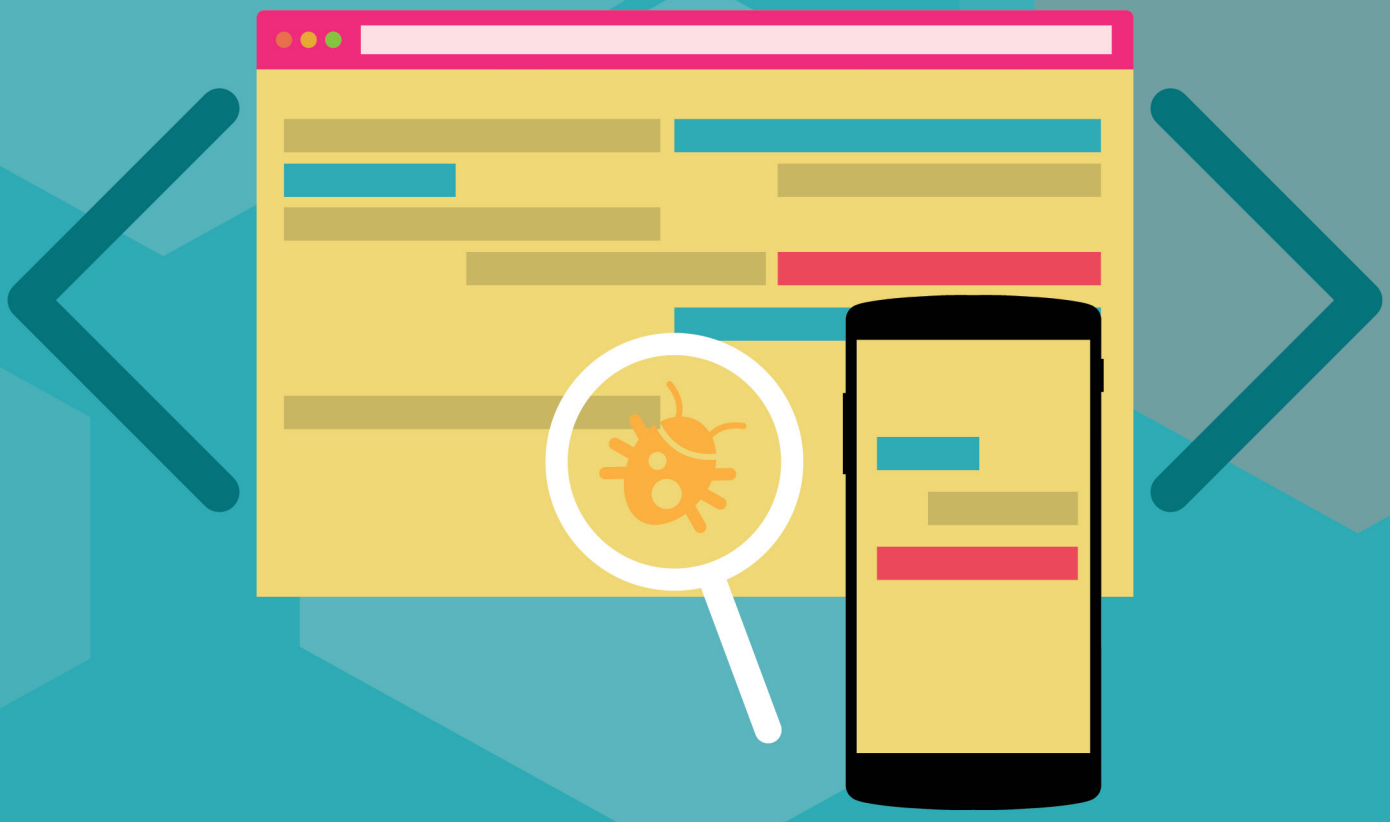
clearbridge mobile

The Ultimate Guide to Mobile App Testing



ABOUT US

Founded in 2011, Clearbridge Mobile has emerged as a leading studio developing state of the art mobile applications.

At Clearbridge we go beyond checklists and simple requirements; we strive for the best product. We get to know our clients (their users and needs) and we push the limits of technology and design to achieve an unparalleled connected experience.

Our apps have received more than 55 million downloads to date, and our client list includes world-leading brands like Disney, ABC, Tim Hortons, PayPal, Shell, USA TODAY, The New York Times and more.

TABLE OF CONTENTS

1 WHY IS MOBILE QA IMPORTANT?

The mobile market has increased exponentially in the last decade, in every way imaginable. From hardware, to practical applications, to mobile web usage and mobile app downloads, the speed and scope of this growth has been staggering.

- Mobile internet usage has increased by 67% in the past year alone (StatCounter)
- 86% of time on iOS and Android devices is spent on mobile apps (Flurry)
- By 2017, the global app market is expected to reach $77 billion (Entrepreneur)

More and more people are choosing to use their smartphones, tablets, and other mobile devices to browse the web, download and use mobile apps, make purchases, and more. There is high consumer demand for mobile, and the market is becoming more saturated as everybody seeks to capitalize.

