Finding Bugs Before Your Customers Do

You do exhaustive requirements-based testing, examining both “happy path” (best-case) scenarios, as well as some error conditions and maybe some challenge conditions.

Maybe you supplement this testing with additional path-coverage testing; possibly up to demonstrating 100% path coverage.

Maybe you even get end users to do some testing on a prototype.

Companies typically perform testing to achieve regulatory approval to market and feel confident about their product… until the first bug arises; possibly leading to corrective actions up to recalls.

So why are there still failures in the field?

Issues arise because of the inherent limitations of requirements-focused testing and path coverage. These include:

- Requirements that are incomplete or ambiguous can result in incorrect interpretation during implementation.
- Requirements may be sufficient but implementation can result in a design that is not robust.
- Testers may be biased by company culture or not understand how the system will be used in the field.
- Environmental conditions or data may be different across scenarios, resulting in different outcomes even when the same path is followed.
- System resources are constrained under varying conditions, which leads to different behavior across scenarios.
- End users are often constrained by either not wanting to break the system or by their own bias on how things are used.

No software is ever “bug free,” but at Realtime, we put techniques in place to increase testing effectiveness. The following chart is an estimate of such effectiveness based on experiences at Realtime:

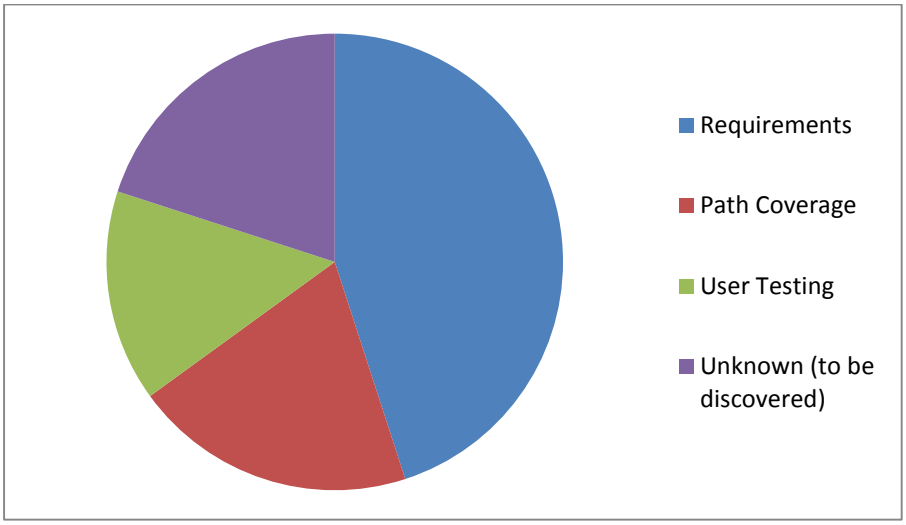

Figure 1 Test Effectiveness

Bugs found in the field are much more costly to fix. Estimates vary, but the cost to fix problems generally follows a curve similar to the one below:

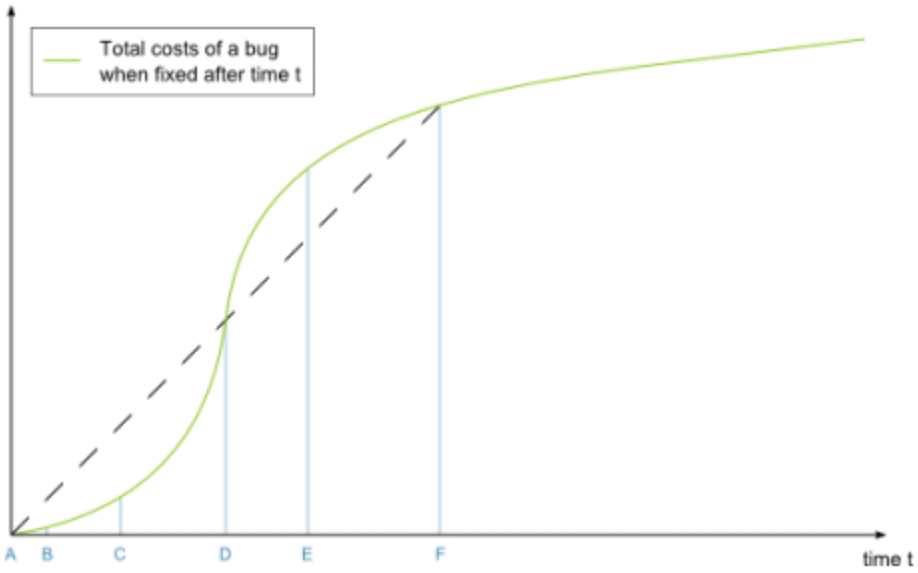

Figure 2 Bug Fix Costs

And this doesn't even factor in the indirect costs associated with damage to a company's reputation.

Consider the following seemingly simple requirement:

- *The user shall be required to enter the time the test started.*

A strictly requirements-based test demonstrates "happy path" compliance, meaning a valid time is accepted. For complete coverage, the test developer should add a test to confirm that not entering a time raises an error condition.

Only testing these conditions, however, still allows for field failures. Other important tests might include checking for:

- 24-hr time (i.e., "military" time) vs. 12-hr time
- hh:mm:ss format vs. hh:mm format only
- Invalid time entries
- Incomplete time entries
- And more.

Also, consider these two requirements:

- *The system shall raise an alarm if the door is opened*
- *The system shall raise an alarm if AC is disconnected*

Pretty common-sounding requirements, right? Traditional testing confirms that each condition is satisfied.

