SQA Audit Report: EventHub Project

External Quality Assurance Assessment

COMP 2136: Software Quality Assurance Assignment 3: Group SQA Audit & Improvement Plan

Submitted by: Ben Morrison, 101572409

Due Date: November 28, 2025

Introduction

Project Overview

EventHub is a mobile-friendly web application developed by Local-Connect Inc. to help local community members find and post events such as farmers' markets, charity runs, and free concerts. The project is currently in public beta with a small team of four developers (two senior, two junior) and one part-time Project Manager. Notably, there is no dedicated QA engineer or team.

The Crisis

EventHub is experiencing significant quality problems that threaten its viability:

- Numerous bugs reported by beta users, including critical data leaks and payment failures
- Website crashes and usability issues consuming PM's time
- Pressure to launch "Premium Event" features to secure additional funding
- Growing investor concerns about project stability
- User complaints escalating to social media

The Project Manager hired our external SQA consulting team to conduct an independent audit, identify root causes of quality problems, and provide actionable recommendations for improvement.

Audit Scope

Our audit analyzed nine artifacts from the EventHub project, including:

- Project documentation and team structure (Artifact 1)
- Bug tracking system (Artifact 2)
- Internal communications about testing strategy (Artifact 3)
- Meeting notes and RACI charts (Artifacts 4-5)
- Code samples and unit tests (Artifact 6)
- Performance test plans (Artifact 7)
- Incident reports and acceptance criteria (Artifacts 8-9)

We evaluated the project across four critical dimensions:

1. Testing Strategy & Process
2. Defect Tracking & Management
3. Process, Culture & Roles
4. Ethical & Professional Concerns

Our findings reveal systemic quality issues requiring immediate attention.

Audit Findings

Testing Strategy & Process

Requirements & Acceptance Criteria (Artifact 9)

Premium Events acceptance criteria contain **vague, untestable language**: "simple and user-friendly" (AC-1), "respond quickly" (AC-3), "work most of the time" (AC-7). Missing critical details about what premium events actually get, payment methods, failure handling, and security. AC-5 requires "should not introduce any new bugs" - impossible to test. This violates SMART principles - requirements must be specific, measurable, and testable.

Test Strategy (Artifact 3)

Alex describes their approach as "pretty informal" - "We're moving too fast for a heavy process." Testing consists of "another dev does a quick 'buddy check' - just click around for 5 mins to see if anything is obviously broken." They rely on users to "catch most bugs" since "that's what 'beta' is for."

Critical issues:

- No documented test strategy, scope, entry/exit criteria, or test environment
- "5-minute buddy check" has no test cases, scenarios, or requirements traceability
- Claims to use users as QA but then ignores their reports (Bug 109 reported 3 times, still open; Bug 111 dismissed as "caching blip")
- No regression testing to prevent breaking existing features

Test Levels & Roles (Artifact 5 - RACI Chart)

RACI violations:

- "Write Test Plan" has TWO Accountable parties (Mike & Alex) - violates RACI principle
- No "A" for Execute System Tests, Execute Unit Tests, or Execute Integration Tests
- Junior devs (Ben, Sara) perform system testing without senior oversight or QA review
- Final Quality Sign-off by developers (Alex, Kenji), not independent QA - PM not even informed

No dedicated QA role (Artifact 1) - developers test their own code, creating conflict of interest and confirmation bias. Violates separation of duties principle.

Performance Testing (Artifact 7)

Alex's plan confuses test types - calls a Load Test a "Stress Test", calls 10-minute ramp-up a "Spike Test" (should be sudden). **Subjective criteria**: "site feels fast" - unmeasurable. **Catastrophic error rate**: "below 10%" acceptable (industry standard: <1%). No baseline metrics, no specific response time targets, no monitoring plan. Final line reveals motive: "good enough to tell investors" - testing for optics, not quality.

Unit Test Quality (Artifact 6 - EventTest.java)

Shared mutable state - single `testEvent` instance used across all tests violates independence. `testPriceAndDate()` explicitly states "depends on testSetPrice() running first" - execution order matters, tests fail in isolation.

Missing test cases: boundary values (price=0.0), null inputs, error conditions, initial state.

