

How to debug mobile apps across devices / Mobile-App-Debugging auf verschiedenen Geräten

Dokumentation von Quelltext und Übersetzung

Herausgeber	TestMu AI
Originaltitel	How to debug mobile apps across devices
Übersetzungstitel	Mobile-App-Debugging auf verschiedenen Geräten
Quelle	MDN Web Docs Blog – https://developer.mozilla.org/en-US/blog/debug-mobile-apps-across-devices/
Lizenzangabe	CC BY-SA 2.5

Zusammenfassung

Der Artikel erklärt, warum Mobile-App-Debugging auf mehreren Geräten für eine stabile User Experience wichtig ist. Er beschreibt typische Ursachen wie Gerätefragmentierung, unterschiedliche Betriebssystemversionen, herstellerspezifische UI-Anpassungen und Hardware-Optimierungen. Anschließend stellt er Debugging-Techniken wie Trace-Logs, Breakpoints, Variablenanalyse, Stack Traces, Exception Handling und externe Debugger vor und nennt Tools wie Android Studio, Xcode und TestMu AI.

Summary

The article explains why debugging mobile apps across multiple devices is important for a stable user experience. It covers device fragmentation, operating-system variation, vendor UI changes, and hardware optimization as possible sources of bugs. It then presents debugging techniques such as trace logs, breakpoints, variable analysis, stack traces, exception handling, and external debuggers, and names tools including Android Studio, Xcode, and TestMu AI.

Herausforderungen bei der Übersetzung

- Fachterminologie und Präzision: Begriffe wie „debugging“, „breakpoints“, „stack trace“, „exception handling“, „root cause“ und „real device cloud“ mussten fachlich korrekt und im deutschen Technikregister verständlich bleiben.
- UI/UX- und Dokumentationsklarheit: Der Text erklärt Verfahren, Ursachen und Handlungsabfolgen; die Übersetzung musste diese Logik klar führen, damit sie als Anleitung für Testing, Handbuch- und Dokumentationskontexte funktioniert.
- Plattform- und Gerätekontext: Unterschiede zwischen Betriebssystemen, Hersteller-UIs und Hardware-Optimierungen mussten nachvollziehbar übertragen werden, ohne die technischen Zusammenhänge zu vereinfachen oder zu verfälschen.

Translation challenges

- Technical terminology and precision: Terms such as “debugging,” “breakpoints,” “stack trace,” “exception handling,” “root cause,” and “real device cloud” had to remain technically accurate and understandable in a German technical register.
- UI/UX and documentation clarity: The text explains procedures, causes, and sequences of action; the translation had to preserve this logic clearly so it works in testing, manual, and documentation contexts.
- Platform and device context: Differences between operating systems, vendor UIs, and hardware optimization had to be rendered coherently without simplifying or distorting the technical relationships.

How to debug mobile apps across devices

Bugs in mobile apps negatively impact user experience, often affecting visitors' first impressions and increasing uninstall rates. When users encounter bugs, they frequently choose alternative apps, leading to lost customers and revenue.

Therefore, businesses should focus on developing robust and bug-free apps as part of a strategy to retain customers. Debugging across the many types of devices visitors use is challenging but essential for maintaining long-term user experience, making mobile app testing a critical component of the process.

What is mobile app debugging?

Mobile app debugging is the process of finding and fixing bugs, ideally before releasing them to users. Bugs are often attributed to failures in functional processes. However, that is frequently not the case.

For example, excessive response time for an API can be classified as a bug, but it doesn't relate to the core functionality of any modules on a mobile application. Along the same lines, debugging may not always mean fixing the functional aspects of a mobile app, but rather fixing any application issues that have a user impact or even implementing fixes that prevent problems from occurring down the line.

Typically, developers use debugging techniques to see how features and fixes behave in real-time as they're being developed. Testers use debugging to explore vulnerabilities and how an app handles usage that may be unexpected or unforeseen in the development phase, often leveraging a real device cloud to ensure thorough testing across various environments.

Why debug mobile apps on multiple devices?

As stated above, the steps of the debugging process may not be the end of the mobile app debugging cycle. An additional parameter to consider in the debugging cycle is the existence of thousands of types of mobile devices.

For instance, in the snapshot below, consider just the screen size as a differentiator in these three devices and observe how the UI design should change accordingly:

Mobile-App-Debugging auf verschiedenen Geräten

Bugs in mobilen Apps beeinträchtigen die User Experience – der erste Eindruck der Besucher:innen ist entsprechend schlecht und treibt so die Deinstallationsrate in die Höhe. Wenn User auf Bugs stoßen, entscheiden sie sich zudem häufig für alternative Apps. Das Ergebnis? Nachhaltiger Verlust von Kundschaft und Umsatz!