The result of this demand and saturation is that it is essential that your mobile app offer an absolutely perfect user experience. And to achieve this, your mobile QA needs to be excellent.
This is easier said than done. The sheer number of devices your app needs to function on is high, and constantly growing. OS compatibility is another issue

altogether, as your app may function differently depending on the version. Functionality, UI, and UX all need to be seamless. On top of that, you can't automate everything, so at least some degree of manual testing is necessary. Consequently, mobile testing can be both difficult and time consuming.

The purpose of this guide is to help you improve your mobile QA in order to make your process more efficient and thorough. It will cover a number of integral components involved in a strong mobile testing strategy, including issue reporting, build machines, automated testing, application and web server updates, testing on multiple devices and operating systems, and implementing an agile testing process.

2 ISSUE REPORTING

Issue reporting is a foundational aspect of quality assurance, whether mobile or otherwise. The better your issue reporting, the more smoothly your project will run. Improving this component of your process is not just advantageous to your QA team and developers, but to every person working on a project, including product owners, designers, and project managers.

WHY ISSUE REPORT QUALITY IS IMPORTANT

Regardless of project planning and preparation, bugs find their way into the code. This is an inevitability, and it's why QA exists in the first place.

Fixing these bugs can be a very difficult process for a number of reasons, including the volume of the issues, their complexity, and more. But the pain of correcting such issues can be reduced by improving the quality of issue reports.

Logging proper issues has a huge impact on both the efficiency and quality of a team's work, and ultimately on the product being developed.

THE CORE BENEFITS OF IMPROVING ISSUE REPORTS

The better your issue reports are, the more effectively every member of your team can do their job. Product owners can better triage issues; project managers can quickly and properly assign issues; developers can more effectively resolve issues as they are more understandable; and quality assurance can better verify issues.

High quality issue reports also offer a number of ancillary benefits, including:

- Fewer duplicate tickets reported, less developing, and less testing
- Reducing cycles because issues will be resolved the first time
- Better communication between developers and quality assurance teams
- Insight and control better maintained by POs and PMs
- Fewer surprises, which means a smoother project

The majority of issues will be logged by your quality assurance team, but issue reporting quality is important for everybody involved in the project. Product managers, product owners, and especially developers need to be able to identify and write proper issue reports and understand their benefits. If key components are missing, the ticket should be sent back with a request for the reporter to provide the missing details.

HOW TO IMPROVE ISSUE REPORT QUALITY

1. **Be Clear and Concise**
   Cut out the fluff. Don't write lengthy prose and don't include unrelated information. Clearly and concisely describe the issue.

2. **One Issue, One Ticket**
   Each issue report should have one issue listed on one ticket. Issues need to be individually triaged, assigned, resolved, and verified – and because of that, they need to be individually tracked.

3. **Include the Most Important Components**
   In order for issues to be tracked correctly, issue reports must include steps to reproduce, actual results, and expected results. We'll go over each in turn.

KEY COMPONENTS OF AN ISSUE REPORT

Steps to Reproduce

Without the properly laid out steps to reproduce, the issue is not worth logging. To be fixable, people need to be able to understand how the issue is found. Issues like "sometimes the app is slow", "loading takes too long", or "crashes often" don't have any value. Repetitive steps can be condensed as the project progresses.

A
1. *Launch app*
2. *Enter valid log in credential*
3. *Press log in button*

B
1. *Launch app*
2. *Log in*
3. *Navigate to the Settings page*
4. *Look at the Change Password button*

C
1. *Launch app*
2. *Log in*
3. *Disconnect from the internet*
4. *Press the Refresh button*

Actual Results

This is a short section where you describe the issue itself. In most cases, the results section should be accompanied by crash logs, screenshots, and/or videos.

A. App crashes. Refer to attached crash log.
B. Text on the button is larger than the button. Refer to attached screenshot.
C. Loading wheel never stops turning. Refer to the attached video.

Expected Results

Including the expected results may be the most important component of a proper issue ticket, yet it is often overlooked in the issue logging process. The purpose of expected results is to lay out how the issue could be fixed. If these are not indicated, the issue may be resolved but in a different way than desired, which in itself can result in more issues.

Expected results are used by each team member:

- **Product Owners** make sure the issues are valid and modify the tickets as necessary
- **Project Managers** assign the issues to the developers who can best implement the fix
- **Developers** completely understand their goals before starting to resolve the issues
- **Quality Assurance** verifies that the steps to reproduce now lead to the expected results and the issues have been properly resolved

    A. App should log in without crashing
    B. Text should fit within the button
    C. Network Disconnected error message should appear

Below is a chart (*Table 2.1*) comparing poorly logged issues to properly logged issues. The first issue does not have enough information to be valuable, the second has too much information including some that should be split into multiple tickets, and the third has all the right details but is written in a way that is difficult to understand. Note the formatting of the issue reports – laying them out correctly doesn't take much time, yet makes a big difference.