Poor assertions: Uses `System.out.println()` instead of `assertEquals()` - test passes even if values wrong. Violates F.I.R.S.T. principles (Fast, Independent, Repeatable, Self-validating, Timely).

Defect Tracking & Management

bugs.xlsx Issues (Artifact 2)

Inconsistent lifecycle: Status values like "???", "Looked at", "blank" - no standardized workflow. Bug 110 completely blank.

Missing fields: No Severity, Priority, Assigned To, or Steps to Reproduce. PM marks Bug 106 "CRITICAL!!" in notes but no proper field. Can't prioritize objectively.

Vague descriptions: "create event broken" (Bug 101), "wrong date" (Bug 109) - not reproducible.

Dismissive resolutions: Bug 102 (firefox login) → Alex: "works for me in chrome" (different browser!). Bug 111 (data leak) dismissed as "caching blip" without investigation. Bug 107 performance issue marked "Won't Fix" - "not a bug."

Bugs lost/ignored: Bug 109 reported 3 times, still open. Bug 111 deliberately hidden from tracker to "avoid panicking investors" (Artifact 4).

Excel inadequacy: No workflow enforcement, audit trail, or traceability. Violates defect lifecycle management principles.

Process, Culture & Roles

Firefighting Culture

Artifact 4: "Premium Events MUST SHIP by EOD Friday" (underlined 3x). "All devs focus 100% on Premium." PM spends "most time handling complaints" (Artifact 1). Deadline-driven, feature-focused with no quality gates creates vicious cycle.

Quality as Obstacle

Artifact 3: "moving too fast for a heavy process." Quality seen as obstacle to speed, not enabler. Bug fixing conditional - "if time, fix cosmetic bugs" (Artifact 4).

Users as QA (Then Ignored)

Artifact 3 claims "users will catch bugs" but Bug 109 reported 3x (still open), Bug 111 dismissed and hidden. Cognitive dissonance.

No Quality Ownership

No QA role (Artifact 1). Devs test own code (confirmation bias). Alex (Sr. Dev) has influence but unclear accountability, dismisses bugs subjectively.

Ethical & Professional Concerns

Hiding Security Vulnerabilities (Artifact 4)

Bug 111 (data leak - "users can see other users' private drafts") dismissed as "caching blip." **PM Note: "Let's keep this out of bugs.xlsx for now, don't want to panic investors."** Deliberate concealment violates fiduciary duty, creates legal liability, exposes users.

Faking Security Fix (Artifact 8)

PM to Sara: "I don't care if bug is real. Just add code that shows [Jane's] account a popup: 'Your account security has been updated.' That way we can tell her we've 'fixed' it and she'll stop posting."

This is fraud - deceiving user without fixing vulnerability. Violates ACM Code 1.2 ("Avoid harm"), 1.3 ("Be honest"), IEEE ethics ("honest reporting"), professional responsibility.

Performance Testing as Marketing (Artifact 7)

Plan concludes: "good enough to tell investors we are performance tested." Testing for optics, not quality assurance.

Ignoring User Reports

Jane reports Bug 111 with screenshots → dismissed. Bug 109 reported 3 times → still open. Claims users are QA, then ignores them.

Prioritized Recommendations

Based on our audit findings, we recommend the following prioritized actions to address EventHub's quality crisis:

1. Immediately Address Critical Security Vulnerability (Priority: CRITICAL)

Why: Bug 111 represents an active data privacy breach where users can view other users' private event drafts (Artifact 2, 4, 8). This issue has been:

- Reported multiple times by user Jane with screenshot evidence (Artifact 8)
- Dismissed by Alex as a "caching blip" without proper investigation (Artifact 4)
- Deliberately hidden from the bug tracker to avoid "panicking investors" (Artifact 4)
- Subject to a proposed fake "fix" to silence the user rather than solve the problem (Artifact 8)

This violates ACM Code of Ethics 1.2 ("Avoid harm") and 1.3 ("Be honest and trustworthy"), creates legal liability under data privacy laws, and exposes all users to ongoing privacy violations.