Daher sollten Unternehmen die Entwicklung robuster und fehlerfreier Apps zum integralen Teil einer Kundenbindungsstrategie machen. Die User nutzen eine Vielzahl von unterschiedlichen Endgerätetypen. Das macht das Debugging zu einer Herausforderung, ist aber unerlässlich, um die User Experience langfristig auf dem notwendigen Niveau zu halten. Das macht Mobile-App-Testing zu einem entscheidenden Bestandteil des Prozesses.

Was ist Mobile-App-Debugging?

Mobile-App-Debugging ist der Prozess, der idealerweise Bugs aufspürt und behebt, bevor diese die User erreichen. Bugs werden gerne als Zeichen von Fehlern in funktionalen Prozessen wahrgenommen. Das ist jedoch häufig nicht der Fall.

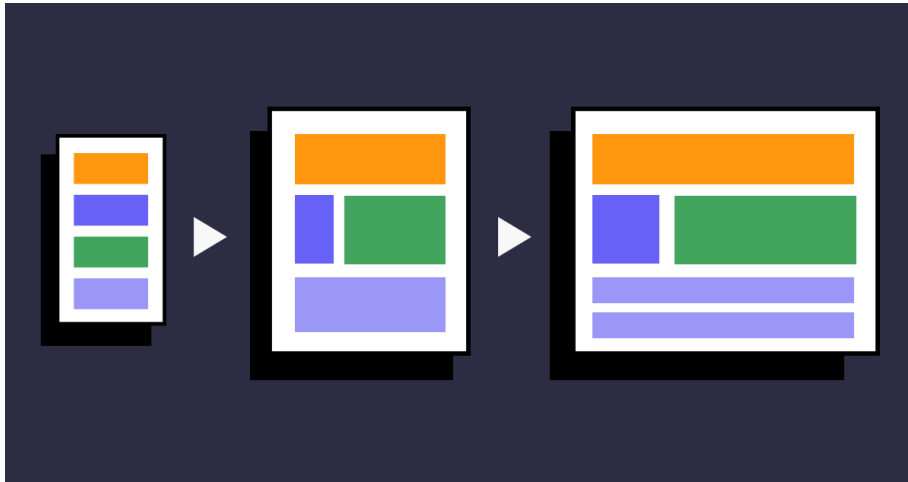
Beispielsweise lässt sich eine übermäßige Antwortzeit einer API als Bug einstufen, aber sie hat nichts mit der Kernfunktionalität eines Moduls in einer mobilen Anwendung zu tun. Ebenso bedeutet Debugging nicht immer, die Funktion einer mobilen App sicherzustellen. Vielmehr geht es darum, alle Anwendungsprobleme zu adressieren, die sich auf User Experience auswirken, oder sogar Fixes zu implementieren, die verhindern, dass im weiteren Verlauf Probleme auftreten.

Typischerweise verwenden Entwickler:innen Debugging-Techniken, um in Echtzeit zu sehen, wie sich Funktionen und Fixes während der Entwicklung verhalten. Tester:innen nutzen Debugging, um Schwachstellen zu untersuchen und herauszufinden, wie eine App mit unerwarteter oder in der Entwicklungsphase nicht vorhergesehener Nutzung umgeht. Dabei greifen sie oft auf eine Real Device Cloud zurück, um umfassendes Testing in verschiedenen Umgebungen sicherzustellen.

Darum sollten Sie auf ein geräteübergreifendes Mobile-App-Debugging setzen

Wie oben erwähnt, sind die Schritte des Debugging-Prozesses möglicherweise nicht das Ende des Debugging-Zyklus für mobile Apps. Ein weiterer Parameter, den es im Debugging-Zyklus zu berücksichtigen gilt, ist die Existenz der vielen tausend Arten von Mobilgeräten.

Betrachten Sie zum Beispiel in der folgenden Abbildung nur die Bildschirmgröße als Unterscheidungsmerkmal bei diesen drei Geräten und beobachten Sie, wie sich das UI-Design entsprechend ändern sollte:



Screen size is only one area where a mobile device can differ. If you consider other device characteristics, debugging on multiple devices early in the development lifecycle becomes increasingly important. Let's look at some ways that devices may be unique:

Extremely high device fragmentation:

