What is a Bug Report?

In the course of the QA process, when a bug has been identified, it has to be documented and sent to developers to be fixed. Given that software is exceptionally complex, layered, and feature-heavy in the current digital environment, most QA pipelines generate multiple bugs.

Additionally, developers often work on multiple development projects simultaneously, which means they have a considerable number of bugs requiring attention. They have to operate under significant pressure and can be overwhelmed without the right resources.

Naturally, QAs spend considerable time researching how to report a bug in a way that benefits developers and helps them debug with speed and efficacy.

Well-structured and adequately detailed bug reports are one of those resources. Good bug reports tell developers exactly what needs to be fixed and help them get it done faster. This prevents software releases from being delayed, offering faster time-to-market without compromising on quality.

Elements of an Effective Bug Report

When studying how to create a bug report, start with the question: What does a bug report need to tell the developer?

A bug report should be able to answer the following questions:

- What is the problem?
- How can the developer reproduce the problem (to see it for themselves)?

- Where in the software (which webpage or feature) has the problem appeared?
- What is the environment (browser, device, OS) in which the problem has occurred?

Want to do a quick bug check on your website across browsers & devices? **Try now.**

How to write an Effective Bug Report

An effective bug report should contain the following:

1. Title/Bug ID
2. Environment
3. Steps to reproduce a Bug
4. Expected Result
5. Actual Result
6. Visual Proof (screenshots, videos, text) of Bug
7. Severity/Priority

1. Title/Bug ID

The title should provide a quick description of the bug. For example, "*Distorted Text in FAQ section on <name> homepage*".

Assigning an ID to the bug also helps to make identification easier.

2. Environment

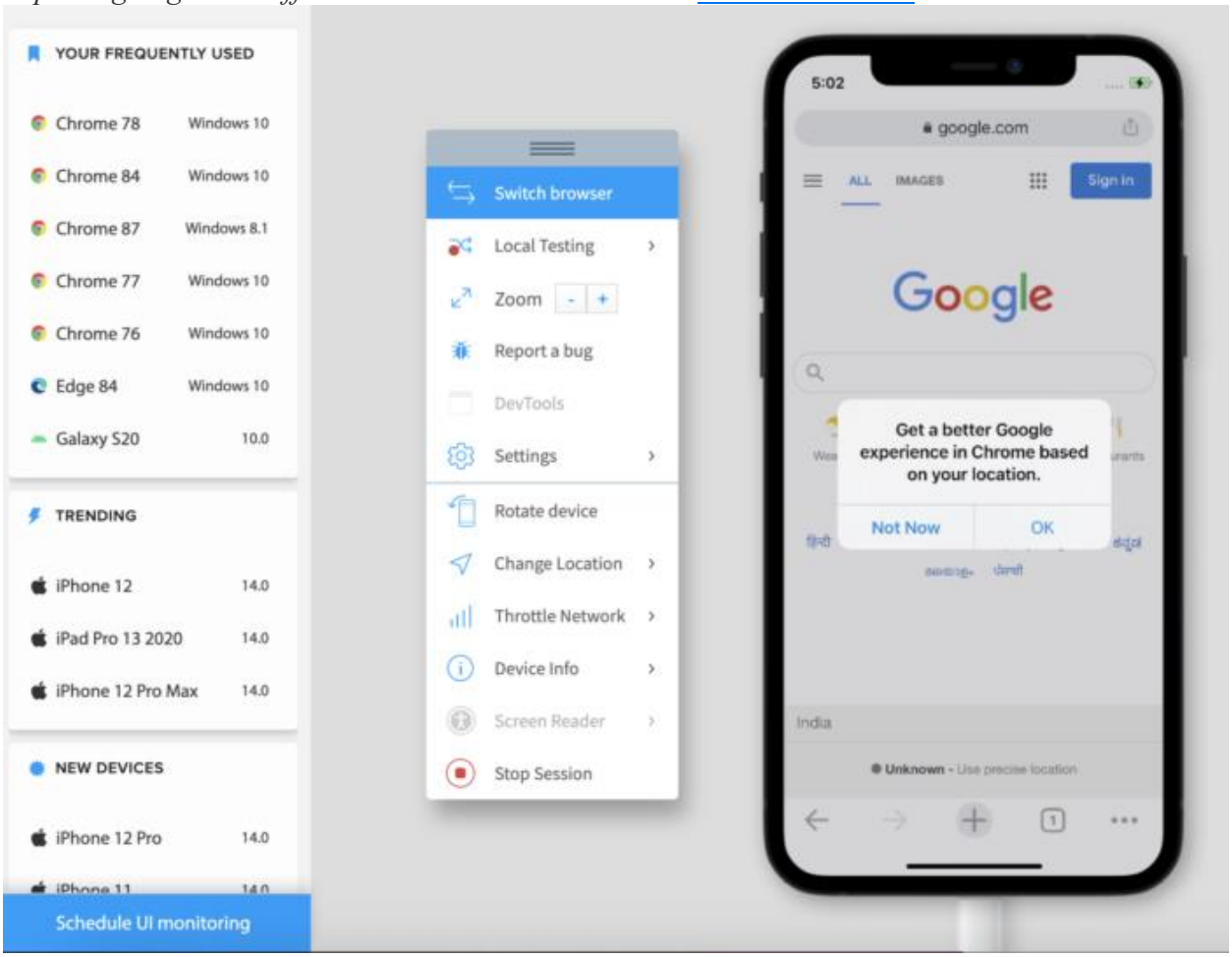
A bug can appear in a particular environment and not others. For example, a bug appears when running the website on Firefox, or an app malfunctions only when running on an iPhone X. These bugs can only be identified with cross browser testing or cross device tests.