How:

- **Immediate action (today)**: Stop all Premium Event development. Assign two senior developers (Alex and Kenji) to reproduce and investigate Bug 111 using Jane's screenshots and account details.
- Create a security incident response process following industry best practices (e.g., NIST SP 800-61)
- Properly log the vulnerability in the tracking system with Severity: Critical
- If confirmed, develop and deploy a genuine fix within 48 hours
- Notify affected users transparently about the issue and resolution
- Conduct a post-incident review to understand why the issue was dismissed and hidden

2. Adopt a Formal Defect Tracking System (Priority: HIGHEST)

Why: The current bugs.xlsx spreadsheet (Artifact 2) is fundamentally inadequate:

- Inconsistent status values ("???", "Looked at", "blank") make workflow impossible
- Missing mandatory fields (Severity, Priority, Assigned To, Steps to Reproduce)
- No audit trail - can't track who changed what when
- Enables bugs to be hidden or "lost" (Bug 110 is completely blank; Bug 111 kept out intentionally)
- No traceability to requirements or test cases
- Vague descriptions prevent reproduction (e.g., "create event broken", "wrong date")

This violates fundamental defect lifecycle management principles (course Week 6) and contributes directly to bugs being ignored (Bug 109 reported 3 times, still open).

How:

- **Within 1 week**: Migrate to a proper tool (Jira, GitHub Issues, or Azure DevOps)
- Implement a standard workflow: New → Assigned → In Progress → Ready for QA → Verified → Closed
- Make these fields **mandatory**: Title, Description, Steps to Reproduce, Expected Result, Actual Result, Severity, Priority, Assigned To, Environment
- Train team on bug report best practices: specific, reproducible, evidence-based
- Establish clear Severity definitions (Critical/Major/Minor/Cosmetic) and Priority levels (P0-P3)
- Migrate all existing bugs from Excel, properly categorizing and filling in missing information

3. Establish Independent QA Role and Clear Quality Accountability (Priority: HIGHEST)

Why: EventHub has no dedicated QA engineer (Artifact 1), creating a critical conflict of interest:

- Developers test their own code, creating confirmation bias
- No independent verification of quality before production
- RACI chart (Artifact 5) shows no "Accountable" person for system testing
- Junior developers (Ben, Sara) execute system tests without senior oversight
- Final quality sign-off performed by developers (Alex, Kenji), not independent QA
- PM not even Informed of final quality decisions

This violates separation of duties and the principle that "testing should be independent of development" (course Week 3). It directly contributes to the current crisis where bugs reach production.

How:

- **Within 2 weeks**: Hire at least one dedicated QA engineer (can be contractor initially)
- Revise RACI chart to fix violations:
  - Only ONE person can be Accountable for each task (fix "Write Test Plan" having two A's)
  - Assign QA engineer as "A" for: Execute Integration Tests, Execute System Tests, Final Quality Sign-off
  - Senior developers should be "C" (Consulted) on system testing, not absent
  - PM must be "I" (Informed) on Final Quality Sign-off
- Implement quality gates: No code reaches production without QA approval
- Apply Test Pyramid principles (course Week 4): Many unit tests (dev-owned), fewer integration tests, focused system tests (QA-led)

4. Implement Formal Test Strategy and Test Planning (Priority: HIGH)

Why: Alex describes their approach as "pretty informal" and "moving too fast for a heavy process" (Artifact 3):

- No documented test strategy, scope, entry/exit criteria, or test environment
- Testing consists of "another dev does a quick 'buddy check' - just click around for 5 mins" (Artifact 3)
- No test cases, scenarios, or requirements traceability
- Relies on "users will report bugs" - but then ignores user reports (Artifact 2: Bug 109 reported 3x)
- No regression testing to ensure new changes don't break existing features
- Alex advises Mike to "just click around for 5 minutes" (Artifact 3) - this is not testing

This ad-hoc approach violates IEEE 829 test planning standards and directly causes the quality issues driving the current crisis.