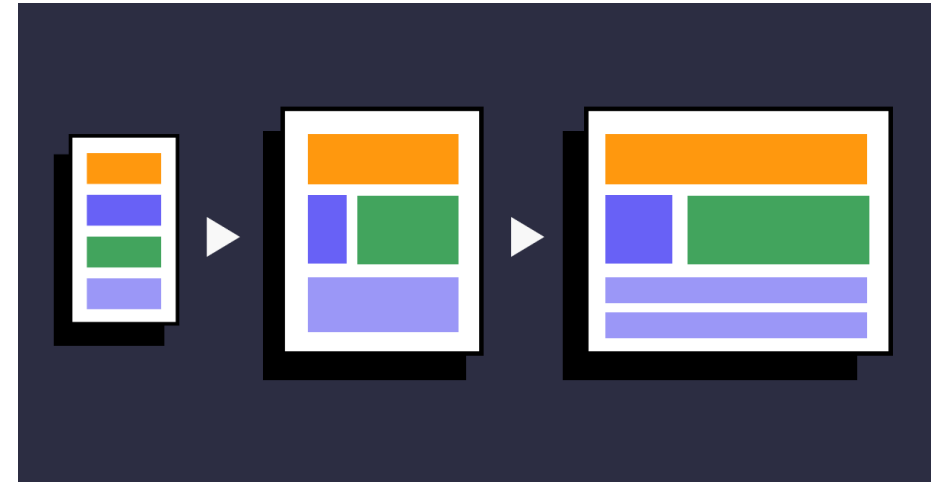
Devices today come with unique specifications, and each of them affects the presentation of an application. Some may have different screen sizes, while others work on different processing powers. This makes the compilation, execution, and operation of the application vary on various devices.

Variation in operating systems:

The StatCounter report puts Android and iOS at 99.34% of the world's mobile market share. However, it's more complicated than testing on two OSes, as each operating system ships a new version to end users yearly. Each new OS version includes deprecations, newer APIs, and optimizations in existing methods. These changes affect how applications work and are developed and are a potential cause of bugs.

Vendor UI modifications:

The Android operating system is open-source and welcomes downstream modifications according to mobile manufacturers' requirements. An excellent example of this is Samsung's One UI. The company uses the source code of the newly released Android and makes changes to the UI in-house. This makes



Die Bildschirmgröße ist nur ein Bereich, in dem sich Mobilgeräte unterscheiden können. Wenn Sie andere Geräteeigenschaften berücksichtigen, wird geräteübergreifendes Debugging früh im Entwicklungslebenszyklus zunehmend wichtiger. Sehen wir uns einige Aspekte an, in denen sich Geräte unterscheiden können:

Extrem hohe Fragmentierung des Geräteangebots:

Geräte verfügen heute über einzigartige Spezifikationen, und jede davon wirkt sich auf die Darstellung einer Anwendung aus. Einige haben möglicherweise unterschiedliche Bildschirmgrößen, während andere mit unterschiedlicher Rechenleistung arbeiten. Das führt dazu, dass Kompilierung, Ausführung und Betrieb der Anwendung auf verschiedenen Geräten variieren.

Variation bei Betriebssystemen (OS):

Einem Bericht von StatCounter zufolge halten Android und iOS einen weltweiten Marktanteil von 99,34 % bei mobilen Betriebssystemen. Es geht jedoch nicht nur darum, auf zwei Betriebssystemen zu testen: Jedes Betriebssystem liefert jährlich eine neue Version für End-User aus. Jede neue OS-Version enthält veraltete Funktionen, neue oder überarbeitete APIs sowie Optimierungen an bestehenden Methoden. Diese Änderungen wirken sich darauf aus, wie Anwendungen funktionieren und entwickelt werden, und sind eine potenzielle Ursache für Bugs.

UI-Anpassungen durch Hersteller:

Das Android-Betriebssystem ist Open Source und begrüßt Downstream-Modifikationen entsprechend den Anforderungen der Mobilgerätehersteller. Ein hervorragendes Beispiel dafür ist Samsungs One UI. Das Unternehmen verwendet den Quellcode des neu veröffentlichten Android und nimmt intern

their devices different from those of other manufacturers. Still, a tester should understand that such changes happen because UI elements are designed and analyzed differently in One UI as compared to in 'vanilla' Android. Therefore, vendor-specific changes can cause bugs in an application's UI, making it an essential action item in the testing and debugging cycle.

Considerable system changes:

Many manufacturers have changed the general design of Android (including core kernel functionality) and incorporated that into their devices. An example of this modification is the OxygenOS from OnePlus. This OS modification can affect the functional aspects of a mobile application, making it work differently on certain operating systems compared to Android.

Hardware optimization:

Many device manufacturers hire specialists who optimize hardware based on their software for increased performance. Optimizations include complete SOC design and development, custom battery discharge rates, and managing heat dissipation. All these factors are critical to mobile app behavior. Therefore, a bug on one device may appear (or not appear) on another due to these optimizations.

When the above points are combined, it should become more apparent why each device, no matter how different it is from another, must be placed in a device farm and incorporated into all debugging cycles.

How to debug mobile apps on multiple devices?

