Developing a Mobile Test Concept

Gabriel González López

Developing a mobile test concept can become an extensive task and involve several factors that affect the success of a mobile application that meets business needs and customer expectations and remains a cost-effective project.

Identifying stakeholders and their interactions

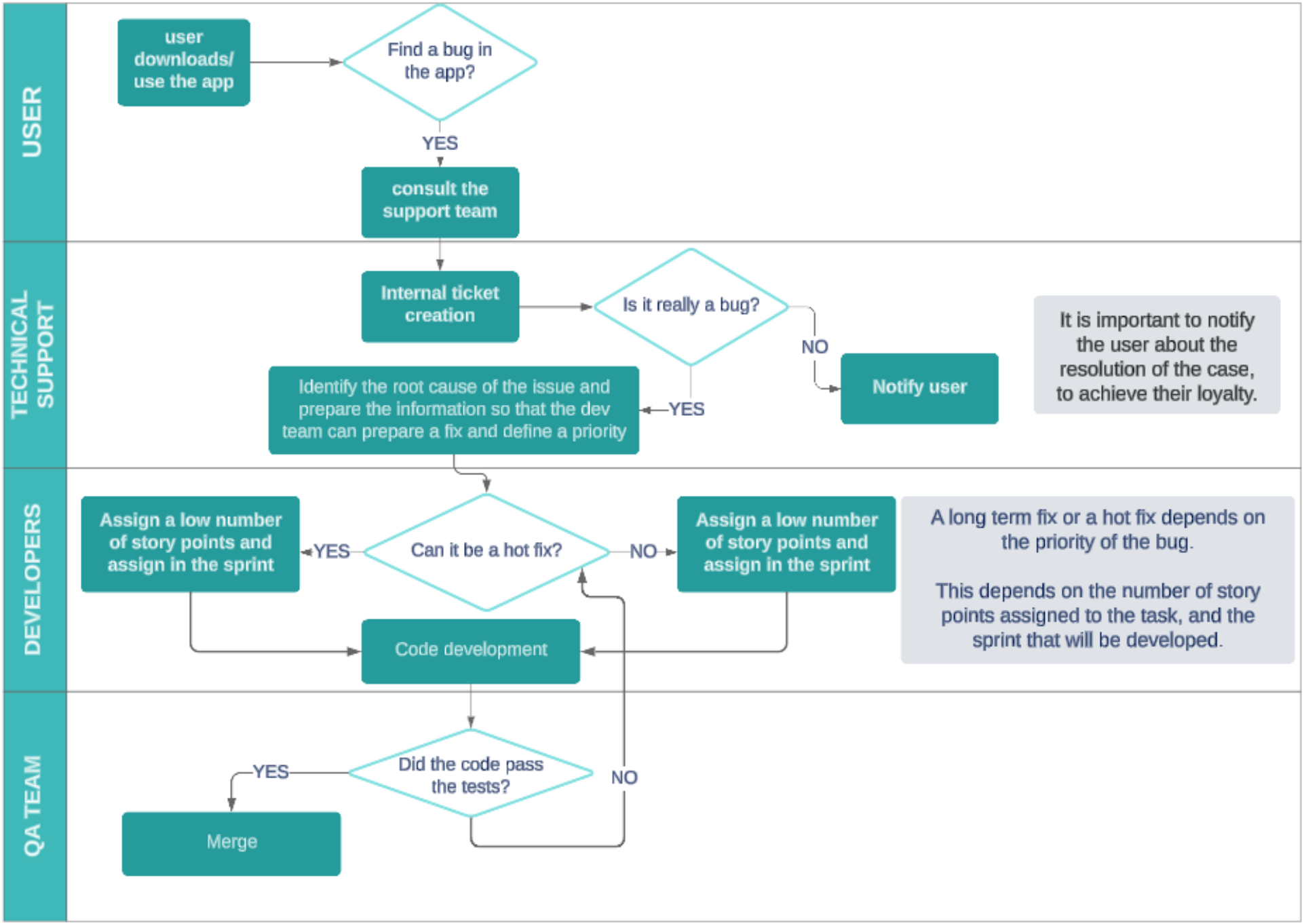
In this section, we must ask ourselves: what problems are we looking to solve with Lipa? This may include:

- Fast BTC transactions using Lightning with reduced costs.
- Facilitating Bitcoin adoption.

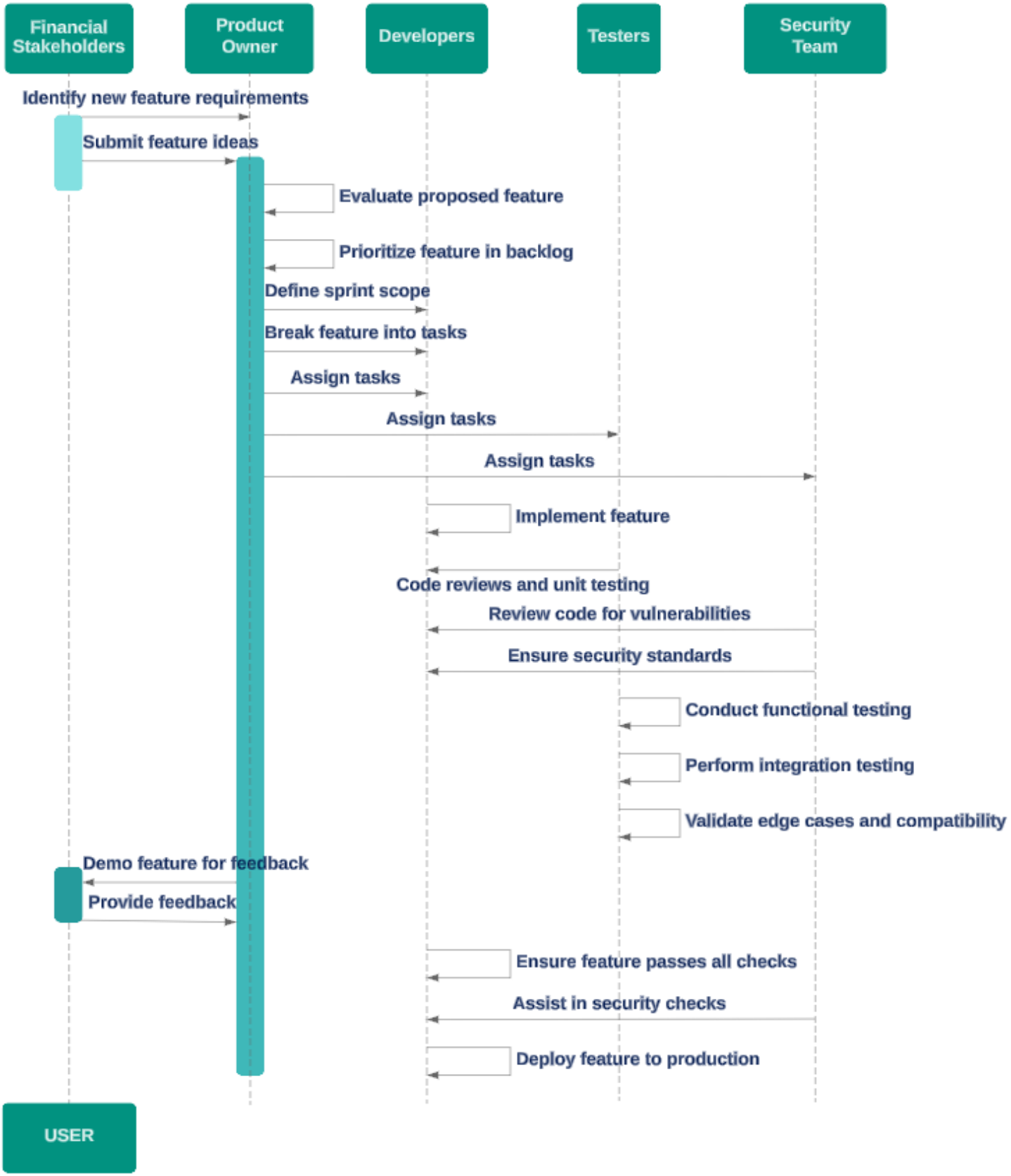
Based on this, we can identify the following stakeholders:

- **End Users:** Individuals or businesses who will use the wallet to send/receive BTC.
- **Technical Support:** Aids post-development.
- **Product Owner:** Defines business objectives and prioritizes features.
- **Financial Stakeholders:** Investors interested in the product.
- **Developers:** Implement the system (backend Lightning, frontend wallet).
- **Testers:** Ensure software quality.
- **Security Team:** Assess wallet vulnerabilities.

Each of these stakeholders has different interactions. Which we will review below using diagrams in a general manner.



Support Interaction Diagram



New Features Diagram

Some recommendations for metrics to use are as follows:

Metric name	Description
Requirement Coverage (%)	(Number of Requirements Tested/Total Number of requirements) * 100
Code Coverage (%)	(Lines of Code Tested/Total Lines of code) * 100
Pass Rate (%)	(Number of Passed Tests/Total number of tests Executed) * 100
Bug Detection Rate (%)	(Number of Defects Detected/Total number of test cases executed) * 100
Bug Leakage Rate	(Bugs Found in Production/Total bugs (Testing + Production)) * 100
Test Execution Cycle Time	Time from test initiation to completion (including defect retests).
Bug Closure Time	Sum of Times to Close All Defects / Total number of defects closed
Test Automation Rate	(Number of Automated Test Cases/Total number of test cases) * 100
Customer-Reported bug	Count the total bugs reported via customer feedback or crash reports.
Bug resolution time	Time to resolve a defect after it has been detected.