How:

- **Within 1 week**: Create a lightweight Test Strategy document defining:
  - Test levels: Unit (dev-owned), Integration (dev-led, QA-reviewed), System (QA-led), Acceptance (PM/stakeholder)
  - Entry/exit criteria for each test level
  - Test environment requirements (browsers, devices, test data)
  - Regression test approach (automated smoke tests at minimum)
- For each feature, write a Test Plan containing:
  - Test scope (what's in/out)
  - Test cases mapped to requirements (traceability)
  - Test data requirements

  - Pass/fail criteria
- Reference: IEEE 829 Standard for Software Test Documentation (course Week 5)
- Start with the Premium Events feature - write proper test plan before launch

5. Fix Unit Test Quality Issues (Priority: HIGH)

Why: The provided EventTest.java (Artifact 6) demonstrates poor unit testing practices:

- Shared mutable state (`private Event testEvent`) violates test independence
- Explicit test interdependency: "This test depends on testSetPrice() running first"
- Tests will fail if run in different order or in isolation
- Missing critical test cases: boundary values (price = 0.0), null inputs, error conditions
- Uses `System.out.println()` instead of proper assertions - test passes even if date is wrong
- No test for initial object state

These issues mean unit tests provide false confidence and won't catch bugs, undermining the Test Pyramid (course Week 4).

How:

- **Immediate action**: Refactor EventTest.java:
  - Remove shared state - create new `Event` instance in each `@Test` method or use `@BeforeEach`
  - Split `testSetPrice()` into two independent tests: `testSetPriceValid()` and `testSetPriceInvalid()`
  - Replace `System.out.println()` with `assertEquals(expectedDate, testEvent.getEventDate())`
  - Add missing tests: boundary values, null handling, initial state
- Establish unit test standards:
  - Tests must be independent (F.I.R.S.T. principles - course Week 4)
  - One logical assertion per test
  - Descriptive test names: `testSetPriceShouldRejectNegativeValues()`
  - Arrange-Act-Assert pattern
- Require minimum 80% code coverage for new code
- Add unit test review to code review checklist

6. Establish Measurable Performance Requirements and Proper Testing (Priority: HIGH)

Why: Alex's performance test plan (Artifact 7) demonstrates fundamental misunderstandings:

- Confuses test types: calls a Load Test a "Stress Test", calls gradual ramp-up a "Spike Test"
- Subjective acceptance criteria: "site feels fast" - unmeasurable
- **Catastrophic error rate target: "below 10%"** - industry standard is <1%
- No baseline metrics, no specific response time targets
- Test purpose is "good enough to tell investors" - testing for optics, not quality

Acceptance Criteria AC-3 and AC-7 (Artifact 9) are equally vague: "respond quickly", "work most of the time" - untestable.

This violates the principle that requirements must be SMART (Specific, Measurable, Achievable, Relevant, Time-bound) and that non-functional requirements are as critical as functional ones (course Week 7).

How:

- **Before Premium Event launch**: Define measurable performance requirements:
  - Response time: 95th percentile <2 seconds, average <500ms for payment submission
  - Error rate: <1% (not 10%!)
  - Throughput: Support 1,000 concurrent users with above metrics
  - Uptime: 99.9% availability (not "most of the time")
- Establish baseline: Measure current system performance
- Revise Performance Test Plan with:
  - **Load Test**: 1,000 users over 10 min (normal load) - measures if system meets SLAs
  - **Stress Test**: Increase load until system breaks - finds capacity limits
  - **Spike Test**: Sudden jump from 100 to 1,000 users in 30 seconds - tests elasticity
- Define objective pass/fail: Test passes if 95th percentile response time <2s AND error rate <1%
- Purpose: Quality assurance, not investor demos

7. Establish Ethical Guidelines and Professional Standards (Priority: CRITICAL)

Why: Multiple incidents reveal serious ethical violations:

- PM instructs Sara to create a fake security fix - show user a false "security updated" popup without fixing Bug 111 (Artifact 8): **fraud and deception**
- PM decides to hide Bug 111 from bug tracker to "avoid panicking investors" (Artifact 4): **dishonest reporting**
- Alex dismisses serious security issue as "caching blip" without investigation (Artifact 4): **professional negligence**
- Performance testing framed as "good enough to tell investors" (Artifact 7): **misleading stakeholders**

These practices violate:

- ACM Code of Ethics: 1.2 "Avoid harm", 1.3 "Be honest and trustworthy", 1.6 "Respect privacy"
- IEEE Code of Ethics: "Be honest and realistic in stating claims or estimates"
- Software Engineering Code of Ethics: "Act consistently with the public interest"

This creates legal liability, damages user trust, and violates professional responsibility.

