SINCE 1994

# The fine art of writing a *good bug report.*

TEN TIPS THAT TURN SLOPPY NOTES INTO A DOCUMENT DEVELOPERS WILL ACT ON · REX BLACK, INC.

# A bug report is a *technical document.*

Not an escalation. Not a vent. Not a joke.

---



**Accurate. Concise. Well-conceived. High-quality.**
The only tangible product the test team ever ships.

WHAT THIS IS ABOUT

# In plain English.

Most bug reports that get kicked back as "can't reproduce" *aren't* unreproducible. They are *poorly written*. The failure was real — the report was bad.

---

This talk shows the ten habits that turn a one-line rant into a crisp, reproducible report management can read and developers can fix:

- **What counts as reproducible** — and how many tries.
- **What to isolate, generalize, and compare** before you submit.
- **How to write the one line** management will actually read.
- **How to neutralize tone** before you hit save.

TIP 01 OF 10 · FOUNDATION

# Structure.

# *Test carefully.*

TIP 01 · STRUCTURE

# Reports rest on *structured testing.*

Bug reporting begins the moment **expected** and **observed** results diverge. Which means you need the expected result in writing before you run the test.

---

DO

- Deliberate, documented approach.
- Written or automated test cases.
- Notes as you go.

DON'T

- Ad-hoc clicking with no record.
- "I think I saw it earlier."
- Writing the report from memory.

TIP 02 · REPRODUCE

# Test it again.

Always verify reproducibility before writing the report. **Three tries** is a reasonable rule of thumb.

---

Document a **crisp sequence of actions** that reproduces the failure. State the incidence rate openly if it's intermittent ("reproduced 2 of 3").

Clean steps to reproduce head off the *can't reproduce* bounce before it starts.

TIP 03 · ISOLATE

# Test it *differently.*

Change **one variable at a time.** Watch whether the symptom changes.

---

Isolation takes thought and understanding of the system. It can be as much work as the original test run — **match effort to severity.**

The goal is not to debug. The goal is to give the developer a head start by eliminating the obvious dead ends.

TIP 04 · GENERALIZE

# Test it *elsewhere.*

Does the same bug show up in other modules, with other data, on other platforms? Are there **more severe** occurrences of the same fault?

---

Generalizing reduces duplicate reports, and it often reveals that the real bug is **two layers down** from the symptom you first caught.

TIP 05 · COMPARE

# Review results of *similar tests*.

Did the same test pass against an earlier build? On which build did it *start* failing?

---

Not always possible — the feature may be new, or the old build impractical to reinstall. But when you can answer **"new to build X,"** you just cut the developer's search space in half.

TIP 06 · SUMMARIZE

# Relate the test to the *customer.*

**One line.** Capture the failure and its impact on the customer.

---

Harder than it looks. A good summary:

- Gets **management attention** — someone reads the tracker every morning.
- Becomes the **bug's name** to developers.
- Helps **priority sort itself out** without a meeting.

Spend real time on it.

TIP 07 · CONDENSE

# Trim unnecessary information.

Everyone's time is precious. **Don't waste any on verbiage** — and don't cut any meat either.

---

Cryptic one-liners are as bad as droning on. The target is *clear prose at minimum length*.

TIP 08 · DISAMBIGUATE

# Use *clear words.*

Remove, rephrase, or expand vague, misleading, or subjective statements. The goal is **clear, indisputable statements of fact** that cannot be misread.

---

RHETORIC

"Trashed the text."

FACT

"Converted all text to control characters, numbers, and apparently random binary data."

Lead the developer by the hand to the bug.

TIP 09 · NEUTRALIZE

# Express the problem *impartially.*

Deliver bad news gently. **Be fair-minded** in wording and implications.

---

Do not attack developers. Do not criticize the underlying error. Do not try to be funny. Do not use sarcasm.

**You never know who reads it** — executives, auditors, customers, lawyers. Write for that audience every time.

TIP 10 · REVIEW

# Be sure.

Every tester submits each bug report to **one or more peers for review.**

---

Reviewers make suggestions, ask clarifying questions, and challenge whether the thing really is a bug when that's warranted.

The test team ships the **best possible report** — at a time budget appropriate to the bug's priority.

WORKED EXAMPLE

# The *SpeedyWriter* font-corruption bug.

One bug. Eight drafts. Each draft adds one of the tips.

# From rant to reproducible.

### DRAFT 0 · BAD

"Nasty bug trashed contents of new file that I created by formatting some text in Arial font, wasting my time."

### DRAFT 1 · + REPRODUCE

Steps 1–5 with crisp actions. Reproduced 3 of 3 tries.

### DRAFT 2 · + ISOLATE

Saved and reopened — garbage remained.
Saving *before* Arializing prevents the bug.
Doesn't occur with existing files.
Only under Windows 98.

# Generalizing, comparing, summarizing.

## DRAFT 3 · + GENERALIZE

Also happens with Wingdings and Symbol fonts.
Rewords the isolation broadly: *if you save before changing the font, the bug does not occur.*

## DRAFTS 4 & 5 · + COMPARE, SUMMARIZE

**New to build 1.1.018.** Same test case passed 1.1.007 → 1.1.017.

Summary line: *Arial, Wingdings, and Symbol fonts corrupt new files.*

# The finished report.

**Summary:** Arial, Wingdings, and Symbol fonts corrupt new files.

**Steps to reproduce:** (1) Started SpeedyWriter. Created a new file. (2) Typed four lines. (3) Highlighted all four lines, pulled down the font menu, selected Arial. (4) All text converted to control characters, numbers, and other apparently random binary data. (5) Reproduced 3 of 3 tries.

**Isolation:** New to build 1.1.018. Same pattern with Wingdings/Symbol. Saving before font change prevents it. Doesn't occur with existing files. Only under Windows 98 — not Solaris, Mac, or other Windows flavors.[18]

TAKEAWAYS

# Ten tips, one document.

# The basics never change.

- **A bug report is a technical document.** Four words to hold it against: accurate, concise, well-conceived, high-quality.
- **The summary is half the value.** One line. Spend real time.
- **Isolate before you submit.** Change one variable at a time.
- **Neutralize — always.** You never know who reads it.

# Review pays for itself.

- **Review each other's reports.** Single cheapest quality lever on a test team.
- **Process pays off.** Faster bug lifecycles. Fewer reopens. Better tester–developer relationships. Credibility with senior management.

---

> *Software testing should be more like a carefully designed laboratory experiment than a random walk in an electrical storm. Testers are engineers, after all.*

SINCE 1994

# Thank you.

REX BLACK, INC. · REXBLACK.COM/RESOURCES/TALKS/WRITING-A-GOOD-BUG-REPORT