Audit the existing CI/CD pipeline and automation infrastructure

Integrations and External Credentials:

- Evaluate the use of access tokens, API keys, and other secrets.
- Review the permissions assigned to external integrations (e.g., connection with deployment tools).

Pipeline Security

- Validate who has access to the pipeline and their permissions.
- Ensure compliance with the principles of least privilege.
- Review the configuration of credentials and key secrets.
    - Verify the implementation of multi-factor authentication (MFA).
    - Confirm that there are no hardcoded credentials in the source code or scripts.

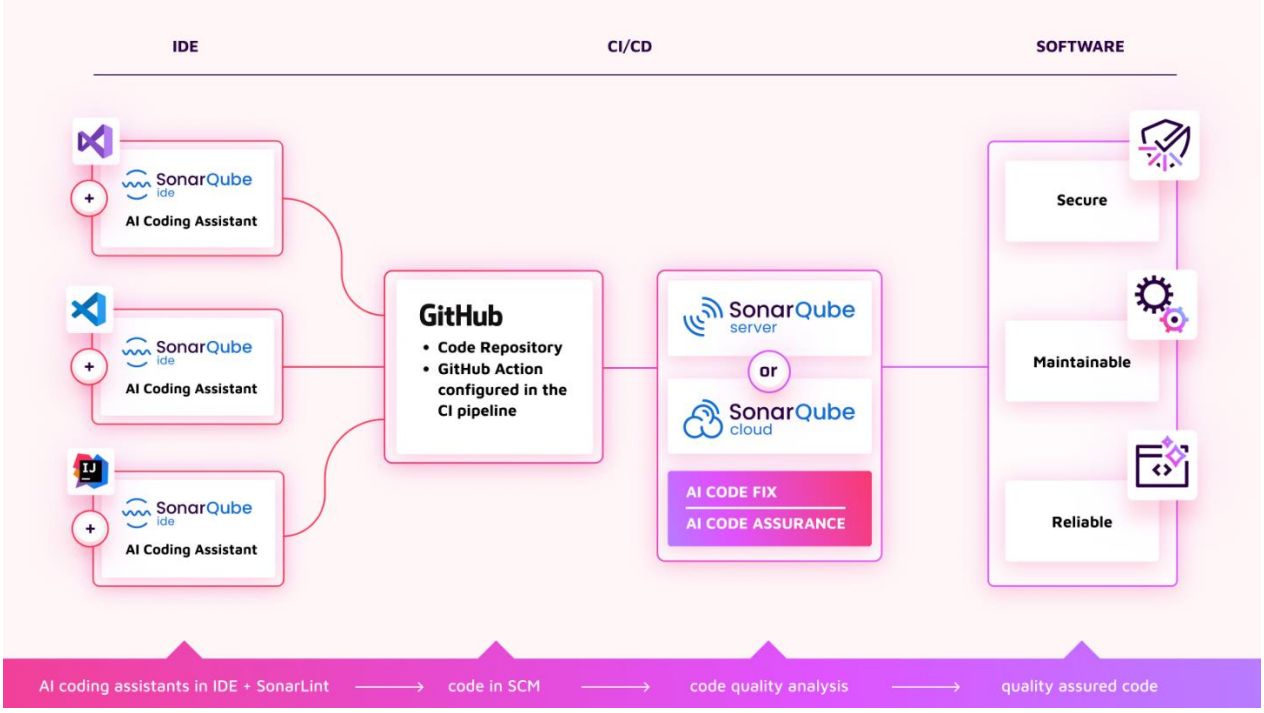
Repository Security

- Check for protected branches (e.g., prevent direct pushes to main or master).
- Verify that merges require code reviews.
- Enable automatic vulnerability scanning for dependencies and code.

Using multiple tools may seem like a significant expense, but for a small team, having tools that simplify their work can be extremely useful, allowing them to focus on other tasks that require greater attention. For this section, tools such as **SonarQube** or **Snyk** are suggested, which are evaluated below:

Resolves unique code quality challenges as a result of combining AI and human code

- Scans and detects bugs and vulnerabilities in code, even deeply layered issues
- Guides you to remediate code issues both in the IDE before the code is committed and within your DevOps workflow
- Powerful static code analysis with thousands of rules for over 30 languages
- Built-in review workflows and reports facilitate comprehensive code assessments with actionable insights for remediation
- Powerful Quality Gates enforce defined code quality standards, blocking merges and deployments that aren't production-ready



Snyk is a developer security platform that enables application and cloud developers to secure their whole application — finding and fixing vulnerabilities from their first lines of code to their running cloud.

Also empower developers to maintain code security means that vulnerabilities are addressed early in and across the development process, leading to fewer bottlenecks and more efficient development cycles and Snyk's DeepCode AI ensures a high-level of accuracy — without hallucinations — in scans

and suggested code fixes by combining symbolic and generative AI, several machine learning methods, and the expertise of Snyk security researchers.

Helps with the transition to DevSecOps, integrating developer security into existing systems and providing visibility and governance across all applications.

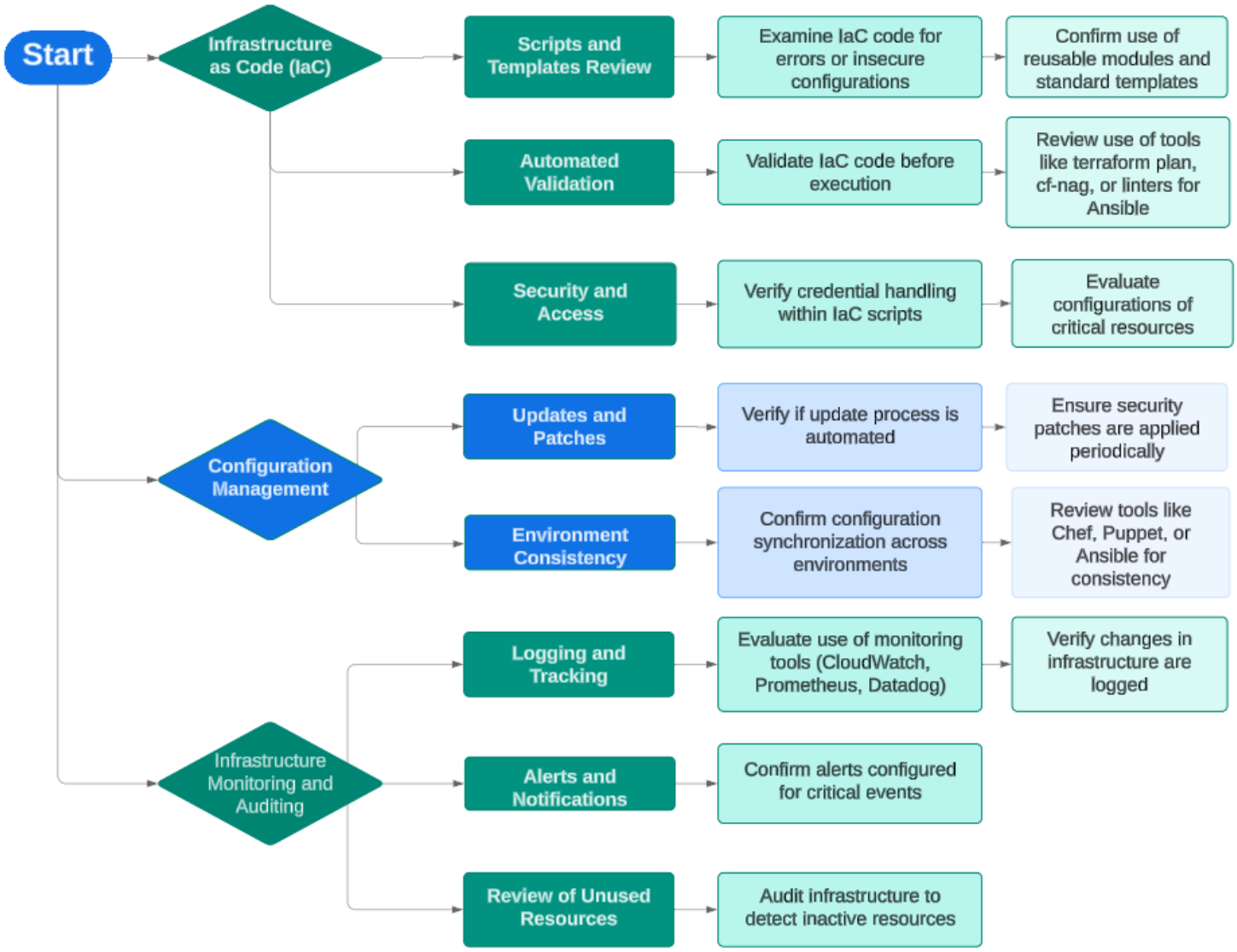
Comparison of features and prices.

SNYK	SONAR
Team price $25	**Team price $32**
- Cloud source code management integration, including GitHub, GitLab, Bitbucket and Azure Repos - Scanning throughout the SDLC - Real-time custom code scanning - Dev-first fix examples in the IDE by the Deepcode AI Engine - Dev-first integrations, including IDE, CLI, Repo, and CLI - Automated fixes in the IDE with Deepcode AI Fix - Scanning throughout the SDLC - Dev-first integrations, including IDE, SCM, CLI, and Terraform Cloud/Enterprise - Data encryption in transit and at rest SOC 2 Type II, GDPR, ISO27001/ISO27017 compliant	- Scan your private projects (up to 50k lines of code) - Scan your public projects (unlimited lines of code) - Issue detection and SAST - Main branch & pull request analysis - DevOps platform integration - Advanced secrets detection - AI CodeFix (early access) - AI Code Assurance - Analyze feature and maintenance branches - Customize quality standards

Infrastructure Automation Audit

Adopting an Infrastructure as Code (IaC) approach involves defining and provisioning infrastructure using code that is executed automatically, instead of relying on manual processes. By treating IT infrastructure as code, organizations can automate management tasks and benefit from software development best practices, reducing human error in the process.

