SINCE 1994

Investing in software testing.

THE ROI CASE AND HOW TO EARN IT · REX BLACK, INC.

"What does testing *actually return?*"

If the answer is *"it's just good practice,"* the budget shrinks.

If the answer is **"here's the baseline, here's the investment, here's the measurable change in customer-found defects and cost,"** the budget holds. Sometimes grows.

WHAT THIS IS ABOUT

In plain English.

This talk gives engineering leaders the number they need to defend the testing budget in a CFO's language: **return on investment.**

Walks through a **concrete worked example** answering four questions:

- **What does testing cost** — and what does it return vs. no testing at all?
- **Where do you get 2× or 4× on your test dollar?** Manual team? Automation?
- **How do you pick the tests** that actually reduce customer-found defects?
- **What's the operating model** that makes the numbers repeat every release?

445%

ROI IN THE WORKED EXAMPLE — A $150K TOOLING INVESTMENT, AMORTIZED OVER 12 RELEASES.

REX BLACK, INC. · INVESTING IN SOFTWARE TESTING

START WITH THE BASELINE

Stage 0 — *developer testing only.*

No dedicated testing team. Developers test their own code.

- **Every bug that ships is a customer-found bug.**
- Support cost per incident and engineering cost per hotfix at full rate.
- Quality costs are whatever they are — and *nobody's measuring.*

This is the number the next two stages move.

STAGE 1 — MANUAL TESTING

One testing team.

50% fewer customer bugs.

Add a dedicated manual testing team to a developer-tested project.

FOUND INTERNALLY

Developers · 250 bugs
Testers · **350 bugs**

Two to three bugs internally for every one developers were catching alone.

THE NEXT QUESTION

Can we reduce further **without doubling the team?**

(Stage 2 answers yes.)

STAGE 2 — MANUAL + AUTOMATION

66% fewer. *Quality cost halved.*

$150,000 tooling investment, amortized over **twelve quarterly releases** = $12,500/release.

- **Automation** picks up regression, load/volume, performance, structural API checks, standards compliance.
- **Manual** retains usability, localization, error handling, configuration, installation.
- Customer-found bugs **–66% vs. baseline.** Quality costs **halved.** Return on tooling investment: **445%.**

THE CATCH

ROI comes from *the right tests.*

Pick the wrong ones — test investment return is **zero.**
Pick the worst ones — it's **negative** (false sense of security).

Usage *fidelity* is the lever.

HIGH-FIDELITY (INVEST HERE)

Real data volumes. Real configurations. Real workflow sequences. Real transaction mixes. Real latency and failure conditions.

Finds customer-critical bugs before the customer does.

LOW-FIDELITY (DON'T BOTHER)

Developers' happy path. Clean data. Ideal environment.

Mostly proves the code compiles.

Spend more on fidelity for the **highest-risk subsystems.** Less for the rest.

WHERE TO INVEST

Quality risk *categories.*

Quality is fitness for use. Risks are the potential for dissatisfying behaviors.

- Functional — missing or broken features.
- Use cases — features fine alone, workflows broken.
- Robustness — errors not handled.
- Performance — too slow at key points.
- Localization — dates, language, money, culture.
- Usability — it works, but what a pain.
- Capacity — can't handle large datasets.
- Reliability — crashing, hanging, misbehaving.

Most programs **overweight functional.** Look at your customer-found defect profile — the

THREE WAYS TO ANALYZE

Pick the process your org tolerates.

INFORMAL

Classic quality-risk categories (above). **For teams that will not tolerate heavy process.**

ISO 9126 / FRUEMP

Six main quality characteristics decomposed into subcharacteristics. **For teams already using standards.**

FMEA

Failure Mode and Effect Analysis. Failure modes × severity × priority × likelihood → Risk Priority Number. **For safety-critical, regulated, or enterprise work** that needs a defensible audit trail.

All three get you to the same outcome: **tests weighted by risk.**

TECHNIQUES · THREE FAMILIES

Static · Structural · Behavioral.

STATIC

Tests **without running the code.** Inspections, reviews, static analysis. Finds bugs before they're built.

STRUCTURAL

Inside the system. Unit, component, integration. White-box. Owned by developers.

BEHAVIORAL

Outside the system. Integration, system, acceptance. Black-box. Owned by testers.

THE INVESTMENT

Three families. Different tools, different phases, different owners. **Cross-pollinated** data, cases, harnesses.

AUTOMATED OR MANUAL?

Pick the right tool per *test type*.

MANUAL-FIRST

Operations & maintenance · configuration & compatibility · error handling · localization · usability · installation · documentation.

AUTOMATION-FIRST

Regression & confirmation · monkey/random · load/volume/capacity · performance · reliability · standards · API structural · static analysis.

EITHER OR COMBINED

Functional · use cases · UI · date/time.

THE TRAP

Inappropriate manual testing misleads about coverage.

Inappropriate automation usually just fails.

OPERATING MODEL

Pervasive testing.

Not a phase.

FOUR QUALITIES

What makes the math *repeat*.

- **Concurrent** — starts with requirements, happens across the whole lifecycle.
- **Cross-functional** — sales defines requirements, support provides usage profiles, devs write specs and do structural tests, testers build tools and do behavioral tests.
- **Collaborative** — dependencies throughout the project team.
- **Committed** — teams deliver what they promise on schedule so testing has something to test.

Pervasive testing turns the ROI math from a one-time projection into a repeating number.

Frame the return.

Earn it with the right tests.

TAKEAWAYS

Four to hold against.

- **Investment, not expense.** Run the math before you argue for the budget, not after you lose it.
- **The right tests.** Usage profiles first, risk analysis second. Tests that cover actions customers never perform return zero.
- **Technique matches risk.** Static, structural, behavioral each find different bug classes. Wrong technique wastes money and misses the bug.
- **Pervasive, not phased.** Lifecycle-wide testing compounds. Two-weeks-at-the-end testing does not.

> *Don't waste money testing actions customers will never perform, verifying configurations they don't use, and fixing bugs they will never see.*

SINCE 1994

Thank you.

REX BLACK, INC. · REXBLACK.COM/RESOURCES/TALKS/INVESTING-IN-SOFTWARE-TESTING