But what happens if the door is opened and then AC is disconnected? Does one condition override the other? Is one condition masked by the other? What if the door is opened, AC is disconnected and then re-connected? Is the door-open alarm lost?

These seemingly simple requirements demonstrate how requirements-based testing alone or even in combination with other testing may still fall short of finding issues.

At Realtime, we have developed several methods beyond "traditional V&V" to help reduce the likelihood that bugs will be revealed in the field.

Structured Exploratory Testing (a.k.a. unscripted) is a method employed at Realtime to efficiently assess the system from a user's perspective without constraining the tester to specific actions or parameters.

Traditional V&V is often structured to follow a single path. Exploratory testing allows the tester to go down one path and then re-visit the scenario if something is observed that may be an issue.

Exploratory ad-hoc testing also enables a risk-based approach by focusing on critical areas. The tester is allowed to explore a scenario without prescriptive actions, enabling data capture. Bounds for pass/fail are defined, typically in terms of safe operations.

The following excerpt from one example scenario demonstrates the effectiveness of Structured Exploratory Testing:

Objective: Assess pump reaction when user tries to modify an infusion to exceed pump limits. **Setup Instructions:** Establish the indicated Setup Configuration. Perform each test procedure for all Setup Configurations (columns B-I unless indicated). Repeat each test using a weight based standard drug, a non-weight based drug, and a basic mL/hr infusion. **Pass Criteria:** A valid infusion cannot be modified to run with invalid settings.	**A -Primary** **B - Primary**	**A -Bolus** **B -Primary**
TEST ID: Test1 **TEST PROCEDURE - CHANNEL A:** While infusions indicated by Setup are running: -Select DOSE soft key. -Press Clear key. -Select VTBI soft key. -Press Clear key. -Select DOSE soft key. -Using auto-increase key, adjust dose to exceed pump limits. **RECORD**: Record drug used, and details about Dose and VTBI at start and finish as indicated on Test Datasheet.	**Record entries in datasheet BD1-A** [ ]Pass [ ] Fail	**Record entries in datasheet BD1-B** [ ]Pass [ ] Fail

In this case, the pump limits were not checked and the pump was allowed to operate at an unsafe delivery rate. Requirements testing alone only confirmed the individual actions met specification. This procedure revealed the interaction of events which resulted in the error.

Multi-Event Testing is another method employed at Realtime that considers how events "piling up" can impact operations. These are often critical in life-sustaining applications, and they reveal unexpected and unaddressed scenarios in most applications. Alarms and events are usually sufficiently tested in isolation during verification testing, but the behavior is rarely defined when events pile up.

The table below shows an example of how Realtime structures testing to consider multiple events.

1) Initial RUNNING condition below	2) Then do action below 5) Then do action in column to the right Legend: x - Not allowed n - Change not accepted y - Change is accepted	Battery Low --> RUN	Battery Very Low --> RUN	Door Open (close) --> RUN	Door Not Latched (correct) --> RUN	PRI ONLY: clear program (no)	clear prgm (primary) (no) --> RUN
Primary Infusion	Chng Delay Run (no OK)	n	n	n	n	x	n/a
	Chng Delay Run (OK)	y	y	y	y	x	n/a
	Chng Time (no OK)	n	n	n	n	n	n/a
	Chng Time (OK)	y	y	y	y	y	n/a
	Chng VTBI (no OK)	n	n	n	n	n	n/a
	Chng VTBI (OK)	y	y	y	y	y	n/a
	Chng Rate (no OK)	n	n	n	n	n	n/a
	Chng Rate (OK)	y	y	y	y	y	n/a
	Chng Rate → advisory (do not ack)	n	n	n	n	x	n/a
	Chng Rate → advisory, cancel (no)	n	n	n	n	n	n/a

Again, the interaction of events, in this case, revealed an error that would normally not have been exposed through requirements-based testing.

Both methods have proven extremely effective in the identification of potential issues.

The following table shows the effectiveness of these approaches on actual testing Realtime has conducted:

Project Description	Issues Revealed	Notes
Infusion pump	• 52 safety issues • 30 observations revealed • 6 discrepancy between documentation and implementation found	Product had been in field for approximately 10 years
Labeling System	First round: • 15 issues (potential (mis)labeling • 27 observations • 2 unreproducible crashes Second round: • 16 issues • 32 observations • 1 unreproducible crash	First round was a "quick, will there be benefit" round; after proving benefit, second round was a deeper dive.
Laser / Ultrasound system	43 issues	Testing on initial baseline.
Infusion pump	130 issues / observations	Testing on initial baseline

Legend

- Issues: Potential safety concerns
- Observations: Unexpected results but not considered safety concerns

Both methods can also be used in support of V&V efforts (approved protocols, expected results, gathering objective evidence, etc.), including to assess maturity as part of a Test Readiness Review.

Structured Exploratory Testing often reveals use (and misuse) cases, and Multi-Case Testing reveals timing or "overlap" conditions that may not have been considered through traditional test methods.

Realtime further optimizes the process by utilizing a risk-based approach to focus more heavily on critical areas (alarms, conditions that could make life-sustaining pumps stop, etc.).

There's no test "silver bullet" (test method) that will reveal all the bugs in the software. The methods explained here, however, have repeatedly demonstrated with a high-degree of confidence that errors found in the field will be minimized.

For more information or to discuss your testing needs with Realtime, contact us by any of the means below:

The Realtime Group
3035 W. 15th Street
Plano, TX 75075
Office: 972-985-9100
dhurd@therealtimegroup.com

Connect with us on LinkedIn: The Realtime Group