Here are some processes that can be followed to audit the infrastructure:



My recommendation is to use Prometheus to monitor infrastructure performance, as it is an open-source tool. We will review its features below.

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open-source project and maintained independently of any company.

Prometheus's main features are:

- A multi-dimensional data model with time series data identified by metric name and key/value pairs
- PromQL, a flexible query language to leverage this dimensionality
- No reliance on distributed storage; single server nodes are autonomous
- Time series collection happens via a pull model over HTTP
- Pushing time series is supported via an intermediary gateway

- Targets are discovered via service discovery or static configuration
- Multiple modes of graphing and dashboarding support

Prometheus metrics for infrastructure monitoring, grouped by type of resource:

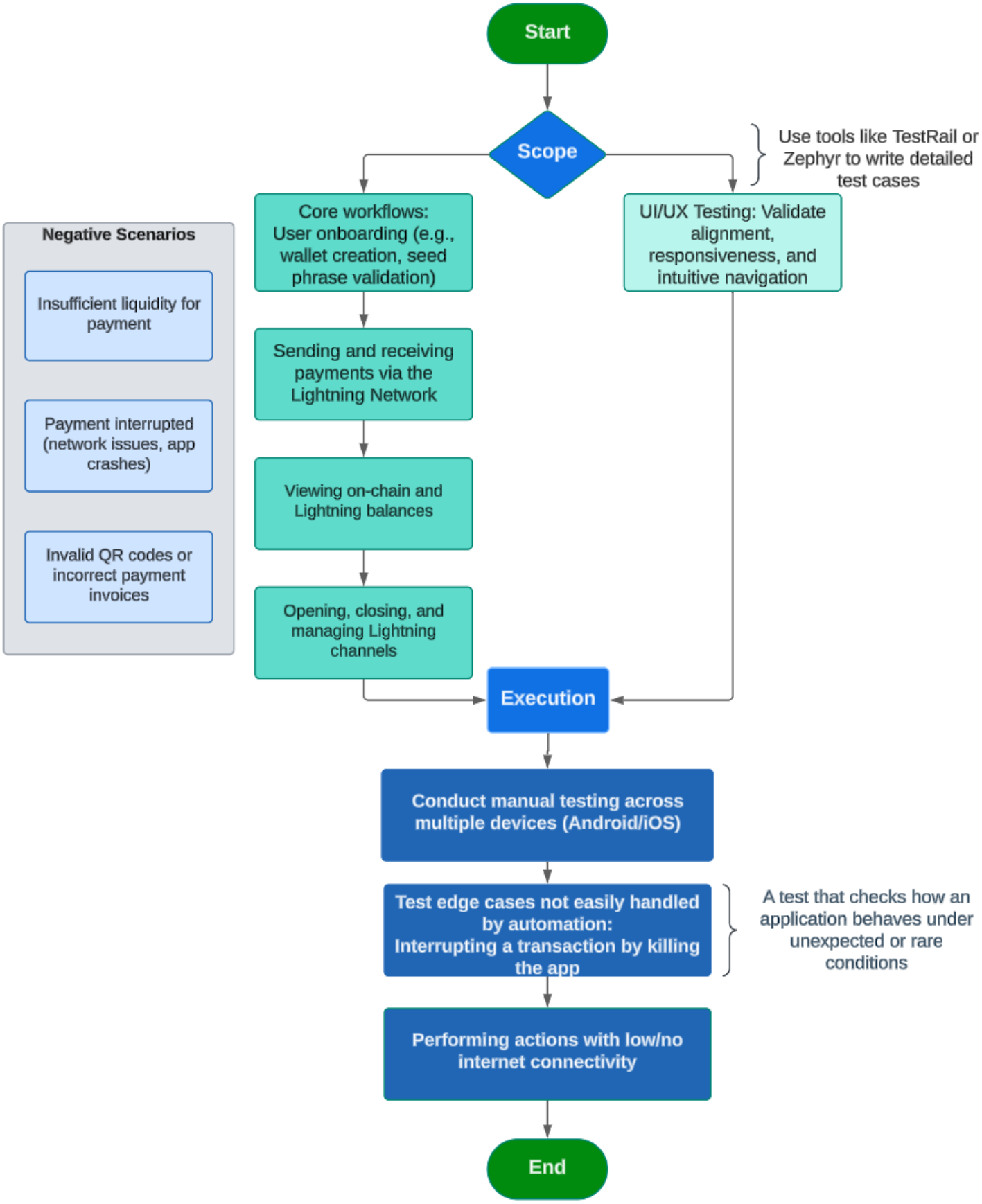
CPU Metrics	**node_cpu_seconds_total**: Total CPU time spent in various modes (e.g., user, system, idle). o Example: rate(node_cpu_seconds_total{mode="user"}[5m]) o Insight: CPU usage over time in user mode. **node_load1**, **node_load5**, **node_load15**: Average CPU load over the last 1, 5, and 15 minutes. o Insight: Overall CPU load on the system.
Memory Metrics	**node_memory_MemAvailable_bytes**: Amount of available memory in bytes. • Insight: Indicates how much memory is free for use. **node_memory_MemTotal_bytes**: Total memory available on the system. • Example: 1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) • Insight: Memory usage percentage. **node_memory_Active_bytes**: Active memory in bytes (used actively by processes).
Disk Metrics	**node_filesystem_avail_bytes**: Available disk space in bytes. • Insight: Remaining space on a specific disk. **node_filesystem_size_bytes**: Total size of the filesystem. • Example: (node_filesystem_size_bytes - node_filesystem_avail_bytes) / node_filesystem_size_bytes • Insight: Disk usage percentage. **node_disk_reads_completed_total**: Total number of completed disk read operations. • Insight: Read activity on disks. **node_disk_writes_completed_total**: Total number of completed disk write operations.
System Uptime and Health	**node_time_seconds**: Current system time in seconds. • Insight: System's current time (useful for uptime calculations). **node_boot_time_seconds**: Time when the system was last booted. • Example: time () - node_boot_time_seconds • Insight: System uptime in seconds. **up**: Indicates whether a target is up (1 for up, 0 for down). • Insight: Basic health check for monitored targets.
Process and Service Metrics	**process_cpu_seconds_total**: Total CPU time consumed by a process. • Insight: Resource usage of specific processes.

	process_resident_memory_bytes: Resident memory size used by a process. • Insight: Memory footprint of specific processes. **node_service_run_time_seconds**: Total runtime of a service or process. • Insight: Tracks uptime of specific services.
I/O Metrics	**node_disk_io_time_seconds_total**: Total time spent on I/O operations. • Insight: Identifies I/O bottlenecks. **node_disk_io_time_weighted_seconds_total**: Weighted I/O time (combination of time spent and queue length). • Insight: Disk latency issues.
Kubernetes Metrics (if applicable)	If monitoring Kubernetes infrastructure using Prometheus, common metrics include: • **kube_node_status_capacity_cpu_cores**: Total CPU cores available on a node. • **kube_pod_container_resource_requests_memory_bytes**: Memory requested by pods. • **kube_deployment_status_replicas_unavailable**: Number of unavailable replicas in a deployment.