Once a bug is found, use one or more techniques and strategies to expose its root cause, tracing the flow from the source to the output (the bug itself).

Debugging techniques and strategies

The following techniques (one or more) are helpful in debugging mobile applications:

Trace log files:

Ideally, log files provide enough information to understand how execution flowed from a user action to a bug. Sometimes, the error is also logged with a description. These are rare scenarios, however.

Most of the time, log files only provide the code flow and specific values of variables at various points in execution.

Änderungen an der UI vor. Das unterscheidet die Geräte des Unternehmens von jenen anderer Hersteller. Tester:innen müssen entsprechend in Erwägung, dass solche Änderungen auftreten, weil UI-Elemente in One UI anders konzipiert und analysiert werden als in Standard-Android-Systemen. Solche herstellerspezifische Änderungen können Bugs in der UI einer Anwendung verursachen, was sie zu einem wesentlichen Thema im Testing- und Debugging-Zyklus macht.

Erhebliche Systemänderungen:

Viele Hersteller haben das allgemeine Design von Android (einschließlich der Kernfunktionalität des Kernels) geändert und dies in ihre Geräte integriert. Ein Beispiel für diese Modifikation ist OxygenOS von OnePlus. Diese OS-Modifikation kann sich auf die funktionalen Aspekte einer mobilen Anwendung auswirken, sodass sie auf bestimmten Betriebssystemen anders funktioniert als auf Android.

Hardware-Optimierung:

Viele Gerätehersteller beschäftigen Expert:innen, um die Hardware auf Basis ihrer Software für eine höhere Leistung optimieren. Zu den Optimierungen gehören das vollständige System-on-a-Chip-Design (SOC) und dessen Entwicklung, benutzerdefinierte Batterieentladeraten und das Management der Wärmeableitung. All diese Faktoren sind entscheidend für das Verhalten mobiler Apps. Daher kann ein Bug auf einem Gerät aufgrund dieser Optimierungen auch auf einem anderen auftreten – oder eben nicht.

Wenn man die oben genannten Punkte kombiniert, sollte deutlicher werden, warum jedes Gerät, egal wie sehr es sich von einem anderen unterscheidet, in eine Gerätefarm aufgenommen und in alle Debugging-Zyklen integriert werden muss.

So debuggen Sie eine Mobile App geräteübergreifend

Sobald ein Bug gefunden wurde, verwenden Sie eine oder mehrere Techniken und Strategien, um die Ursache aufzudecken, indem Sie den Fluss der Daten von der Quelle bis zur Ausgabe (dem Bug selbst) verfolgen.

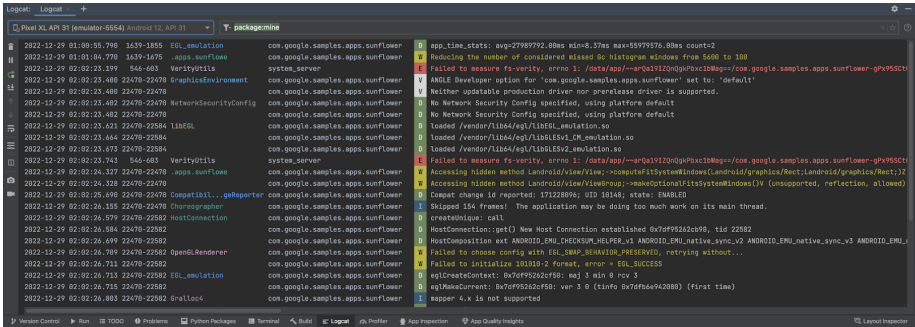
Debugging-Ansätze und -Strategien

Die folgenden Ansätze (eine oder mehrere) sind hilfreich beim Mobile-App-Debugging:

Trace-Logdateien prüfen:

Idealerweise liefern Logdateien genügend Informationen dazu, wie der Ausführungsfluss von einer Nutzeraktion zu einem Bug geführt hat. Manchmal wird der Fehler auch mit einer Beschreibung protokolliert – allerdings eher selten.