Poorly logged issues	Properly logged issues
app crashes all the time	Steps to Reproduce: **1.** Launch app **2.** Enter valid log in credentials **3.** Press log in button Actual Results: App crashes. Refer to attached crash log. Expected Results: App should log in without crashing

Table 2.1

Poorly logged issues	Properly logged issues
I was using the app when i was asked to look at it from a design perspective and I was on the log in screen and I found a bunch of problems with the buttons on the page. The text is too big on a button and the button is too small anyway. The color of the background isn't right. I think the background should be green. Green really matches the feeling I get when i use the app and it reminds me of my childhood. Oh speaking of the good old days, there was this one time back in elementary school...	Steps to Reproduce: 1. Launch app 2. Log in 3. Navigate to the Settings page 4. Look at the Change Password button Actual Results: Text on the button is larger than the button. Refer to attached screenshot. Expected Results: Text should fit within the button
I started the app then I logged in then I disconnected from the internet then i tried to reload the page and the loading wheel started spinning but it wouldn't stop but really the wheel should only be spinning when the app is connected to the internet but show an error message instead.	Steps to Reproduce: 1. Launch app 2. Log in 3. Disconnect from the internet 4. Press the Refresh button Actual Results: Loading wheel never stops turning. Refer to attached video. Expected Results: Network Disconnected error message should appear

Table 2.1 (continued)

3 BUILD MACHINES

Every build that quality assurance tests should be built from the same computer. When your product is ultimately ready for distribution, the build machine is the machine that will be used to build the release version of the mobile app. In this context, we will be referring to a build machine as both a physical computer and the automated scripts used to build and distribute the app to the testers.

WHY USE A BUILD MACHINE?

A build machine allows your team to work more effectively and more efficiently. It enables you to automate the more menial tasks so that you can focus on unique challenges, edge cases, and other issues that can't be discovered via automation.

Core Benefits

Every developer should be running the exact same OS and IDE, down to the version. It is the developer's responsibility to make sure the work they do will work with the team, not just on their own computer. If members are using different environments with different versions, the project may be affected in unexpected

ways. “But it worked on my machine” is never a valid excuse.

In the same way, each member of quality assurance should be testing the same version of the application. “But it worked when I tested it” just doesn't cut it. The purpose of a build machine is to ensure this consistency. A proper build machine will manage the app build number and make it universal for the team instead of individually managed. This significantly improves the issue tracking and resolution process.

Other benefits of using a build machine include more accurate testing, fewer mistakes, easier issue tracking and resolution, and allowing testing to start immediately rather than waiting for the builds.

At Your Command

You can set your build machine up to perform its tasks as frequently as you need. Below are a few examples of when you can have your build machine run:

- Upon request
- At a scheduled hour every day
- Automatically when certain repository events are triggered:
  - When any commit is made to a certain branch
  - When any branch is merged into a certain branch
  - When merge conflicts are resolved on a certain branch

Beyond controlling when your build machine runs, you can also control what it does: build, email, upload, perform any git operation, etc. Commands are strung together in scripts to create your build machine – writing your own script allows you to have your build machine complete any task you tell it to.

Example Build Machine

The build machine we've created performs the below tasks for every project before the office opens every work day:

1. Resets the local repository, discarding any changes
2. Pulls any changes to the branch
3. Updates the build number
4. Compiles and exports the build so it can be installed
5. Signs the build so it can be distributed
6. Uploads the build to distribution service
7. Emails testers announcing that the build has been made available
8. Tags the commit with the build number
9. Backs up the build and output logs for archiving purposes

If any of the above steps fail, the build machine has been set-up to send an email containing the output logs along with an explanation of the issue and how to proceed.

This is just one example of a build machine. You can set yours up to perform the kinds of tasks that are required for your specific app build to ensure consistency and efficiency.

4 AUTOMATED TESTING

Automated testing is useful to QA teams because it has the potential to significantly reduce the labour resources required to run regression tests or other tedious, repetitive tasks in the testing process. Rather than having a QA team member test manually, the process is automated.

The automated testing process can be very simple or capable of running virtually infinite test cases. Navigating between each section of your program, clicking/tapping every visible button possible, filling in every text field on the screen – all of this and more is possible with automation.

And thanks to a number of libraries that can be downloaded and used right from the internet, starting out with automated testing isn't particularly complex. All you need to do is download the library file, add it in your automation project suite, and start coding test cases.

When Should You Use Automated Testing?

Despite the advantages of automated testing, it isn't right for every project. If the design and flow of your project keep undergoing changes, for example, then your

automation will need to be adjusted accordingly. If there are many changes, it might not be worth spending time writing and adjusting automation test cases, but rather just have your QA team handle the tests instead.

Furthermore, automated testing on mobile can only go so far. There are limits for VQA, as well as discovering and testing for edge cases. Thus, there is typically some degree of manual testing needed.