Mobile Test Concept

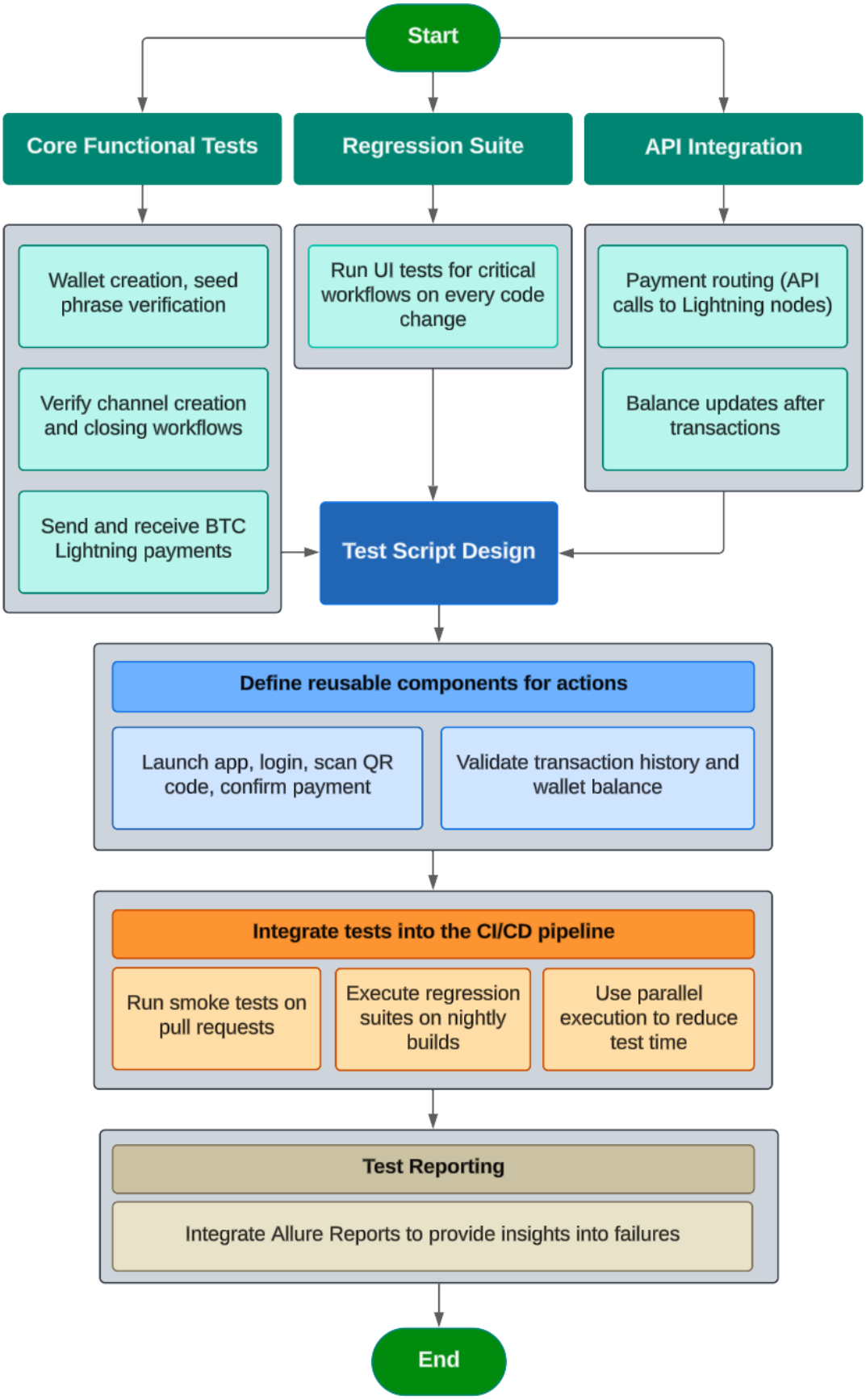
Manual Testing

Purpose: To validate new features, critical workflows, and edge cases that require human judgment.

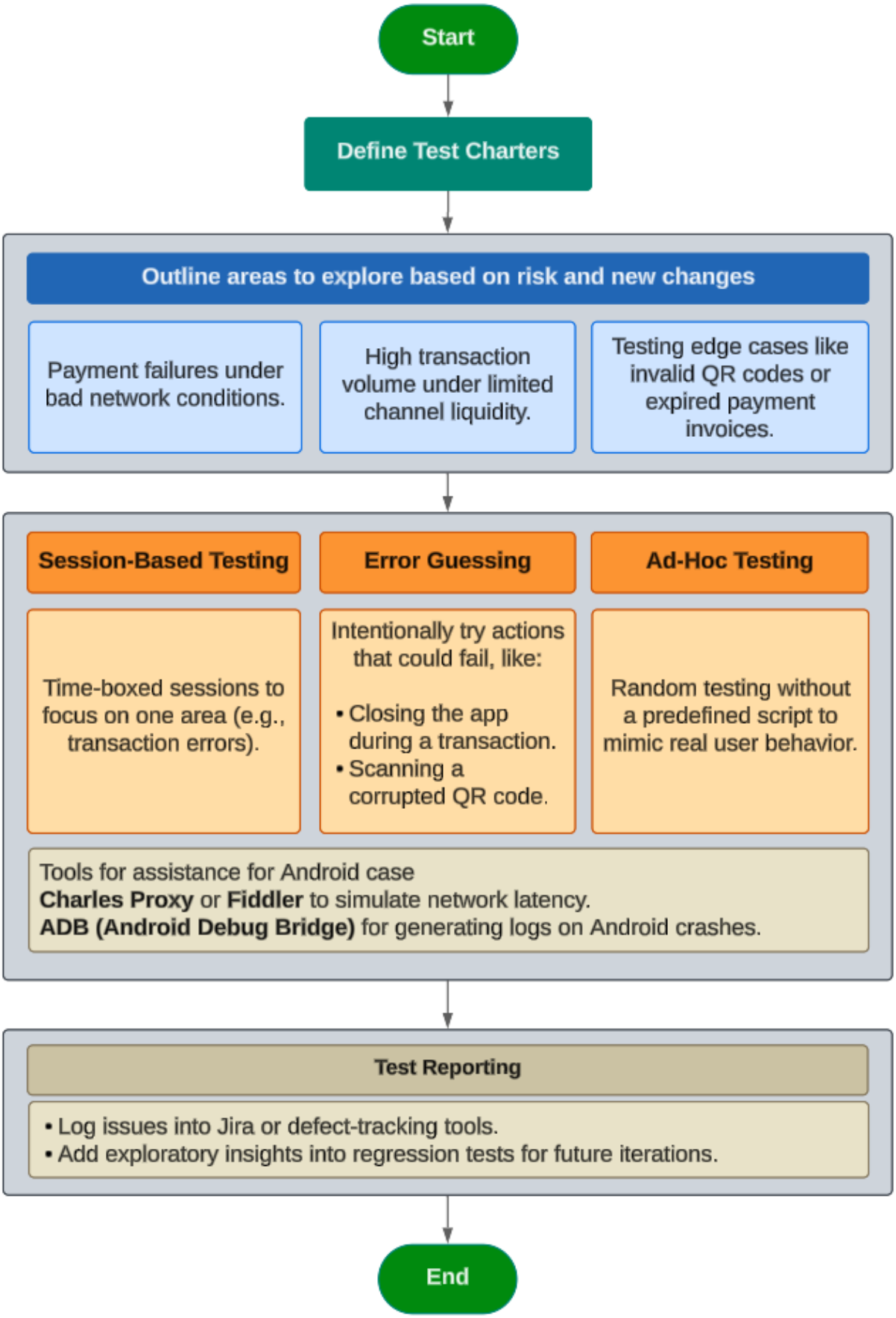


Automated Testing

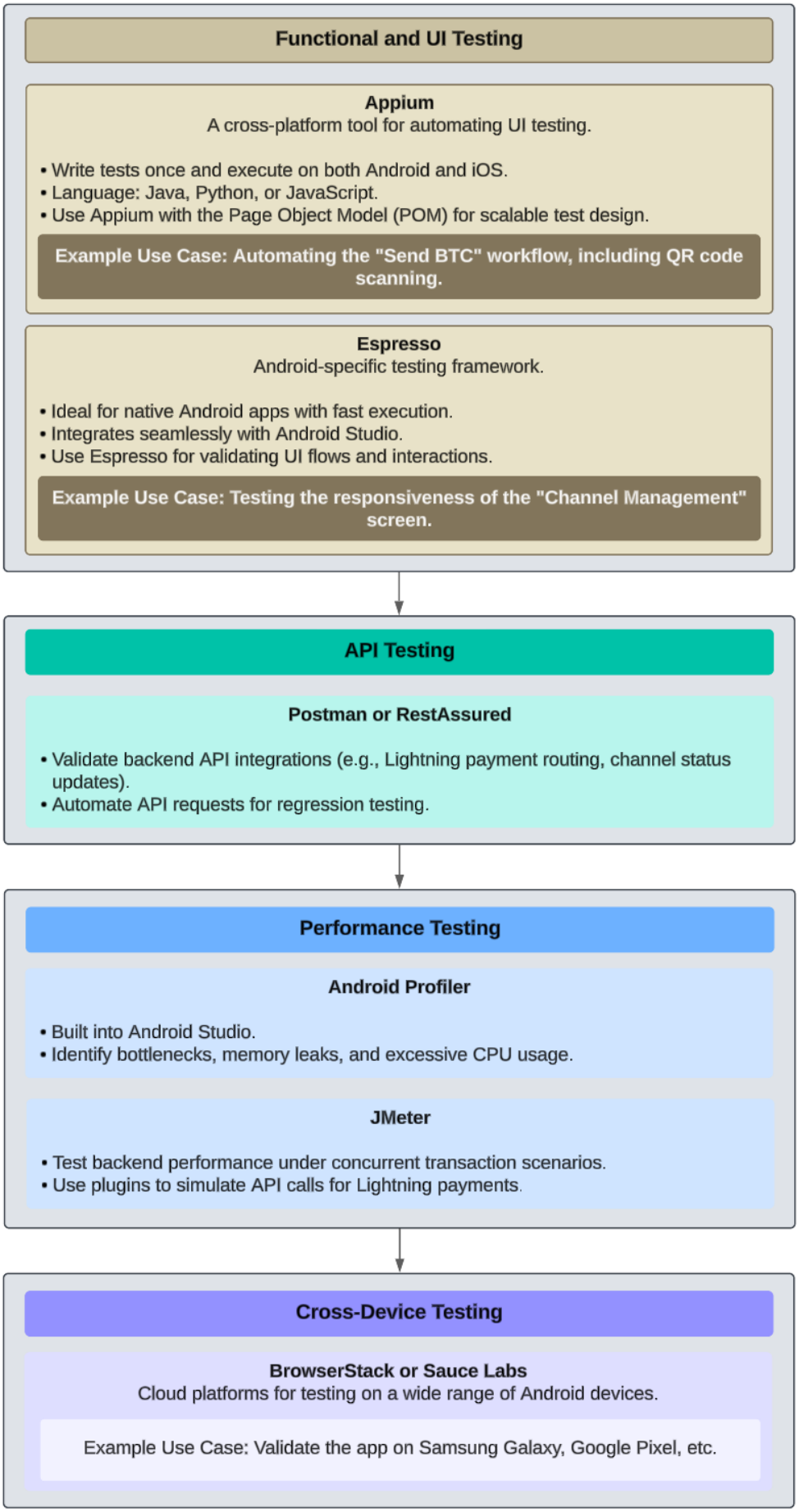
Purpose: To ensure consistent regression testing for repetitive and critical workflows.