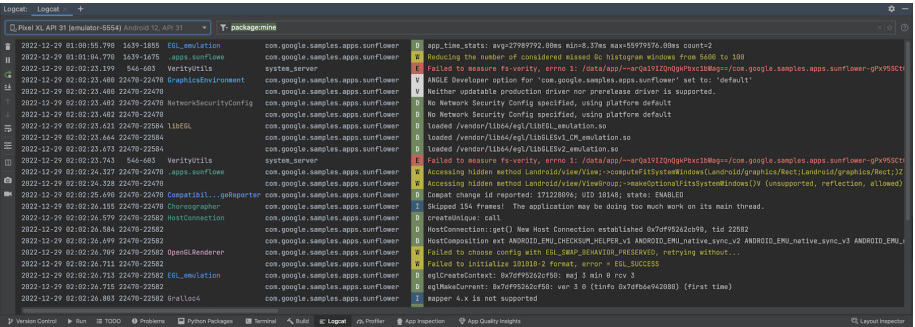
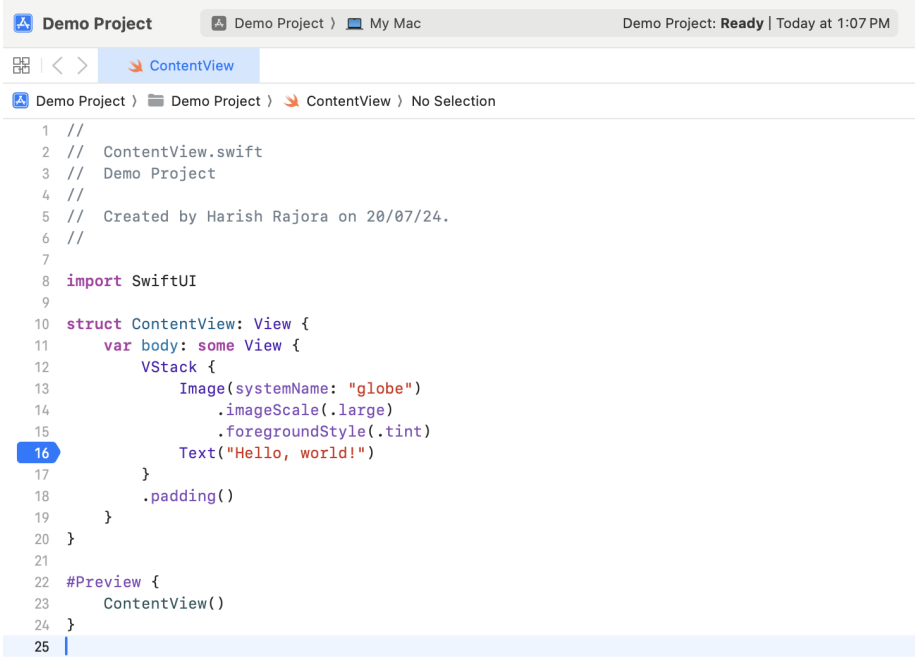
Meistens liefern Logdateien nur den Codefluss und spezifische Werte von Variablen an verschiedenen Punkten der Ausführung.



“Debug“-level logging, which is disabled in production but enabled during debugging, can greatly assist troubleshooting during mobile app development.

Use breakpoints:

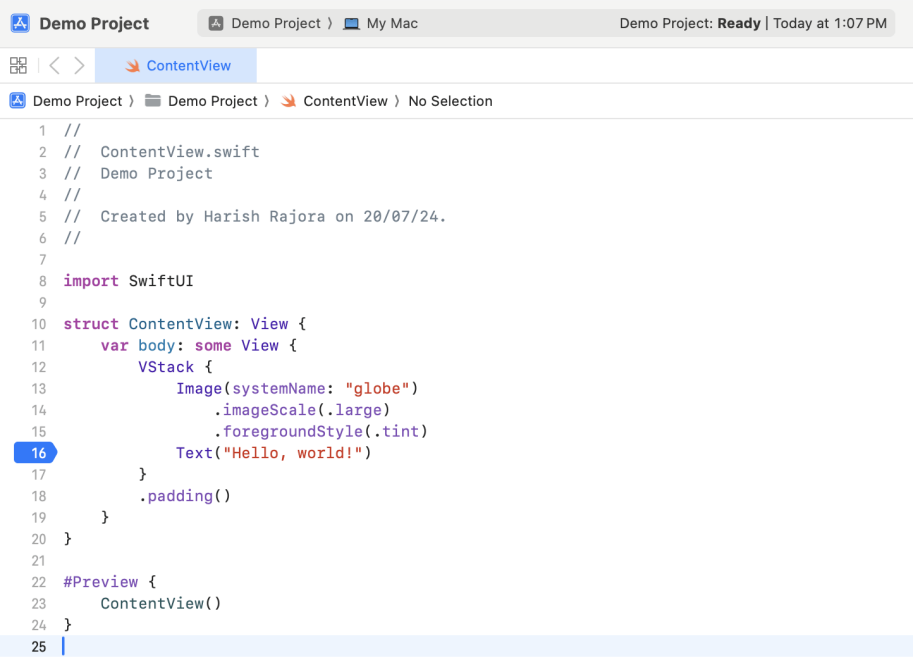
Modern mobile app development IDEs have a breakpoint feature in the source code. Breakpoints work by pausing the program execution at a specific point (called a breakpoint).