So how do we determine when to use automated testing? As is true of most good QA practices, it's important to layout what will need to be tested even before the development process starts. Additionally, a clear and concise understanding of the project flow should also be established. If the flow isn't concrete, or design is subject to change, writing automation tests may not be worth it as you will likely spend more time on coding than you would actually testing.

On the other hand, if a project requires tedious and repetitive testing, such as filling out forms, navigating between different sections, logging into different accounts, and whatever else QA find themselves doing endlessly during each testing session, then having an automation suite run all those tests is probably worth it. It will save time and resources, letting QA focus their time on other sections of the project or on other projects altogether.

Ultimately, it comes down to planning. If design and flow are in a good state and there will be many repetitive tests that need to be run during each iteration of the project, automated testing is probably your best bet.

5 TESTING ON ANDROID DEVICES

With the proliferation of many different types of mobile devices – from various smartphones, to tablets, to phablets – one significant challenge faced by developers is ensuring that their applications function correctly across all of them. To achieve this goal, app developers and their QA teams need to account for the many shapes, sizes and restrictions that emerge when dealing with multiple devices.

Due to Apple's markedly closed system, their entire library of devices share very similar specifications. No other developers can modify the platform – only Apple devices run iOS. This allows developers to expect that their application will function properly without necessarily having to test on each Apple device.

With Google, the situation is much different. It has opted to allow manufacturers to create their own hardware that often runs a customized version of the Android platform. This open system creates a broader spectrum of devices that vary in screen resolution and size, memory and processing limits, and even app availability. Consequently, there are a variety of unique, device-specific issues that testers have to be cognizant of.

Below are a list of devices and some of the more significant testing challenges they present, as well as some tips on how to avoid potential QA issues.

NABI

Nabi devices are child-friendly tablets that operate on one of the customized versions of the Android platform mentioned above. One of the major benefits of Nabi from a consumer perspective – apart from the large rubber borders that protect it from physical damage – is the inclusion of two separate modes: Parent and Nabi. The latter is the mode that is used for children. They have access to apps, but cannot modify settings or add applications themselves. Parent mode allows full functionality and enables parents to approve the apps that their children have access to.

These two distinctive modes can present development challenges. For example, having too long of a package name will cause Nabi mode to ignore your app when grabbing all approved apps, even if you do approve it. Thus, when testing, one of the most important things to check is whether the app can appear in Nabi mode, especially considering it is the biggest selling point of the device.

KINDLE

Kindle devices have evolved from simple e-book readers to fully functional tablets. They are becoming an increasingly popular choice, and developers would be wise to ensure their applications work on the Kindle platform.

The hardware itself contains only the minimum amount of physical buttons (power and volume up/down) – everything else is placed into a digital navigation bar. The bar can be hidden, but will be revealed with a swipe. This navigation bar is typically the root of many QA and dev issues. The app requirements may dictate that when the bar is visible, the entire UI of the app shifts up to accommodate the extra space that is being taken away. In that case, the UI shifts may cause unwanted

visual and (potentially) functional issues. Consequently, when testing an app on a Kindle tablet, it is very important to test all UI features both with and without the navigation bar.

OTHER DEVICES RUNNING ANDROID

The majority of other Android devices follow general conventions like physical navigation buttons or app distribution via Google Play. However, their hardware differences can affect the functionality of applications.

Variations in screen size, resolution, CPU and memory limits become the biggest challenge when testing on some of the more "common" devices, including Samsung, Nexus, Asus, Acer, etc. All of these devices will have different specs, which unfortunately can result in a number of device-specific problems. For example, some UI elements may appear correctly (quality and position) on Device A, but will appear improperly on Device B.

Having a list of supported devices will certainly help with testing, ensuring that any issues raised only apply to those listed devices. Another thing to keep in mind is that devices may be running different OS versions. Not all users upgrade their operating system immediately, and different OS versions may handle certain functions, layouts, etc. differently. Keeping a small variety of OS versions on the same type of device will give you the most coverage possible.

Testing Android devices is a much more involved process than iOS devices, due to the large variety of hardware being created and the manufacturer-specific Android setups being put on those devices. Learning the hardware-specific quirks of your testing devices will allow you to create a more stable, polished app through thorough testing.

6 TESTING/UPDATING APP AND WEB SERVERS

When you are providing a live service to active users, it is dangerous to update production without a safe process. Having a well laid out process that is followed by your team will help make sure the users are not disturbed by the update.

Figure 6.1 and *Table 6.1* depict the process we use for web sites and servers.