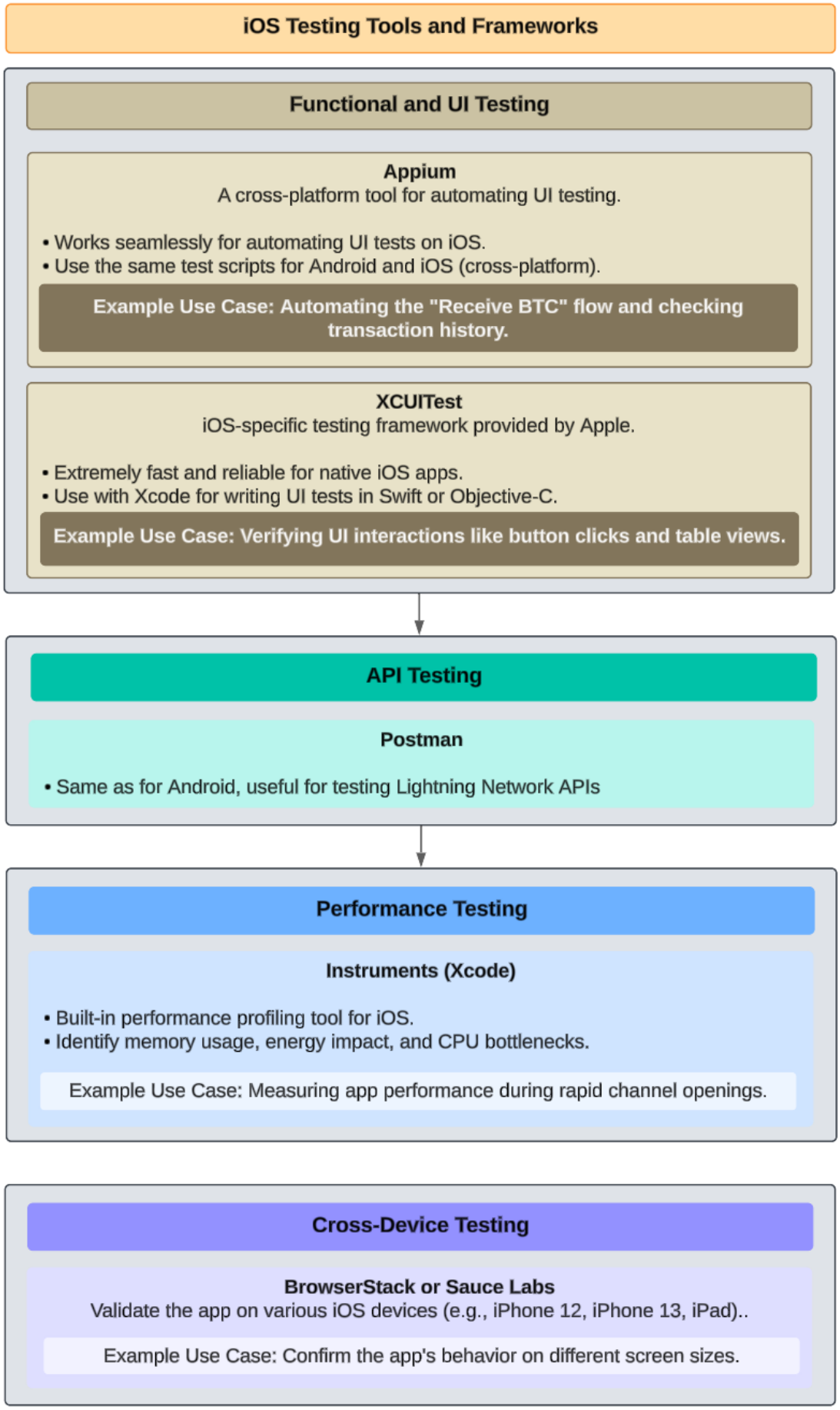
Exploratory Testing



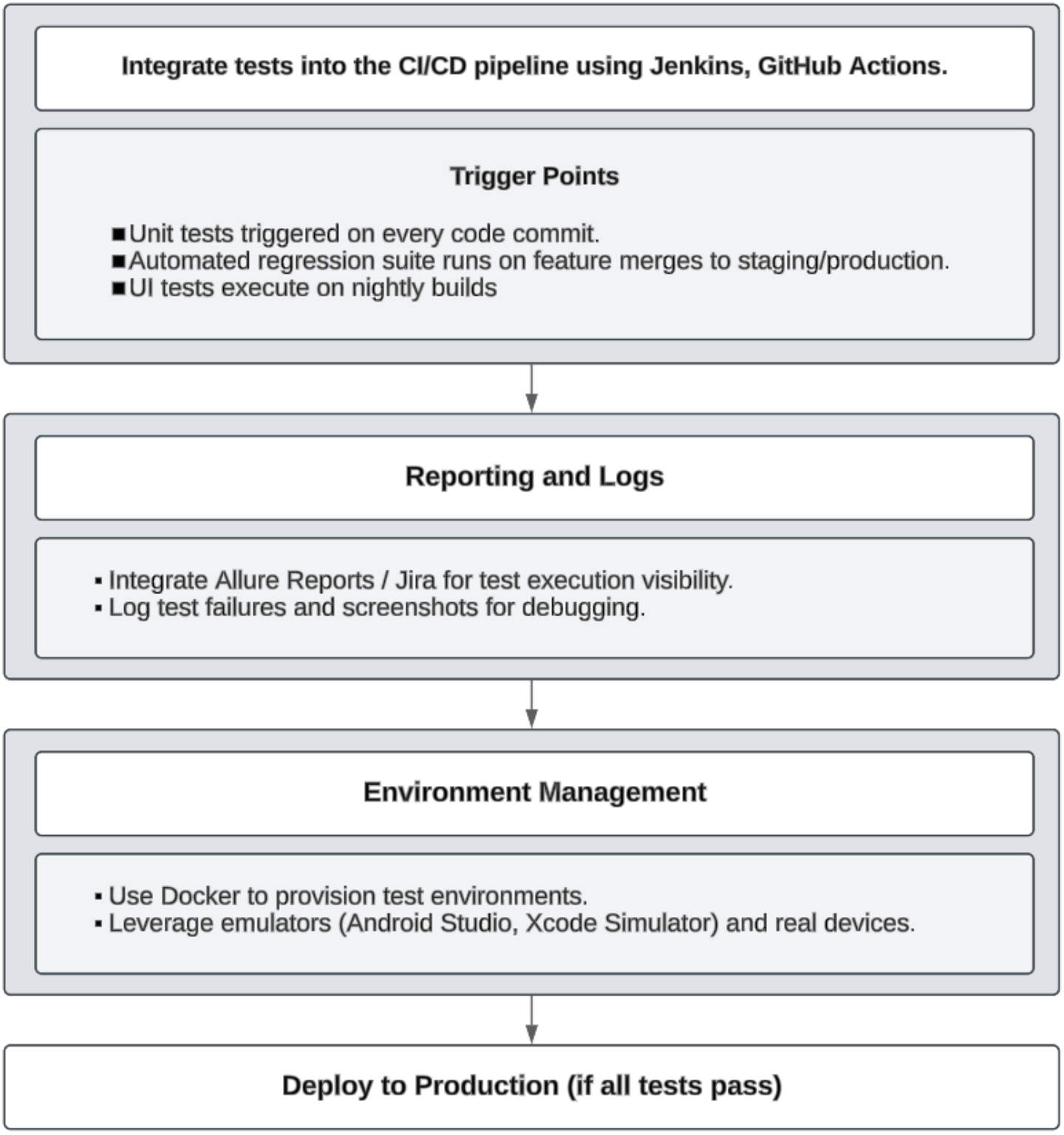
Tools and frameworks for Android



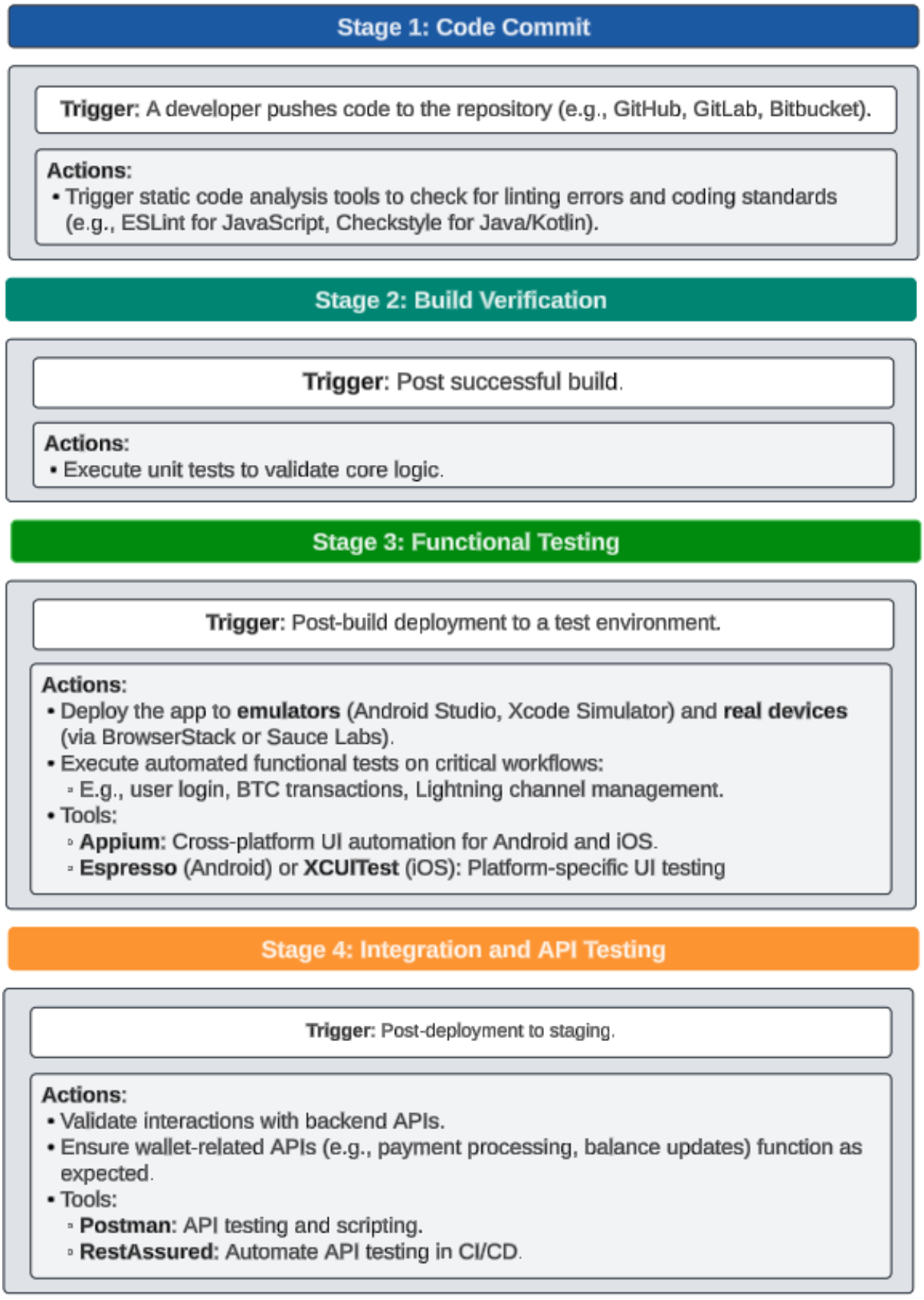
Tools and frameworks for iOS

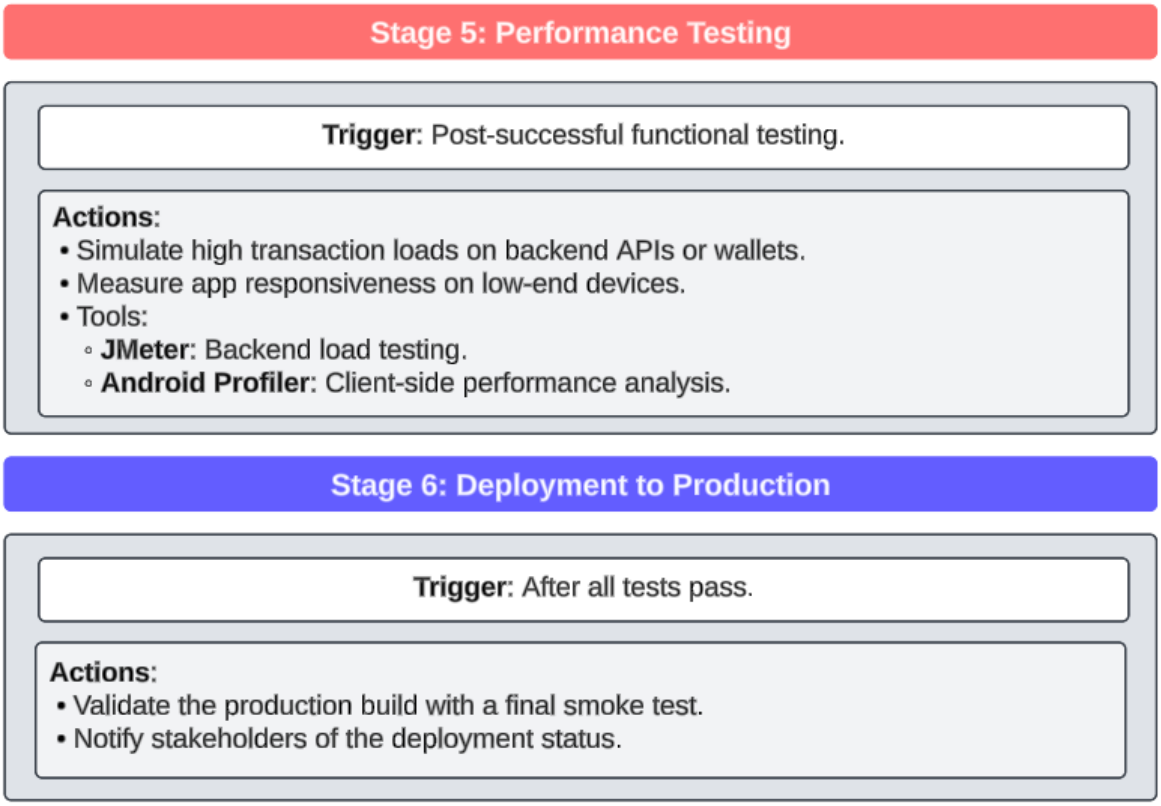


CI/CD Integration



CI/CD Pipeline Stages with Mobile Testing





Some tools to use are the following:

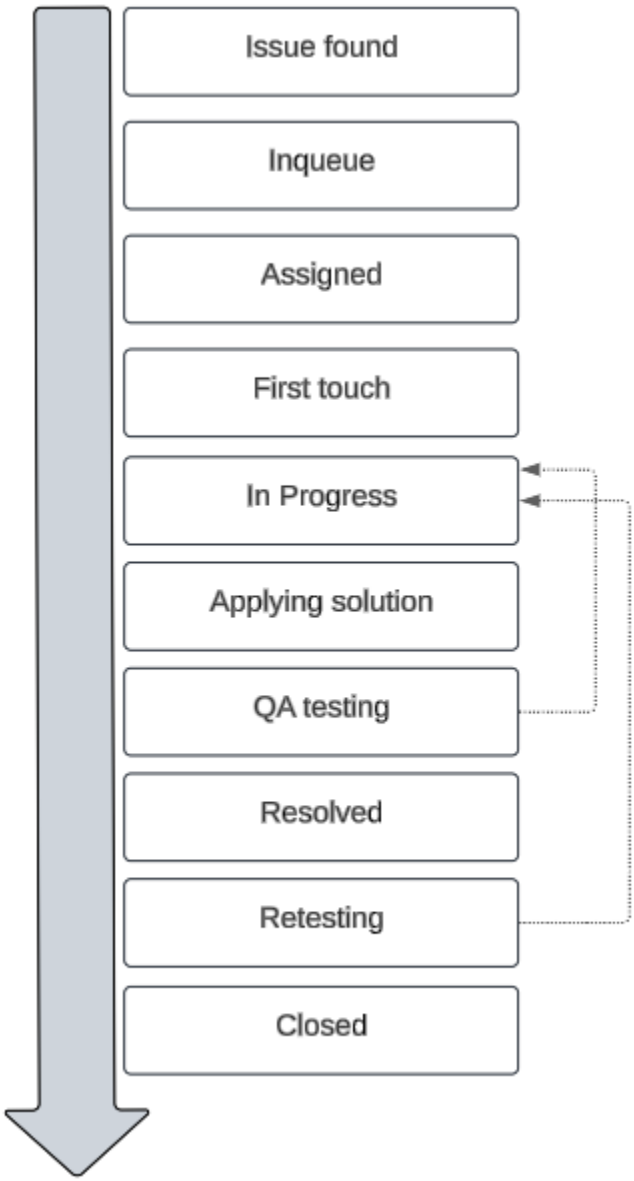
Category	Tool	Purpose
Source Code Management	GitHub, GitLab	Code version control and pipeline triggers.
Build Tools	Gradle (Android), Xcode (iOS)	Automate app builds.
Static Code Analysis	SonarQube	Enforce code quality and detect vulnerabilities.
Test Automation	Appium, Espresso, XCUITest	Automate functional and UI tests.
Cross-Device Testing	BrowserStack, Sauce Labs	Test on real devices and emulators.
API Testing	Postman, RestAssured	Validate backend API interactions.
Performance Testing	JMeter	Measure backend and frontend performance.
CI/CD Orchestration	Jenkins, GitHub Actions	Automate pipeline execution and reporting.

Defect management workflow

For issue tracking, based on my experience, I recommend using Jira due to its widespread adoption and ease of implementing and documenting processes.

Additionally, in my current job, we have a system that uses the Jira API to monitor the tickets assigned to developers and evaluate performance metrics through proprietary dashboards. These dashboards display data such as the average time taken to resolve tickets, the time spent in each status, and more.