Logging auf „Debug“-Ebene, das in der veröffentlichten App deaktiviert, aber beim Debugging aktiviert ist, kann die Fehlerbehebung bei der Entwicklung mobiler Apps erheblich unterstützen.

Breakpoints verwenden:

Moderne IDEs für die Entwicklung mobiler Apps verfügen über eine Breakpoint-Funktion im Quellcode. Breakpoints stoppen die Programmausführung an einem vorgegebenen Punkt (einem Breakpoint).



Once the program pauses, the developer can witness the complete flow from that point onward in single steps (each line of code as it executes) and determine the buggy line of code.

Analyze variable values:

Many problems arise due to incorrect variable values. Analyzing variable values at different points can help track down the specific section that affected their values. Developers can isolate a program section to narrow down and identify the problematic line.

Analyze stack trace:

Stack traces are the method calls with information about these calls printed on the console. These are often found when there is a "Warning" or "Error" during the execution of the program. The developers can trace the root cause with the file name and line number by analyzing these messages and printed errors.

However, this technique may yield fruitful results only sometimes. As mentioned before, all bugs in mobile apps are not necessarily functional. In cases of logical bugs, there will be no stack trace because nothing went wrong functionally to raise the errors; therefore, this method will not be able to provide any results.

Involve exception handling:

Exception handling is a recommended code development practice to ensure the code always exits gracefully. While debugging the applications, developers can look for exceptions inside the log file to understand what went wrong during program execution. The exceptions are primarily specific to the error (such as ValueError) and can decode the root cause quickly if clubbed with logging statements.

External debuggers:

For any popular IDE you are using to develop a mobile application, at least one third-party debugger is available as an extension. In many cases, the debuggers are embedded inside the IDE by default. Such debuggers have functionalities to facilitate the debugging process.

For instance, each function or file provides all the variable values, pictorial representation of functional flow, and memory usage. Using an external debugger that complements the IDE is an excellent technique for mobile app debugging.

Debugging tools for mobile apps

Sobald das Programm pausiert, können Entwickler:innen den kompletten Ablauf von diesem Punkt an in Einzelschritten (jede Codezeile, wie sie ausgeführt wird) verfolgen und die fehlerhafte Codezeile bestimmen.

Variablenwerte analysieren:

Viele Probleme entstehen durch falsche Variablenwerte. Die Analyse von Variablenwerten an verschiedenen Punkten kann helfen, den spezifischen Abschnitt aufzuspüren, der ihre Werte beeinflusst hat. Entwickler:innen können einen Programmabschnitt isolieren, um die problematische Zeile einzugrenzen und zu identifizieren.

Stapelüberwachung (Stack Trace) analysieren:

Stack Traces sind die Methodenaufrufe, die gleichzeitig Informationen über diese Aufrufe auf der Konsole ausgeben – etwa, wenn während der Ausführung des Programms eine „Warnung“ oder ein „Fehler“ auftritt. Die Entwickler:innen können die Ursache (Root Cause) mit dem Dateinamen und der Zeilennummer aufspüren, indem sie diese ausgegebenen Meldungen und Fehler analysieren.

Allerdings liefert dieser Asantz nur manchmal brauchbare Ergebnisse. Wie bereits erwähnt, sind nicht alle Bugs in mobilen Apps zwangsläufig funktional. Bei logischen Fehlern gibt es keine Stapelüberwachung, da funktional nichts schiefgelaufen ist, um die Fehler auszulösen; daher wird diese Methode keine Ergebnisse liefern können.

Ausnahmebehandlung (Exception Handling) einbeziehen:

Exception Handling ist eine empfohlene Praxis bei der Codeentwicklung, um sicherzustellen, dass der Code immer ordnungsgemäß beendet wird. Während des Debuggings der Anwendungen können Entwickler:innen in der Logdatei nach Ausnahmen suchen, um zu verstehen, was während der Programmausführung schiefgelaufen ist. Die Ausnahmen sind primär spezifisch für den Fehler (wie ValueError) und können in Kombination mit Logging-Anweisungen die Ursache schnell entschlüsseln.

Externe Debugger:

Für jede gängige IDE, die Sie zur Entwicklung einer mobilen Anwendung verwenden, ist mindestens ein Debugger eines Drittanbieters als Erweiterung verfügbar. In vielen Fällen sind die Debugger standardmäßig in der IDE integriert. Solche Debugger verfügen über Funktionen, um den Debugging-Prozess zu erleichtern.

Beispielsweise liefert jede Funktion oder Datei alle Variablenwerte, eine grafische Darstellung des funktionalen Ablaufs und der Speichernutzung. Die Verwendung eines externen Debuggers, der die IDE ergänzt, ist eine hervorragende Technik für das Mobile-App-Debugging.