When reporting the bug, QAs must specify if the bug is observed in one or more specific environments. Use the template below for specificity:

- **Device Type**: Hardware and specific device model
- **OS**: OS name and version
- **Tester**: Name of the tester who identified the bug
- **Software version**: The version of the software which is being tested, and in which the bug has appeared.
- **Connection Strength**: If the bug is dependent on the internet connection (4G, 3G, WiFi, Ethernet) mention its strength at the time of testing.

- **Rate of Reproduction**: The number of times the bug has been reproduced, with the exact steps involved in each reproduction.

Reporting bugs is an effortless task on BrowserStack's [real device cloud](#).



[Run Tests across Environments for Free](#)

3. Steps to Reproduce a Bug

Number the steps clearly from first to last so that the developers can quickly and exactly follow them to see the bug for themselves. Here is an example on how one can reproduce a bug in steps-

1. Click "Get Started" on the Homepage (this takes the user to the Purchase page).
2. Click on "Pricing Options".

Find out everything about [Bug Tracking](#) before starting the QA process.

4. Expected Result

How is the software supposed to work, with regard to the particular area in which the bug appears? The developer needs to know what the requirement is, in order to gauge the extent to which the bug is disrupting the user experience.
Describe the ideal end-user scenario, and try to offer as much detail as possible. Don't just leave it at "the app is crashing, and it shouldn't."

5. Actual Result

Detail what the bug is actually doing, and how it is a distortion of the expected result.

- Elaborate on the issue
- Is the software crashing?
- Is it simply pausing in action?
- Does an error appear?
- Or is it simply unresponsive?

Specificity in this section will be most helpful to developers. Emphasize distinctly on what is going wrong. Provide additional details so that they can start investigating the issue with all variables in mind. For example:

- "Link does not lead to the expected page. It shows a 404 error."
- "When clicked, the button does not do anything at all."
- "The main image on the homepage is distorted on the iPhone X."

Do you know: 5 Common Bugs Faced in UI Testing

---

6. Visual Proof of Bug

Screenshots, videos of log files must be attached to clearly depict the occurrence of the bug. Depending on the nature of the bug, the developer may need video, text, and images.
Testing using BrowserStack can leverage multiple debugging options – text logs, visual logs (screenshots), video logs, console logs, and network logs. These make it easy for QAs and devs to detect exactly where the error has occurred, study the corresponding code and implement fixes.
Find out: How to rectify visual bugs?
BrowserStack's debugging toolkit makes it possible to easily verify, debug and fix different aspects of software quality, from UI functionality and usability to performance and network consumption.