Below is a diagram showing the ticket statuses:

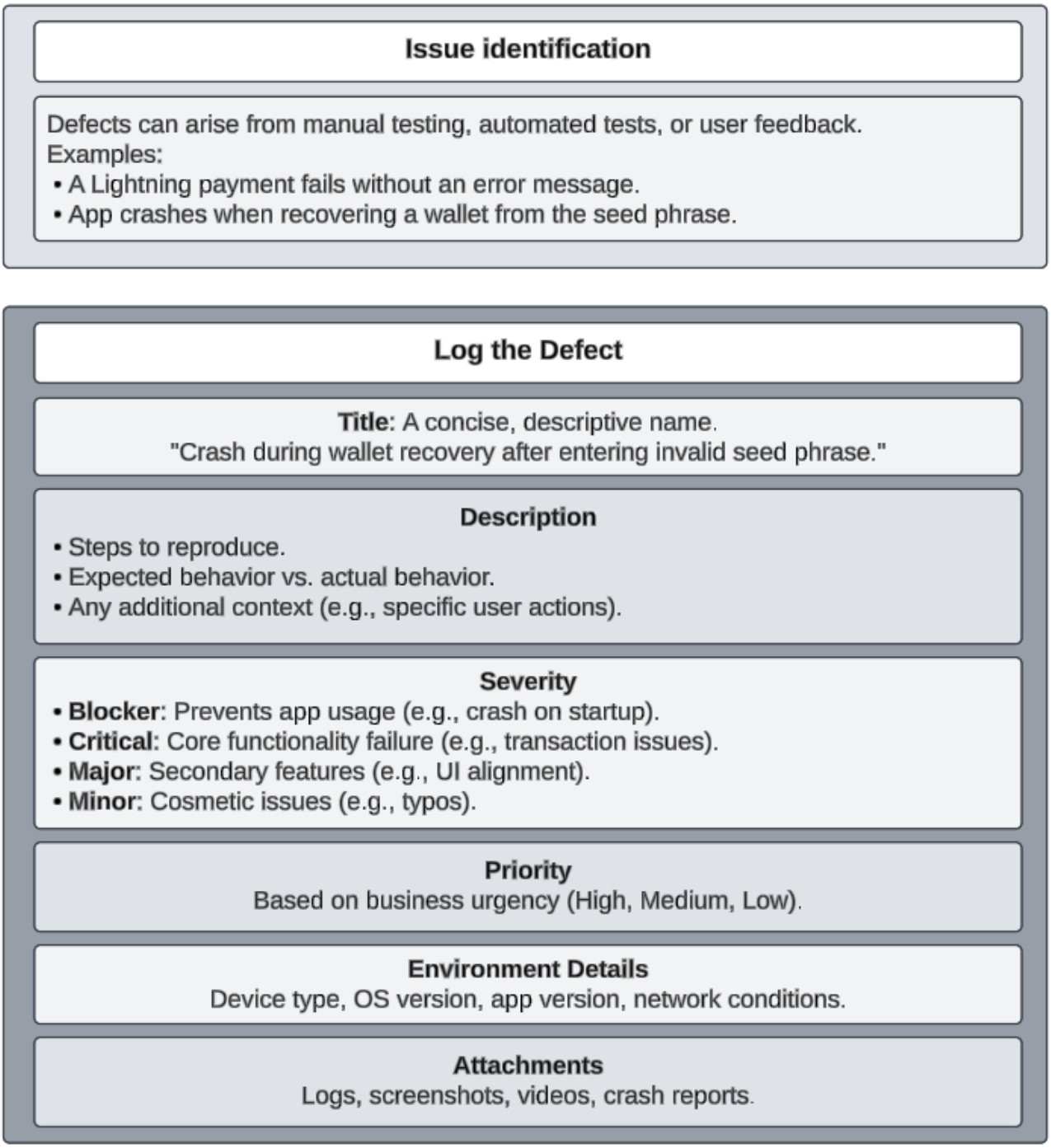


When finding an issue, it is very important to detail all the related information, so that the assigned dev can replicate the issue and find a solution.

As we saw in previous diagrams, this can be an issue found by the user, who in this case requires technical support to determine if it is really a bug or simply a misuse, which will save developers time by discarding "issues".

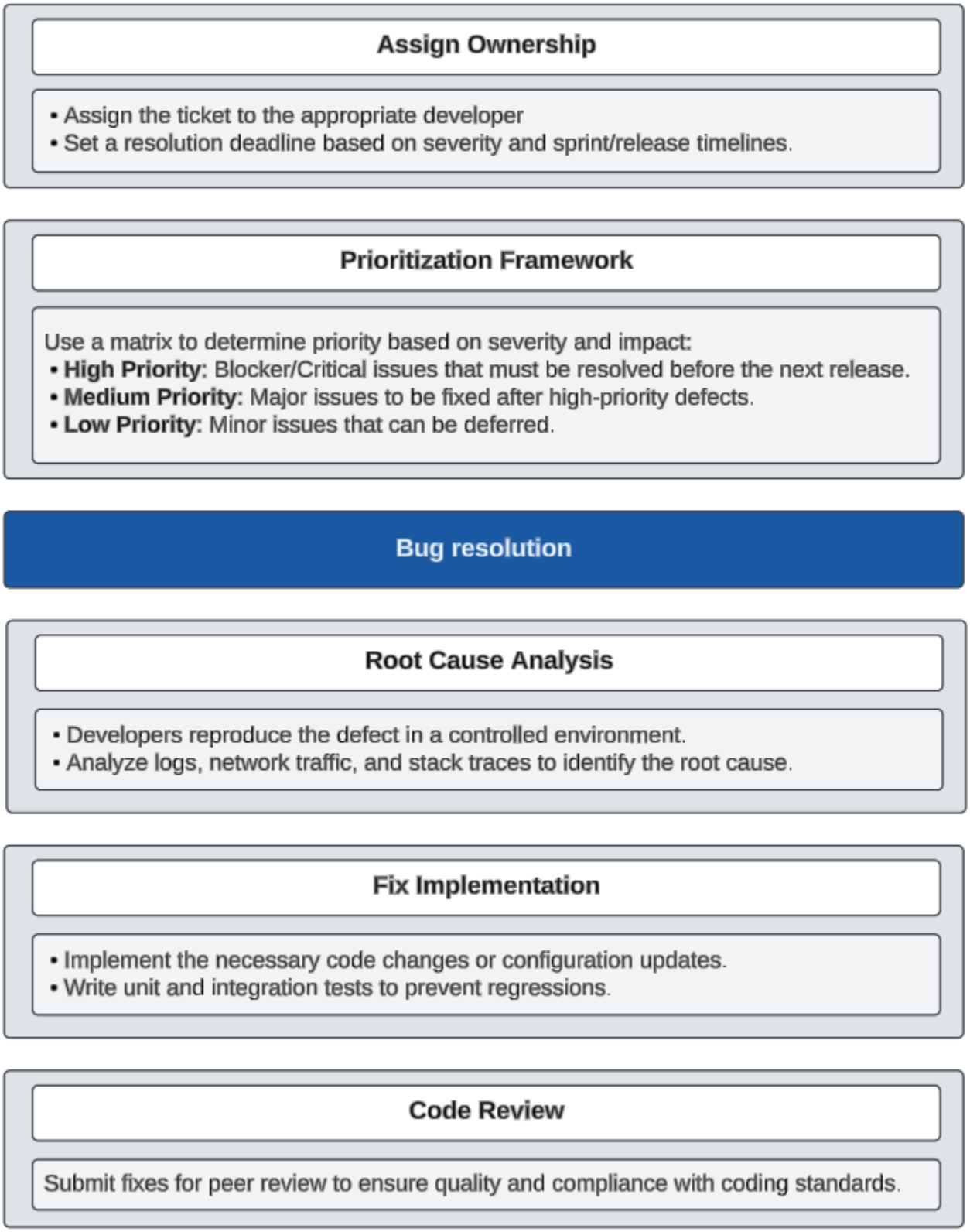
It is also important within an organization to take care of senior time, so developers should only focus their efforts on critical tasks.

The following diagram shows the fields that a ticket must have after identifying an issue.



Then, it’s necessary Validate the bug, assign severity/priority and assign them to the appropriate teams.

For these cases, daily meetings can be held where discussions are held with the team members to determine the priority of the bug and classify it. The ticket is then assigned to someone.



Verification

Retesting

- QA verifies the defect in the environment where it was originally identified.
- Execute the original test steps and confirm the expected results.

Regression Testing

- Run automated or manual regression tests to ensure the fix hasn't introduced new issues.
- Focus on related features or modules to check for side effects.

Closure

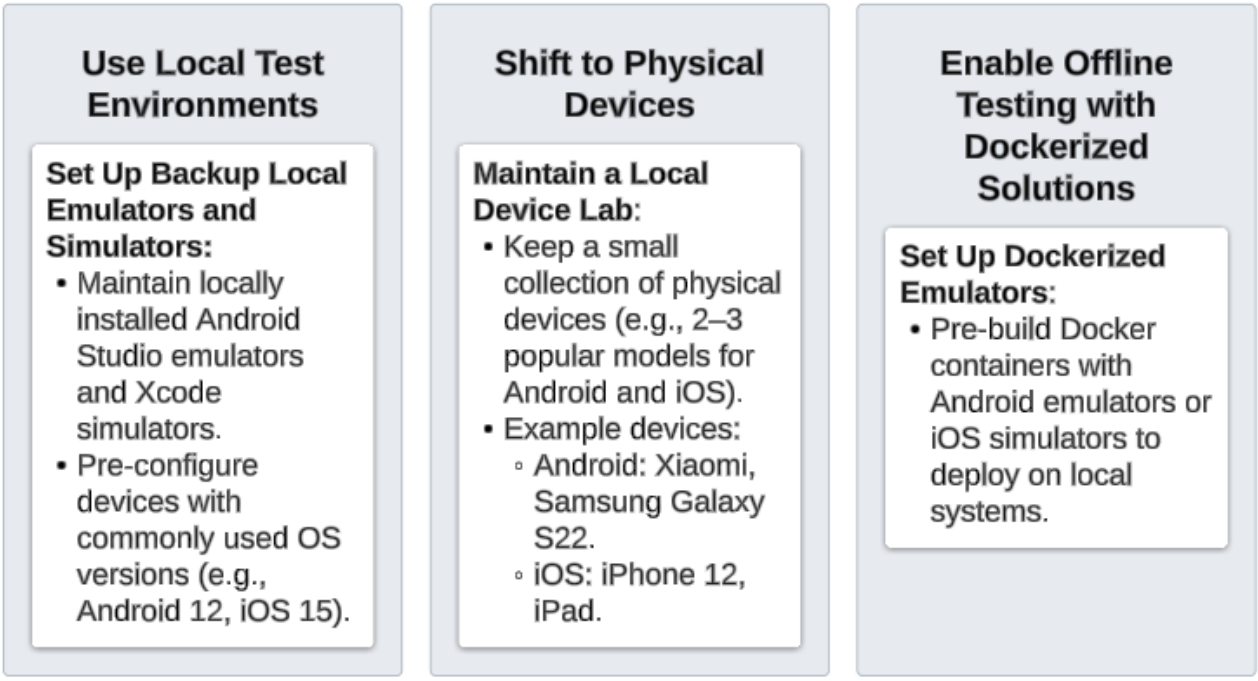
Bug closure

- Close the defect in the tracking tool if:
  - It has been successfully retested.
  - Regression tests confirm no new issues.
- Add a summary of the resolution and testing results.

