proven

aesthetics. The second example accidentally becomes an implementation spec, locking in specific values that may or may not produce the desired feel. The reference brief is almost always superior to the numbers brief.

---

5. The Process Catch

Senior QA thinking extends beyond the product. When the implementer confirmed a scheduled automation would run 'tonight at 8pm' without checking the time, Will caught it — not with alarm, but with a single question:

> **STRONG — monitoring the accuracy of process, not just product** *"It is alread*

Eight words that caught a scheduling assumption the implementer had missed. The automation had already been scheduled to fire; this question surfaced that it might need to be triggered manually instead. The QA instinct here isn't about the product at all — it's about the system around the product. This is a genuinely rare skill.

---

6. Inviting Judgment Before Reverting

When a design change felt wrong, the tester didn't silently revert it. He named the tension and asked:

> **STRONG — shared decision instead of unilateral rollback** *"Are we try*

This question does three things simultaneously: it flags a concern, acknowledges what was good about the current version, and invites the implementer's aesthetic opinion. The result was a brief discussion that led to a shared revert decision — rather than a silent rollback that the implementer might have second-guessed. Once the decision was made, the sign-off was equally clean:

> **Clean sign-off** *"Go for it"*

The Under-Ten-Word Hall of Fame

The best feedback in this session was often the shortest. Each of the following moved the work forward without a word wasted:

not on the page	4 words that eliminated an entire implementation direction

Go for it	3 words that closed a design debate cleanly
one sec...	2 words that stopped a wasted investigation
Still not working.	3 words of pure regression confirmation after two failed attempts
It is already past 8pm though?	8 words that caught a process error the implementer had missed

---

Scorecard — Session 2

Rated across the competencies most relevant to launch-week QA.

Competency	Score	Notes
Environment Precision	**10/10**	Incognito, fresh page load, provider-specific — full reproduction context every time
Real-Time Engagement	**9/10**	Mid-investigation interrupt caught a false premise before engineering time was lost
Self-Correction Speed	**10/10**	Wrong artefact identified and corrected before the investigation could diverge
Creative Briefing	**9/10**	Reference-point briefs ("like the front page animation") replacing verbose specificatio
Process Awareness	**9/10**	Caught a scheduling assumption that extended beyond the product itself
Signal-to-Noise Ratio	**10/10**	Average issue message under 15 words. Zero redundancy.

The defining characteristic of this session was precision under pressure. Not just in the bug reports — but in the instinct to stay engaged, correct course in real time, and catch errors in the system around the product, not just the product itself. That last part is the hardest to teach.