Server Promotion Process (Git Flow)

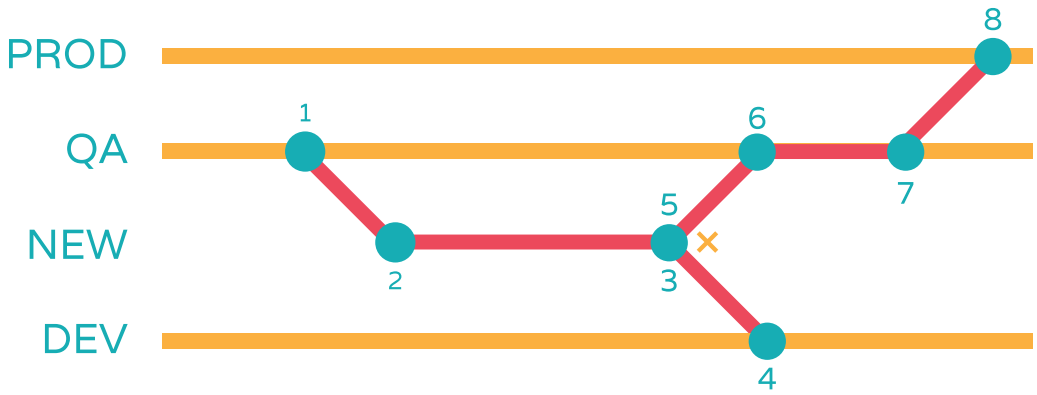


Figure 6.1

Server Promotion Process (Task Flow)

1 To Do	2 In Progress	3 Resolved	4 Ready To Test In Dev
Developer is assigned a story. Developer creates a new feature branch from QA branch Note: Feature branch is never created from Dev branch Developer moves ticket to "In Progress"	Developer works on story on feature branch When complete, Developer publishes feature branch and pushes ticket into "Resolved", indicating the branch name in ticket	Lead merges each feature branch with ticket into Dev environment and pushes the story to "Ready to test in Dev"	QA tests story on Dev environment If fails, story is re-opened and assigned back to developer If passes, story moves to "Ready for push to QA"

Table 6.1

5 Ready For Push To Qa	6 Ready To Test In QA	7 Ready For Push To Production	8 In Production
Lead merges each feature branch with ticket into QA environment Lead pushes the story to "Ready to test in QA" Lead deletes the feature branch **Note: No push is made from Feature branches to QA branch**	Product / QA tests story on Dev environment If fails, story is re-opened and assigned back to developer. If passes, story moves to "Ready for push to Production" QA performs regression testing on QA environment.	Lead merges entire QA branch into Prod environment and pushes every story to "In Production" **Note: No push is made from Feature branches to Production branch**	Feature is live.

Table 6.1 (continued)

Advantages of this process:

- Developers don't need to worry about merging their code - they work on their feature and the lead takes care of the merge
- Only features that have passed testing get pushed into QA branch
- QA branch serves as the future production environment for accurate testing
- Pushing to production is simple and free of merge conflicts

We use a similar process for application updates (*Figure 6.2* and *Table 6.2*):

App Promotion Process (Git Flow)

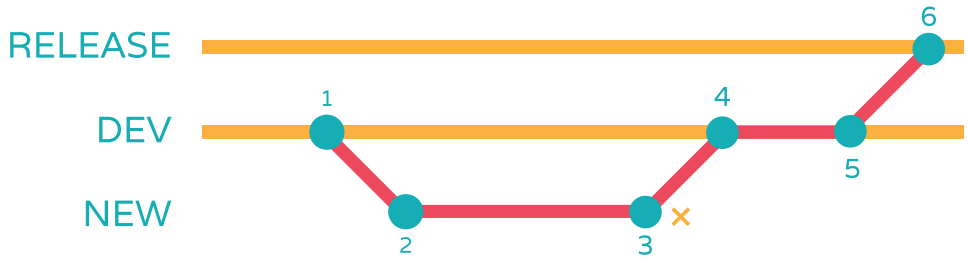


Figure 6.2

App Promotion Process (Task Flow)

1. To Do	2. In Progress	3. Resolved
Developer is assigned a story Developer creates a new feature branch from Dev Developer moves ticket to “In Progress”	Developer works on story on feature branch When complete, Developer publishes feature branch and pushes ticket into “Resolved”, indicating the branch name in ticket	Lead merges each feature branch with ticket into Dev environment and pushes the story to “Ready to test in Dev”

Table 6.2

4. Ready To Test	5. Ready For Release	6. Released
QA tests story on Dev environment If fails, story is re-opened and assigned back to developer If passes, story moves to “Ready for Release”	Lead merges entire Dev branch into Release and pushes every story to “Released” Note: No push is made from Feature branches to Release branch QA builds and releases the app	Feature is live.

Table 6.2 (continued)

7 LINEAR VS. AGILE QA

Everybody's been there. You're working on an important project with a stringent deadline. You've hit a few snags along the way, but it looks like you'll deliver on time. Just one week left to devote to finding and fixing bugs.

But then things start going badly. All mistakes, assumptions, rushed hacks, and misunderstandings surface simultaneously. The open issue count is higher than the number of stories across the project. Everyone is working nights and weekends, and even still, you miss your deadline.

If this happens every time a project meets its time to be tested, your QA process is likely the bottleneck.