The range of debugging tools offered by BrowserStack's mobile app and web testing products are as follows:

- Live: Pre-installed developer tools on all remote desktop browsers and Chrome developer tools on real mobile devices (exclusive on BrowserStack
- Automate: Screenshots, Video Recording, Video-Log Sync, Text Logs, Network Logs, Selenium Logs, Console Logs
- App Live: Real-time Device Logs from Logcat or Console
- App Automate: Screenshots, Video Recording, Video-Log Sync, Text Logs, Network Logs, Appium Logs, Device Logs, App Profiling

7. Bug Severity

Every bug must be assigned a level of severity and corresponding priority. This reveals the extent to which the bug affects the system, and in turn, how quickly it needs to be fixed.
Levels of Bug Severity:

- Low: Bug won't result in any noticeable breakdown of the system
- Minor: Results in some unexpected or undesired behavior, but not enough to disrupt system function
- Major: Bug capable of collapsing large parts of the system
- Critical: Bug capable of triggering complete system shutdown

Levels of Bug Priority:

- Low: Bug can be fixed at a later date. Other, more serious bugs take priority
- Medium: Bug can be fixed in the normal course of development and testing.
- High: Bug must be resolved at the earliest as it affects the system adversely and renders it unusable until it is resolved.

Ensure that QA teams have sufficient knowledge of bug severity and bug priority before they start assigning levels in a bug report.

Needless to say, it is not possible to detect every possible bug without running tests on real devices. Additionally, one has to know how frequently a bug occurs and the extent of its impact on the software. Testers cannot report bugs they have not caught during tests.
The best way to detect all bugs is to run software through real devices and browsers. Ensure that software is run through both manual testing and automation testing. Automated Selenium

testing should supplement manual tests so that testers do not miss any bugs in the Quality Assurance process.

In the absence of an in-house device lab, the best option is to opt for a cloud-based testing service that provides real device browsers and operating systems. BrowserStack offers 2000+ real browsers and devices for manual and automated testing. Users can sign up, choose desired device-browser-OS combinations and start testing.

The same applies to apps. BrowserStack also offers real devices for mobile app testing and automated app testing. Simply upload the app to the required device-OS combination and check to see how it functions in the real world.

Here are the 6 things you need to include in an effective bug ticket:

1. **Title** — a searchable and unique title provides the developer a general understanding of the bug before clicking into it.
   For example: Application crash on 'Sign Up' button click

2. **Description** — a brief description about the issue to provide the developer with some context of what the issue is and what they should be looking for.
   For example: When a user tries to sign up for an account, then the application crashes and the user is unable to create an account.

3. **Steps to reproduce** — explain the steps the developer will take to replicate the bug. This is important because a developer can't fix a bug they can't reproduce.
   For example: Go to registration page > fill in required fields > click on 'Sign Up' CTA > application crashes

4. **Expected result** — describe what you expect to happen when interacting with the feature.

For example: Given a user is on the registration page, when they've filled in the required fields and click on the 'Sign Up' CTA, then the user has successfully signed up for an account.

5. **Actual result** — describe the error, flaw, or failure that is taking place when interacting with the feature.
   For example: Given a user is on the registration page, when they've filled in the required fields and click on the 'Sign Up' CTA, then the application crashes and the user is unable to create an account.

6. **A Visual** — providing a visual helps the developer understand what the exact issue is without any investigation on their end. This can significantly increase the speed of fixing the bug.
   For example: a screenshot, video, or any visual.

7. **Priority** — unfortunately bugs can be found often and are not equally as important to fix. Identifying the priority will ensure that developers are addressing the work that is most important first.
   For example: blocker, major, high, medium, low, etc.