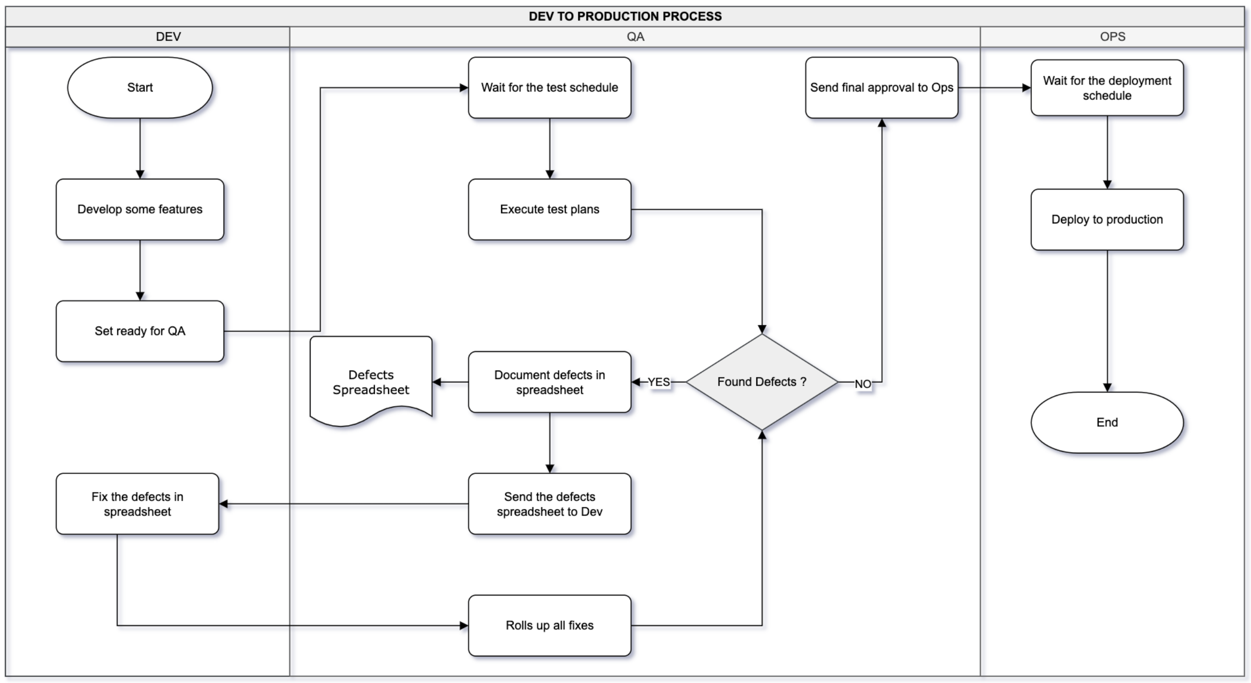
Using a flow chart diagram, please describe the testing process that the team utilizes to get the Phoenix code to production


DEV TO PRODUCTION PROCESS
DEV
QA
OPS
Start
Develop some features
Set ready for QA
Wait for the test schedule
Execute test plans
Found Defects ?
YES
NO
Document defects in spreadsheet
Defects Spreadsheet
Send the defects spreadsheet to Dev
Fix the defects in spreadsheet
Rolls up all fixes
Send final approval to Ops
Wait for the deployment schedule
Deploy to production
End

(Flow Chart of Production)

During development, developers begin by adding new features. Once completed, these features are marked as ready for QA. The QA team then waits for the test schedule before executing the test plans. If any defects are identified, QA records them in a defects spreadsheet and returns it to the developers. Developers address the issues, update the spreadsheet, and submit the revisions back to QA for retesting until all defects are fixed. If no issues are found, QA gives the final approval and passes the release to Operations. In the deployment phase, the team waits for the scheduled deployment, then deploys the approved code into production. This process is largely manual and linear, involving multiple handoffs between Development, QA, and Operations.

What are the main issues preventing Pheonix team from having a successful testing strategy?

The Phoenix team faces challenges due to an outdated, siloed, and slow testing process, leading to unreliable and inefficient software delivery. The primary issue is an excessive reliance on manual testing, where most verification is performed manually, which is both time-consuming and prone to errors. Consequently, only small portions of code are tested at a time, which limits test coverage and increases the risk of undetected defects. Feedback on issues is often delayed, and defects are often discovered only after a full build is handed to QA, by which time they are already embedded in the system. This causes developers to rework old code, resulting in delays, wasted effort, and frustration. Additionally, test environments are often limited, fragile, and challenging to set up, with lengthy waits for approval and access that can take days or weeks, hindering progress. The lack of automation is evident, as there are no automated tests or continuous deployment pipelines; instead, manual scripts and checklists are relied upon. Organizationally, teams are siloed, with minimal collaboration, which leads to finger-pointing and a culture that makes quality everyone's problem but no one's responsibility. This fragmented approach extends the cycle time, making it take weeks or months to move changes from code to production, hindering rapid updates and bug fixes.

What are the tools that the rebellion has that can alleviate pain points, how do these tools fit into the current process that Pheonix has?

The Rebellion, led by Maxine and her allies, introduces a modern DevOps toolset and practices that directly address Phoenix's pain points. Automated testing, including unit, integration, and regression tests, provides developers with immediate feedback whenever code is committed. Early detection of issues at the source would reduce rework and accelerate the cycle time, eliminating the long waits for QA. Continuous Integration (CI) systems such as Jenkins further enhance efficiency by automating builds and test execution, ensuring that every code commit is validated and always in a deployable state. Alongside this, improved version control and branching strategies enable better use of Git, allowing teams to isolate changes, simplify merges, and track history more effectively, thereby reducing chaos in collaborative development. Deployment automation also plays a crucial role. Infrastructure as Code (IaC) and containerization tools, such as Docker, make environment setup consistent and repeatable, thereby removing the constant delays caused by unreliable test environments. Finally, real-time monitoring and telemetry provide visibility into production systems, ensuring that issues are identified quickly and fed back into the development cycle. By integrating these tools into the Phoenix process, the workflow evolves from a slow, manual, sequential series of steps into a continuous, automated pipeline.