THE LINEAR QA PROCESS

The plan

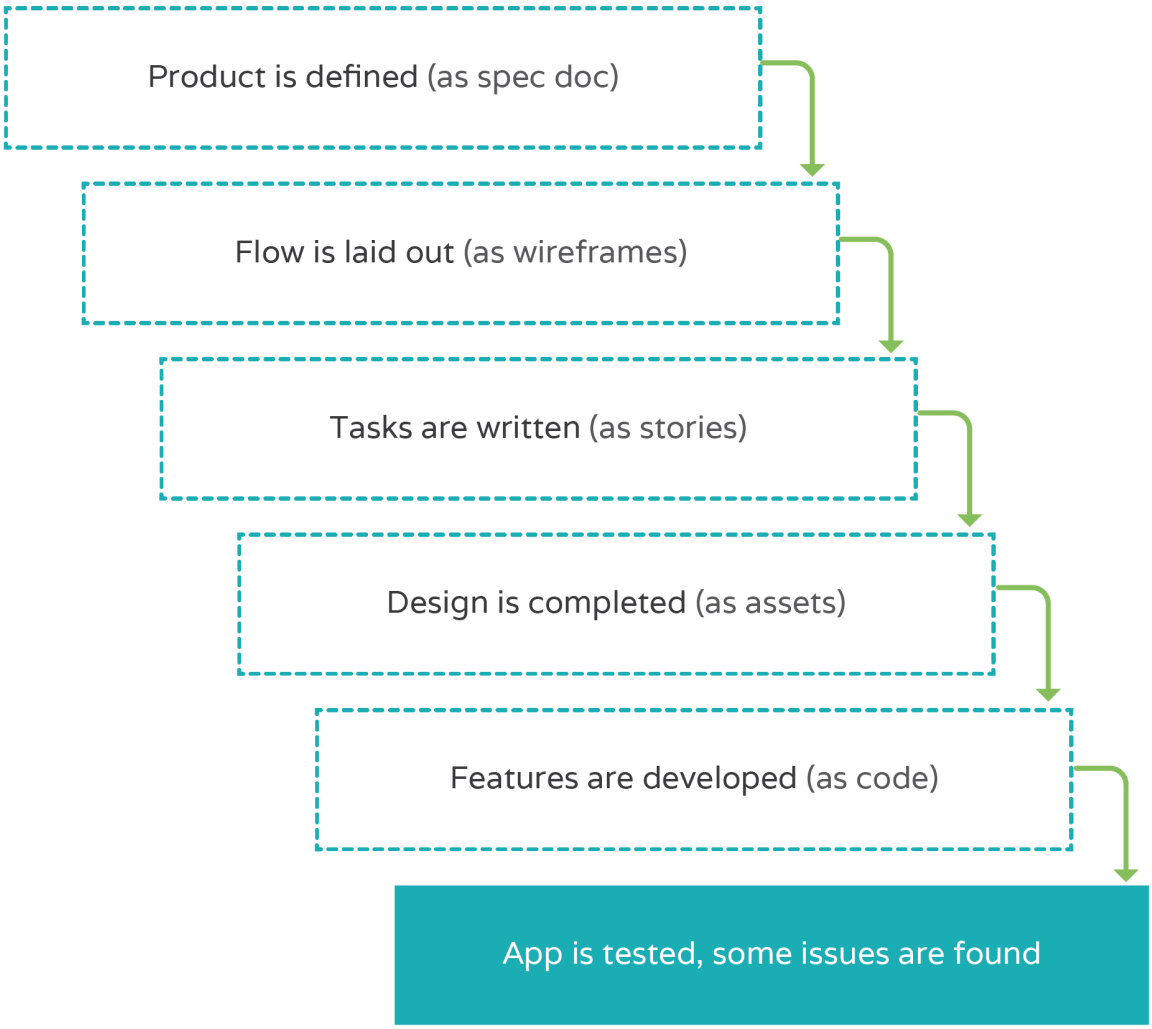


Figure 7.1

The reality

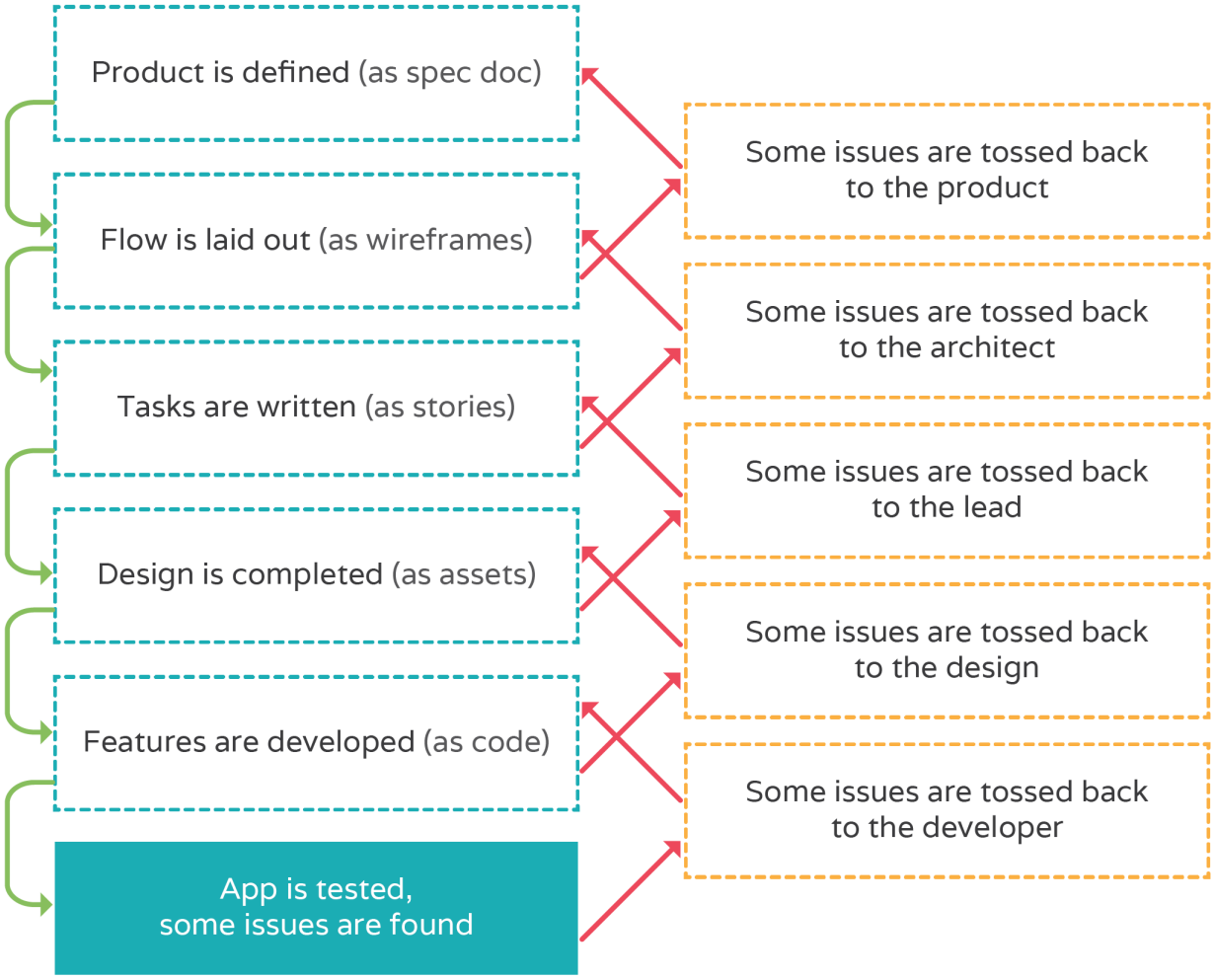


Figure 7.2

One of the issues with the linear process is that when a problem is found, it is often assumed to be the fault of the developer. However, any of the largest, most impactful mistakes are made at a different point in the process, whether from product, design or elsewhere. For example:

Product issues

- Failure to define ways to log-out or change password
- No delete account/stop payment function
- OS incompatibility issues

Task Issues

- Incorrect requirements appear in task
- Some requirements are missing from the task
- Requirements that should be split appear in one task

Flow Issues

- No way for user to refresh a page
- No way to back out of settings page
- User is expected to use physical back button even though the app will be available on iPhone

Design Issues

- UX conflicts with native UX
- Font size is too small for device
- Icon is difficult to see against background color

All of these are basic mistakes which should not happen, but it is most often not a question of the people – every member on your team will make mistakes. Their work is difficult, repetitive, and takes time. On top of that, their work is always rushed since the rest of the team can't even start working until they have completed much of their work for the project.

Testing was born out of the realization that there will be mistakes in the development process. The integrated QA process takes this lesson and applies it to the entire project, from top to bottom.

THE INTEGRATED QA PROCESS

The difference between the linear and integrated QA process is that the latter takes into account the fact that ALL project members – not just developers – will make mistakes. The QA team tests the project continuously, not just when it is a completed app. The major advantage is that a product can be viewed at any point in its life cycle and issues can be recognized immediately. Things like product specs, flow, tasks and designs are looked at before the developer starts work. Thus, if there are issues, they can be corrected before development even begins.