Debugging-Tools für mobile Apps

Developers and testers can leverage the following tools to debug mobile apps across devices:

Android Studio:

Android Studio is an IDE optimized for Android applications, providing a powerful platform for debugging mobile apps. It allows developers and testers to debug apps by plugging them into the system where Android Studio is installed.

It offers features like enabling breakpoints to pause program execution, logging, and variable analysis, and can debug applications during runtime.

Xcode:

Xcode is an IDE with robust debugging capabilities for Apple-based OS. The main features of Xcode include code suggestions and highlighting to avoid bugs based on syntax and programming practices, the ability to set breakpoints for detailed program execution analysis, and more.

TestMu AI:

TestMu AI is an AI-driven testing platform that allows developers and testers to debug mobile apps on a virtual and real device cloud of 10000+ latest and legacy real Android and iOS devices.

It integrates seamlessly with various app distribution platforms wherein you can debug apps by installing them from the App Store, Google Play Store, App Center, and TestFlight. TestMu AI provides a UI Inspector for Android and iOS devices for mobile app debugging, reducing time to identify and resolve root causes. You can test natural gestures and interactions, QR code and image captures, biometric authentication, and more.

Beyond the above tools, developers and testers can enhance their app debugging process by combining test automation and leveraging features like auto-healing, which automatically detects and fixes issues in test scripts, and AI-driven Root Cause Analysis (RCA), which quickly identifies the underlying causes of test failures and suggests fixes, ensuring efficient and reliable debugging and mobile app testing.

Debugging process for mobile apps

Once you select a debugging tool, you can kick-start your mobile app debugging. Here is how to perform debugging for your mobile app:

Entwickler:innen und Tester:innen können die folgenden Tools nutzen, um Mobile-App-Debugging auf verschiedenen Geräten durchzuführen:

Android Studio:

Android Studio ist eine für Android-Anwendungen optimierte IDE, die eine leistungsstarke Plattform für das Mobile-App-Debugging bietet. Sie ermöglicht Entwickler:innen und Tester:innen das Debugging von Apps, indem sie diese an das System anschließen, auf dem Android Studio installiert ist.

Sie bietet Funktionen wie die Aktivierung von Breakpoints zum Anhalten der Programmausführung, Logging und Variablenanalyse und ermöglicht das Debugging von Anwendungen während der Laufzeit.

Xcode:

Xcode ist eine IDE mit robusten Debugging-Fähigkeiten für Apple-basierte Betriebssysteme. Zu den Hauptfunktionen von Xcode gehören Codevorschläge und -hervorhebung zur Vermeidung von Fehlern basierend auf Syntax und Programmierpraktiken, die Möglichkeit, Breakpoints für eine detaillierte Analyse der Programmausführung zu setzen, und mehr.

TestMu AI:

TestMu AI ist eine KI-gesteuerte Testing-Plattform, die es Entwickler:innen und Tester:innen ermöglicht, Mobile-App-Debugging in einer virtuellen sowie einer Real Device Cloud mit über 10.000 aktuellen und älteren echten Android- und iOS-Geräten durchzuführen.

Sie lässt sich nahtlos in verschiedene App-Distributionsplattformen integrieren, in denen Sie Apps debuggen können, indem Sie sie aus dem App Store, Google Play Store, App Center und TestFlight installieren. TestMu AI bietet einen UI Inspector für Android- und iOS-Geräte für das Mobile-App-Debugging, der die Zeit zur Identifizierung und Behebung von Ursachen (Root Causes) reduziert. Sie können natürliche Gesten und Interaktionen, QR-Code- und Bilderfassungen, biometrische Authentifizierung und mehr testen.

Über die oben genannten Tools hinaus können Entwickler:innen und Tester:innen ihren App-Debugging-Prozess verbessern, indem sie Testautomatisierung kombinieren und Funktionen wie die automatische Fehlerbehebung (Auto-Healing) nutzen, die automatisch Probleme in Testskripten erkennt und behebt. Eine KI-gesteuerte Ursachenanalyse (RCA) identifiziert zusätzlich schnell die zugrunde liegenden Ursachen von Testfehlschlägen und schlägt Fixes vor. So ist ein effizientes und zuverlässiges Debugging und Mobile-App-Testing zu gewährleisten.

Debugging-Prozess für mobile Apps

Sobald Sie ein Debugging-Tool ausgewählt haben, können Sie Ihr Mobile-App-Debugging starten. Hier erfahren Sie, wie Sie das Debugging für Ihre mobile App durchführen:

1. Execute functionality: Testers run the app's features following documentation and acceptance tests.
2. Find and document the bugs: Test cases are executed; found bugs are recorded in the bug database.
3. Find the root cause if possible: Testers document scenarios for developers to identify and fix the root cause of bugs.
4. Fix the bug: Developers address and fix bugs by merging updated code into the source codebase.
5. Validate and close: Testers revalidate the fixed code and close the bug if requirements are met.
6. Explore for future bugs: Analyze past bugs and tests to anticipate and prevent future issues.