Reopen Defects

- If the issue persists or reoccurs, reopen the defect and attach additional findings.

Fallback strategy in case of testbed outages



One of the options that caught my attention is running emulators with docker containers, from what I have researched it is a reliable and efficient solution for running an Android emulator for testing, debugging, and continuous integration/continuous deployment (CI/CD) in the development pipeline.

Benefits and Features

- Isolation of the Android emulator in its own environment, preventing conflicts with other software on the host machine, working in headed/headless mode.
- Reproducibility of testing environments through the use of portable Docker containers.
- Scalability of resources to meet the needs of the project.
- Easy switching between different versions of the Android emulator for testing on various Android OS versions.
- Simplified setup and management of the Android emulator through the use of Docker’s user-friendly interface/ docker-compose file.
- Streamlined integration of the Android emulator into the CI/CD pipeline.
- Ability to run the Android emulator on cloud infrastructure such as AWS, GCP, and Azure using Docker.
- No need to rely on external services or device farms (for android).
- Relief from the time-consuming and complex setup process of the Android emulator on your host machine.

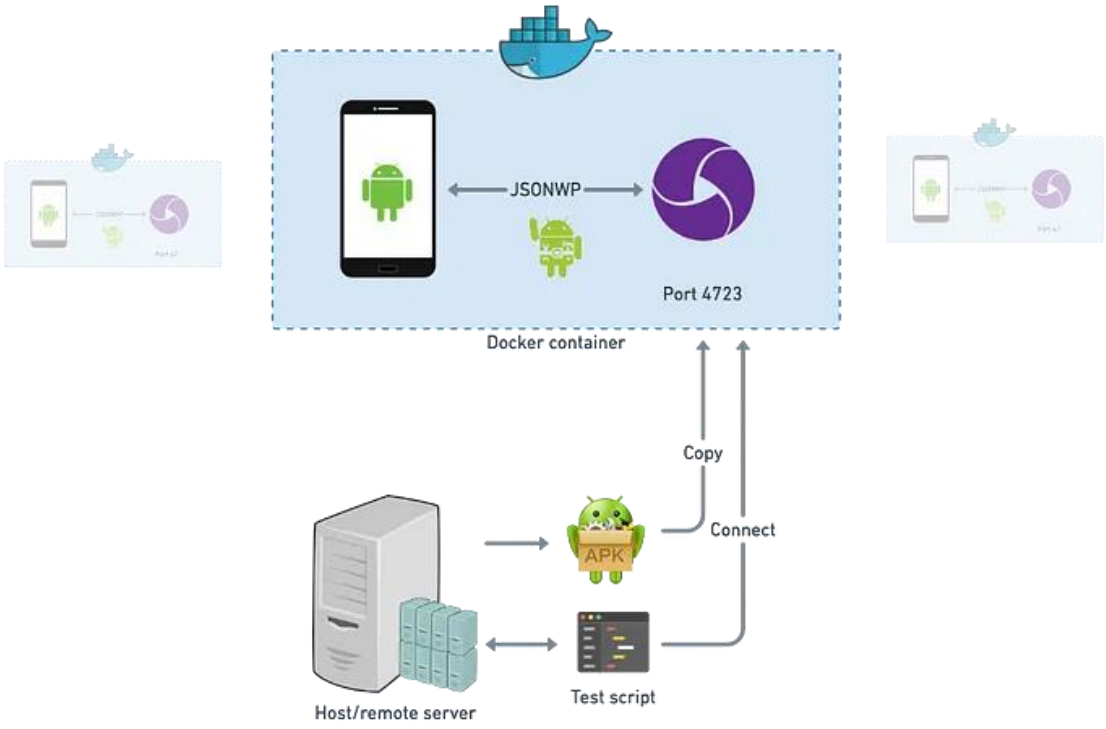


Figure 1 A diagram illustrating the use of images in automation testing and scaling

Testbed Providers

Reviewing the information from the testbed providers, each of them provides very good coverage for testing, but I consider that despite having a smaller number of devices compared to BrowserStack, Sauce Labs is a very good alternative, in addition to being a very good option for small teams.

What also makes it a very good option is that it has integrated CI/CD tools and has automated testing capabilities with detailed reporting.

My approach would be to use SauceLabs services and as a backup have AWS Device Farm since it has a Pay-as-you-go option. This service also provides the advantage of testing real devices, this can be a cost reduction against buying the device. A number of testbed providers characteristics are listed on the following page.

	BrowserStack	Sauce Labs	AWS Device Farm
Device Coverage	BrowserStack provides access to over 3000+ real devices and browsers, making it one of the most extensive device farms for cross-platform testing.	Sauce Labs offers 1000+ device combinations, including strong support for mobile emulators and simulators, which is ideal for early-stage testing.	AWS Device Farm provides access to real devices with various OS versions, enabling testing on actual hardware for better accuracy.
Integration	• Seamless integration with popular testing frameworks like Appium and Selenium. • Supports CI/CD tools, including Jenkins, CircleCI, and GitHub Actions, for automated testing pipelines.	• Built-in integration with popular CI/CD tools such as Jenkins, CircleCI, and Azure DevOps. • Native support for testing frameworks like Appium, Espresso, and XCUITest.	• Compatible with multiple frameworks like Appium, Calabash, Espresso, and more. • Direct integration with AWS CI/CD services, such as AWS CodePipeline and CodeBuild.
User Experience	• Real-time testing with live interaction on devices. • Visual logs, screenshots, and video recordings for detailed bug reporting.	• Comprehensive debugging tools, including live interaction with devices and network monitoring. • Automated testing capabilities with detailed reporting.	• Offers parallel testing on multiple devices to speed up regression testing. • Customizable test scripts and easy integration with AWS.
Pricing	• Starts at $129/month for individual plans. • Enterprise pricing is customized based on needs.	• Starts at $99/month, making it more affordable for smaller teams. • Enterprise pricing is available for larger organizations.	• Pay-as-you-go: $0.17 per minute/device. • Flat-rate plans: Start at $250/month for unlimited testing.
Pros	• Extensive device coverage. • Easy setup with minimal configuration. • Real devices provide highly accurate testing results.	• Strong support for emulators/simulators, reducing cost during early testing phases. • Competitive pricing for individuals and small teams. • Detailed analytics and debugging tools.	• Cost-effective for smaller teams due to flexible pay-as-you-go pricing. • Tight integration with AWS services. • Real devices ensure highly reliable results.
Cons	• Higher starting cost compared to competitors. • Limited support for emulators and simulators, focusing mainly on real devices.	• Limited real-device coverage compared to BrowserStack. • May require additional setup for larger teams managing multiple pipelines.	• Smaller device coverage compared to BrowserStack. • Requires familiarity with AWS services for effective use.
Best For	Large-scale device testing	Early-stage testing with emulators	Cost-efficient real device testing

Sources

JIRA

Management and issue tracking tool developed by Atlassian.
API:https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/#version

Mobile App Testing

https://www.lambdatest.com/learning-hub/mobile-app-testing

Appium

Open-source tool for automating native, mobile web, and hybrid applications on iOS and Android platforms: https://appium.io/docs/en/latest/intro/

Espresso

Testing framework for Android UI tests: https://developer.android.com/training/testing/espresso

BrowserStack

cloud web and mobile testing platform that enables developers to test their websites and mobile applications across on-demand browsers, operating systems, and real mobile devices: https://www.browserstack.com/

Jenkins

open-source automation server: https://docs.github.com/en/actions

SonarQube

open-source platform for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities:
https://www.sonarsource.com/products/sonarqube/

Snyk

A developer security platform that helps developers find and fix vulnerabilities in code, open-source dependencies, containers, and infrastructure as code. https://snyk.io/product/

XCUITest

UI testing framework by Apple for testing iOS applications:
https://developer.apple.com/documentation/xctest/ui_testing

Xcode

Apple's integrated development environment (IDE)

- Documentation: https://developer.apple.com/documentation/xcode
- Support: https://developer.apple.com/support/xcode/

JMeter

An open-source software designed to load test functional behavior and measure performance
https://jmeter.apache.org/

Sauce Labs

Cloud-based platform for automated testing of web and mobile applications: https://saucelabs.com/

RestAssured: A Java DSL for simplifying testing of REST-based services.

- Official Website: https://rest-assured.io/

Gradle: Open-source build automation tool that is designed to be flexible enough to build almost any type of software. https://gradle.org/

AWS Device Farm

- Website: https://aws.amazon.com/device-farm/
- Documentation: https://docs.aws.amazon.com/devicefarm/

Docker

- Documentation: https://docs.docker.com/