The integrated QA process creates a completely different project flow:

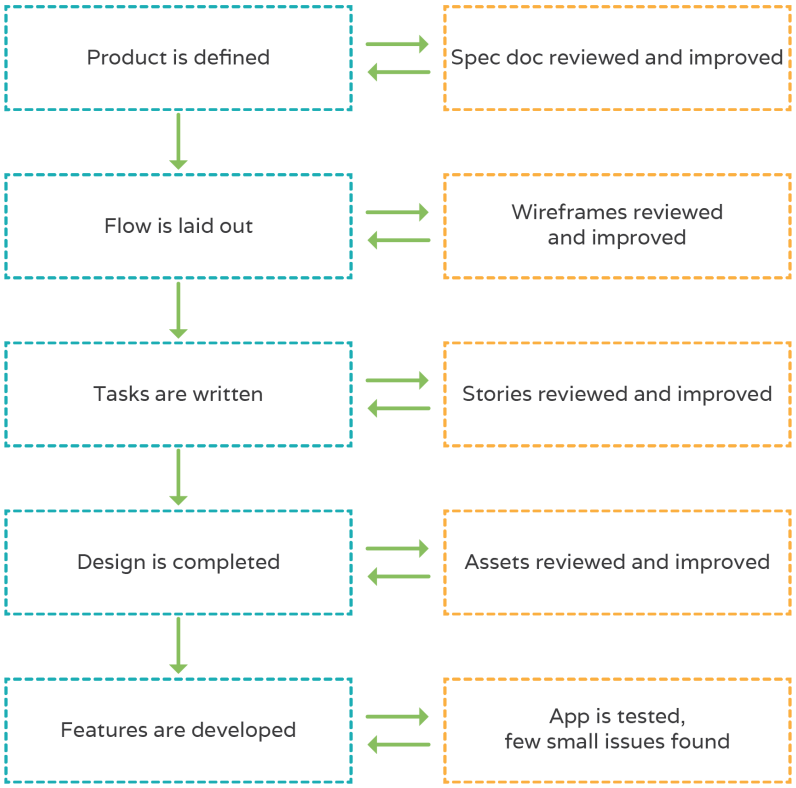


Figure 7.3

Despite initial appearances, the integrated QA process saves a lot of time and frustration when practiced. The following highlights its advantages over the linear process, using an issue created at the product stage as an example.

Linear QA Process

A small oversight is made at the beginning...

1. Product accidentally forgets the settings features while writing the spec doc
2. Architect lays out the flow based on the spec doc in the the wireframes without the settings screen
3. Project Manager writes tasks based on the wireframes as stories, missing all relating to the settings screen
4. Designer creates the assets, mockups, and style guides based on the stories and wireframes, missing everything relating to the settings screen
5. Developer follows the stories using the assets, and develops the app to a usable state without the settings screen

The Testing Sprint Begins Weeks Later...

6. QA tests the app, noticing there is no settings screen and the user can not log out, change their password, or opt out of email notifications

The issue works its way back up the chain...

7. QA logs an issue "No settings button" and it gets assigned to the developer
8. Developer comments on the issue "No assets are made for the settings screen" and it is assigned to the designer

9. Designer comments on the issue "No story is written about settings buttons" and it is assigned to the project manager
10. Project manager comments on the issue "No settings screen was present in the wireframes" and assigns it to the architect
11. Architect comments on the issue "No settings feature was defined in the spec doc" and assigns it to the product manager

On its way back down...

12. Product corrects the spec doc and assigns the issue to the Architect
13. Architect corrects the wireframes and assigns the issue to the Project Manager
14. Project Manager writes new stories and assigns the issue to design
15. Designer creates all the new assets, mockups, and style guides necessary and assigns the issue to the developer
16. Developer follows the new stories using the new assets, and resolves the issue
17. QA tests the settings screen, and realizes that the it has no change password, log out, or opt out of email notifications features

Result: Deadline Missed.

Agile QA Process

A small oversight is made at the beginning...

1. Product accidentally forgets the settings features while writing the spec doc
2. QA reviews the spec doc and points out the missing settings features
3. Product corrects the spec doc before the Architect completes the wireframes

Result: Issue was found and resolved in the first week of the project

If you look at this case in the linear QA process, it is an absolute nightmare. And this is from one small oversight, which happened to affect the entire team. Imagine what could happen if multiple mistakes were made?

On the other hand, in this same case the agile integrated QA process addressed the minor oversight in the first week. It was corrected before the spec doc even got to the architect, and the project continued as planned.

The integrated QA process offers a number of additional benefits as well, which include:

- Better coordination and communication between QA and other teams, which can help improve quality and speed of work
- Every team member understands their tasks in the context of the entire project
- More coordinated, smoothly-running project

By going agile and choosing to adopt the integrated QA process, teams can significantly improve the quality of their project, while ensuring that deadlines are met. Integrated QA simply provides added benefits that linear QA cannot.

8 CONCLUSION

An efficient QA process doesn't just benefit your QA team. The efficiencies and quality gained are advantageous for everybody working on the project, from designers to developers, product owners to project managers. Additionally, stepping up your QA helps your team deliver a better product to market – given the competitiveness of the mobile app market, releasing a good product is essential if you hope to be successful.

Improving your mobile testing strategy centers on a few key areas, which include better issue reporting, using build machines, employing automated testing when appropriate, updating servers correctly, and adopting an agile QA process over the waterfall approach. If you address issues in your QA, you will find that your team will be more efficient and deliver better products.

GET IN TOUCH

PHONE
647 · 361 · 8401

WRITE
sales@clearbridgemobile.com

WEBSITE
www.clearbridgemobile.com

clearbridge mobile