Bugs and Bug Reporting

What's in this doc:

1. *What is QA, and what do testers do?*
2. *What is a bug?*
3. *How do you find one?*
4. *How do you report it?*

I. Testing and Quality Assurance

Quality assurance is the process that guarantees a product - software, a game, a hat - meets the standards that someone - the client, the consumer, the producer - expects it to.

Testing is one of the primary tools in the Quality Assurance process, the only way to guarantee that something meets a particular standard is to *test* it against that standard, and check the results to see if they are what was expected. So, the primary job of a QA tester is precisely that: play through the game, comparing what occurs in game vs what is expected to occur (according to design), and reporting as bugs all those places where the game's performance does not match the expectation.

II. What is a bug?

Simply put, a *bug* is a behavior in the software that causes the game to behave in a manner that doesn't meet the expected behaviors or standards. A more technical term for a bug would be a *software defect,* but the colourful parlance of 'bug' is well rooted in the history of computers and engineering in general. For a more detailed description about bugs, go ahead and read the wikipedia article.

III. Finding bugs.

Basic testing: Check to see if it works!

- *The player should not be able to break or crash the game while playing it.*
- *All features should behave as detailed in the game design/ checklist. If they don't, either the game or the checklist should be fixed.*

Even before a tester knows the system specs (specifications) for a game (or a piece of software), there are easily apparent bugs to be found. Any obvious software failures, such as crashes, glitches, hangs; characters that move oddly or don't move at all, buttons that when pressed don't do anything - those are bugs. This is still testing against a standard; it's the just the simple standard any person starting up a program should expect: the player should not be able to break or crash the game while playing it. The program's behavior should not confuse players unless it is intended to.

Once a tester is more familiar with the game,the next standard of testing is against the actual design of the game. Here is where checklists and design docs become important - if a tester does not know how a game is supposed to behave in a given situation, it is impossible to know if particular feature - be it a weapon, a button, or a cutscene - is behaving properly. How does a tester know how high the avatar should be able to jump, or how long a cutscene should run? Read the design docs. Ask the team. Talk to other testers. Read the checklist. And use some common sense - checklists and design docs get out of date. If something doesn't match the 'described behavior', but it seems to work well with the game, and to fit properly, ask. Update the checklist/design doc if necessary to keep the next tester from making the same mistake. If features do not behave as described in documentation, then it's a bug - often in the game, sometimes in the documentation. Both need to be fixed.

Advanced Testing: Sharing your opinion!

- *Features that look good in a design doc sometimes don't play well.*
- *User interfaces that look like they work... sometimes don't.*
- *Sometimes, if there was just one more thing the player could do....*

Often when a tester is playing a game, everything can run fine, work according to spec... and still not be right. Testers spend a lot of time playing the game - often, more than the designers. Good feedback about the gameplay, the mechanics, and the user interface are important from the QA testers.

Especially the user interface. If you don't like clicking through eight levels of menu to accomplish something in game, the casual user is really going to LOATHE it. If the game controls are very persnickty, and it bothers you... it will truly infuriate your user. And that's something the designers of the game won't see, because they won't play the game for hours - and that's where that sort of repetition goes from annoying to put-the-game-down irritating.

However, this is where many testers get into trouble. As a tester, if you are offering an opinion about the features in the game, be it positive or negative, you need to be careful to be ***critiquing*** rather than ***criticizing***. What's the difference?

A **criticism** is an unsupported opinion, sometimes delivered in an unprofessional tone. "*Hey, the second cutscene sucks - it's too long and nothing happens in it*." The game might even need that criticism - no one wants a lousy cutscene in the game. But the right way to deliver that feedback is politely, with an explanation of why it's not good, how long is too long, and what you mean by 'nothing happens in it.'

A **critique** is a statement of opinion about the game, backed up by facts, examples, and a dispassionate explanation. For example... "*The BFG doesn't feel powerful enough. It takes it 2 full clips of ammo to take out Big Boss Number one, while my MiniGun takes out the same boss in 1 clip. But the BFG is built up in the fiction as being the biggest, baddest gun around*." Alternatively, *"I didn't like placement or the story of the second cutscene. I was having a great time playing the game, zooming around, and then suddenly, control was taken out of my hands*

and I had to sit back for over a minute, watching two people talk about things that didn't feel at all relevant to what I was doing." Those are critiques: they state a problem, or even just an opinion, and give support - facts, evidence, examples - as to why it's a problem.

Good games need critique from testers - in a very real way, the tester is the first player to play it, and their opinions are going to reflect other players' - ie, the audience's - initial reactions. If the testers don't like the game, there's something wrong with it, and the team needs to hear that - politely.

Expert Testing: Feature Suggestions

At some point, everyone who plays a game thinks like a game designer, and has an idea about what to add to make it better, testers included. This is, however, expert testing. Wait until you have a good feel for the game, how it is supposed to play, and what kind of an aesthetic & game dynamic the game designers are going for. Be prepared to have nine out of ten of your suggestions shot down, because they are too hard, there isn't enough time, or 'we tried that already.'