How:

- **Immediate action (today)**:
  - Cancel any work on the fake security fix (Artifact 8) - do not implement this
  - Properly log Bug 111 in the tracking system - no hiding bugs from stakeholders
- **Within 1 week**: Establish Team Code of Conduct:
  - Commitment to honest reporting to all stakeholders (users, investors, management)
  - Security issues must be properly investigated and disclosed
  - Testing is for quality assurance, not marketing
  - Users' privacy and data protection is paramount
  - Reference ACM and IEEE professional codes of ethics

- Conduct ethics training for entire team
- Establish anonymous reporting mechanism for ethical concerns
- PM must understand: Hiding issues from investors is breach of fiduciary duty and potentially fraudulent

---

Conclusion

Overall Assessment

Our audit reveals that EventHub is in a **critical quality crisis** caused by systemic failures across all dimensions of software quality assurance:

Testing & Process: No formal test strategy, inadequate test coverage, fundamental misunderstandings of performance testing, and poor unit test quality create a porous defense against defects.

Defect Management: An Excel-based tracking system with inconsistent workflows, missing critical fields, and vague bug reports enables defects to be lost, ignored, or deliberately hidden.

Culture & Roles: A "firefighting" culture prioritizes feature velocity over quality. The absence of dedicated QA creates conflicts of interest. Senior leadership dismisses quality concerns as obstacles to speed rather than enablers of success.

Ethics & Professionalism: Most critically, we identified serious ethical violations including deliberately hiding security vulnerabilities from investors, proposing fake fixes to deceive users, and treating testing as a marketing exercise rather than quality assurance.

The root cause is clear: **Quality has no ownership, no process, and no accountability** at EventHub. Without intervention, the project will continue to accumulate technical debt, lose user trust, and face potential legal liability for data privacy violations.

Expected Impact of Recommendations

If EventHub implements our seven prioritized recommendations, the project will experience transformative improvements:

Immediate Impact (Weeks 1-2):

- Critical security vulnerability (Bug 111) investigated and resolved, protecting user privacy
- Ethical violations stopped - no fake fixes, transparent bug tracking, honest stakeholder reporting
- Proper defect tracking system enables visibility, accountability, and data-driven decisions
- Independent QA role hired, establishing separation of duties

Short-term Impact (Months 1-3):

- Formal test strategy and planning reduce defects escaping to production by 60-80%
- Measurable performance requirements enable objective quality decisions
- Improved unit test quality provides early defect detection at lowest cost
- Team culture shifts from reactive firefighting to proactive quality assurance

Long-term Impact (Months 3-6):

- Investor confidence restored through transparent, honest quality reporting
- User complaints decrease as defect escape rate drops
- PM time freed from firefighting to focus on product strategy
- Development velocity actually *increases* as technical debt decreases and fewer production issues interrupt feature work
- Premium Events feature launches successfully with proper quality gates

Business Case for Quality:

The cost of implementing these recommendations is minimal compared to the cost of the status quo:

- **Current state**: PM spending "most of his time handling user complaints" (Artifact 1), users posting negative reviews on social media (Artifact 8), investor confidence declining, potential legal liability for data breaches
- **Investment required**: One QA engineer salary, migration to free/low-cost bug tracking tool, time investment in test planning (5-10% of development time)
- **ROI**: Studies show defects found in production cost 10-100x more to fix than those found during testing. Every dollar invested in quality saves $10-100 in production firefighting, customer support, and reputation damage.

Final Recommendation

EventHub stands at a crossroads. The current trajectory leads to project failure through accumulating technical debt, user attrition, and investor withdrawal. However, the issues are addressable through disciplined application of fundamental SQA principles.

We strongly recommend implementing all seven recommendations, prioritizing the CRITICAL items (#1, #7) immediately. These ethical and security issues cannot wait. The remaining HIGH priority items (#2-6) should begin within one week to establish sustainable quality practices before the Premium Events launch.

Standard: Quality is not a luxury or a "nice to have"; it is the starting foundation for successful Software. EventHub must embrace this reality to survive and thrive.

End of Report