Summary

Development and debugging go hand in hand in every sprint/release cycle. Accelerated debugging without compromising on quality enables enterprises to ship exceptional products faster and with confidence. By combining test automation with features like auto-healing and automated Root Cause Analysis, developers and testers can ensure efficient testing, leading to faster app development cycles, reduced maintenance efforts, and improved overall app quality.

This is a sponsored article by TestMu AI. TestMu AI (Formerly LambdaTest) is a Full Stack Agentic AI Quality Engineering platform that empowers teams to test intelligently and ship faster. Engineered for scale, it offers end-to-end AI agents to plan, author, execute, and analyze software quality. AI-native by design, the platform enables testing of web, mobile, and enterprise applications at any scale across real devices, real browsers, and custom real-world environments.

1. Funktionen ausführen: Tester:innen führen die Funktionen der App gemäß Dokumentation und Abnahmetests aus.
2. Die Bugs finden und dokumentieren: Die in Testfällen gefundenen Bugs werden in der Bug-Datenbank aufgezeichnet.
3. Die Ursache (Root Cause) finden, falls möglich: Tester:innen dokumentieren Szenarien für Entwickler:innen, um die Ursache von Bugs zu identifizieren und zu beheben.
4. Den Bug beheben: Entwickler:innen bearbeiten und beheben Bugs, indem sie den aktualisierten Code in die Quellcode-Basis mergen.
5. Validieren und schließen: Tester:innen revalidieren den behobenen Code und schließen den Bug, wenn die Anforderungen erfüllt sind.
6. Nach zukünftigen Bugs suchen: Über die Analyse früherer Bugs und Tests lassen sich zukünftige Probleme antizipieren und verhindern.

Zusammenfassung

Entwicklung und Debugging gehen in jedem Sprint-/Release-Zyklus Hand in Hand. Beschleunigtes Debugging ohne Kompromisse bei der Qualität ermöglicht es Unternehmen, schneller außergewöhnliche und stabil laufende Produkte auszuliefern. Die Kombination von Testautomatisierung mit Funktionen wie automatischer Fehlerbehebung und automatisierter Ursachenanalyse ermöglicht Entwickler:innen und Tester:innen ein effizientes Testing – und damit schnellere App-Entwicklungszyklen, reduzierten Wartungsaufwand und verbesserte allgemeine App-Qualität.

Dies ist ein gesponserter Artikel von TestMu AI. TestMu AI (ehemals LambdaTest) ist eine Full-Stack Agentic AI Quality-Engineering-Plattform, die Teams in die Lage versetzt, Apps intelligent zu testen und schneller auszuliefern. Diese skalierbare Plattform bietet durchgängige KI-Agenten zur Planung, Erstellung, Ausführung und Analyse der Softwarequalität. TestMu AI ist von Grund auf KI-nativ und ermöglicht das Testing von Web-, Mobile- und Enterprise-Anwendungen in jeder Größenordnung auf echten Geräten, in echten Browsern und in benutzerdefinierten realen Umgebungen.

Impressum / Imprint

Übersetzung von / Transcreation by

Helmut Barz

Blumenstraße 52

D-63069 Offenbach am Main

helmut@helmut-barz.com

Hinweis / Note

Dieser Text entstand als Arbeitsprobe auf eigene Initiative und ohne Auftraggeber. Weitere Arbeitsproben finden Sie unter: <https://helmut-barz.com/arbeitsproben/>

This text was written as a sample piece on my own initiative and without a client. You can find more about my service offerings (including additional samples) at:
<https://translations.helmut-barz.com>

Rechte / Rights:

Dieses Dokument ist unter der Creative-Commons-Lizenz „Attribution-ShareAlike 2.5“ (CC BY-SA 2.5) lizenziert. Das bedeutet: Sie dürfen das Material honorarfrei vervielfältigen, verbreiten und bearbeiten, solange Sie die Urheberschaft angemessen nennen und Ihre abgewandelten Werke unter denselben Lizenzbedingungen weitergeben.

Vollständiger Lizenztext: <https://creativecommons.org/licenses/by-sa/2.5/>

This document is licensed under the Creative Commons Attribution-ShareAlike 2.5 (CC BY-SA 2.5) License. This means you are free to copy, distribute, and adapt the material, provided you give appropriate credit and distribute any modified works under the same license terms.

Full license text: <https://creativecommons.org/licenses/by-sa/2.5/>