For the best possible reception, write your feature ideas as critiques, worded politely, and provide supporting evidence. Explain what problem you're trying to solve with this suggestion, and then offer your idea ***as a suggestion.*** Often, the designers will agree with you on the problem, but come up with an alternate way to solve it. That's their job, and their privilege.

Bug Feedback: You are a professional, and should be treated as such.

Game development is often not just a 'job'; most developers see themselves in the work they do, and therefore in the quality of the game. Sometimes this makes them oversensitive to critiques and bug reports. However, just as a tester is responsible for giving professional reports, developers are also responsible for accepting your feedback in a professional manner.

IV. Reporting Bugs

When you report a bug, the goal is to include in the bug report everything the developer will need to go in, locate the bug, reproduce it, and then fix it. In the real world, many bugs aren't that easy to nail down, but as a tester, you still need to try.

So, for every bug, you should follow roughly the same formula, and gather the following information for your bug report.

1. **Check for Duplicates** - has someone already reported this bug? Open the database, do a key word search/ title scan to see if this has already been reported. If so, stop now, unless you have new information to add to the report *that will help the developers fix it.*
2. **Steps to Reproduce:** Explain exactly how to reproduce the bug, and how reliable the reproduction is. (IE, if you follow these steps, it will happen every time; or, it will have 2 out of 3 times, or, I did it once but then couldn't get it to happen again after 10 times.

    a. Take the time to nail down a good set of repeatable steps. Sometimes the Steps to Reproduce are quick and easy, and you just need to do one pass to confirm that's how it works; sometimes it's a lot harder. *Always do the confirmation step.*
    b. It never hurts to ask another tester to reproduce your bug on their machine before filing the report, just to confirm that your Steps can be followed.
3. **Compatibility Issues:** Is there a hardware or software component to the bug? For browser games, which browser were you using? Does it happen in all browsers? Is it OS specific? Machine specific?
4. **Frequency:** How common is the bug? (This is related to repeatability!) Does it happen every 2 seconds, or under a blue moon of carefully constructed sequences?
5. **Severity:** How serious is the bug? IE - when it occurs, how badly does it disturb the player's game? Does it crash the system, or does it mean that the player has to go pick up a different red jewel to go on?
6. **Description:**
    a. What is the precise behaviour of the bug - ie, what happens in game?
    b. What is the *expected behavior* that should have happened?
7. **Supporting Data:** Would taking a screenshot of the bug make it much easier to understand? If so, do so, and attach it to the report.
    a. Is there error data, from a popup window, error box, crashdump log? If so, include it.
    b. If it's a serious bug - a crash, a hang, a feature malfunction that breaks playability is there a developer around you can talk to immediately? Can you try to reproduce the bug for them running the debugger?

Looking over the list, it's a lot of information - and time - for just one bug. However, the more you do it, the faster you will get. Also, the more bugs you report, and the more testing you do, the better you'll get a feel for how much verification any one bug needs before it gets reported. Use your common sense - if there's a misspelling in one of the menus, you don't need to spend a lot of time reproducing, verifying, and checking to see if it happens on all browsers or not. One screenshot, along with an explanation of where it appears, is enough. Crashes and hangs, on the other hand, usually need a second pass to check on the repeatability, and often need some research to figure out their precise Steps to Reproduce.

You've found a bug. You've got a pile of data to support it. Now what do you do with it? Put it in the database, and assign it to the team contact, of course.

VI. Life of a Bug

From Open To Close

You find a bug, you report a bug, it's the developer's problem now. Until, of course, the bug comes back to you. Why would it do that?

Well, that's the normal life of a bug. Bugs are found, reported in the database, assigned to a fixer, fixed, and then confirmed as fixed in the current build. Testers are responsible for confirming that a bug has been fixed, so all bugs (eventually) return to their initial reporter.

Sometimes bugs return unfixed, because a developer needs more information to fix the bug. In that case, bugs will show up as Status: OPEN, Assigned: TESTER NAME. If you get a bug assigned to you, while still open, it's your responsibility to get the information the developer needs. There should be a comment added to the bug report explaining what they need; ideally, the developer came and talked to you in person as well. Feel free to go back and talk to the developer if you don't understand what's needed by them.

Avoid having database conversations, where you trade the bug back and forth, adding comments like "Here's the repro..." "That didn't work, try again"... "No, really, here's the repro.." etc. Go talk to someone face to face.

When a bug shows up as Resolved: Fixed and assigned to a tester, it's time to verify it. Before you go in and do so, make sure the bug fix is in the build you are testing. The developer should indicate in their Fixed comments as of which build the bug is fixed; if you're not sure, ask.

Review the bug before Closing it. Make sure that if it is hardware or software specific, you are reproducing it on the relevant setup! Verify it is fixed, and mark the bug Closed. If it isn't fixed, update the bug with your new information, why you are declaring it not fixed, and assign it back to the developer who 'fixed' it.

MIT OpenCourseWare
http://ocw.mit.edu

CMS.611J / 6.073 Creating Video Games
Fall 2014

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.