PART 2

List of gaps in both teams' processes and/or testing strategy that compromises on overall product quality.

Team 1	Team 2
Problem: 50% unit test coverage is inadequate, a standard product's unit test coverage should never be less than 70%. The problem will start compounding as and when new features keep on adding, in which case not working on unit tests will only result in declining unit test coverage. Resultantly, more defects that could be caught early in the STLC will leak and reach the smoke/sanity round of tests consuming more time and effort to identify and fix. **Remedial Steps:** As a quality assurance owner, I would restrict PR merges or build deployments in case the unit test coverage drops below 70%	**Problem:** Since not all the tickets/bugs/enhancements/chores are being recorded in JIRA, some important tasks that were accomplished could potentially slip through the cracks. **Remedial Steps:** Even if a one-liner, should track work through a ticket in JIRA because it is important.
Problem: Cypress does not support mobile app testing, tablet testing. It only supports web browser testing at the moment. This limitation will impact the quality assurance on device compatibility for tablets and mobiles, which is an integral part. **Remedial Steps:** Will analyze the tablet and mobile device usage, and if it is comparatively lesser - will try to work on a POC and at least get the Priority 1 test cases automated for execution.	**Problem:** Sprint Planning is done once, before the starting of the sprint. A sprint can be of 2/4 or multiple number of weeks. Team dynamics and bandwidths keep changing if story points are not being tracked and there could be people not available in between. Most of the time, such a long planning is not accurate and might not help with the iteration burn down chart. **Remedial Steps:** Planning should happen for not more than 2 weeks at a time.
Problem: Cypress can invoke tests in one browser at a time additionally it does not even support running tests in Safari right now. This will leave a lot of potential issues on Safari because it cannot be automated. **Remedial Steps:** Get some kind of manual testing involved for the frontend tests in	**Problem:** Tests are being written by engineers, who might not have a tester's mindset, so they might only be covering the happy paths and not actually implement tests with the intention of breaking the APIs. Tests related to security, invalid payloads, or performance testing.

Safari, tests can be picked on the usage, or what customer base uses Safari.	**Remedial Steps:** Bring awareness around the need for tests that cover each and every aspect of API testing comprehensively so as to minimize bug leakage.
Problem: Tests are a part of the CI/CD pipeline which is good. But they run only on build deployments which is not right because there may be multiple breaking changes in a build which cannot be left to be tested only on build deployments. **Remedial Steps:** Everytime a PR is raised, at least the smoke tests should be executed. This way individual PR changes are already validated and only then these PR merges should be allowed.	**Problem:** The tests that are being executed automatically run as a part of the integration test suite. No individual focus is given to the APIs to test them for their standalone functioning. This may cause unexpected failures. **Remedial Steps:** Enhance individual test coverage of these endpoints and segregate it from the integration test suite and run it independently.
Problem: Since the developers are creating frontend tests or tests in cypress, they might just automate the acceptance criteria and not other different variations, such as test data manipulation, negative scenarios because they don't have a QA mindset. This makes the application not exposed to variations of data or user inputs with potential bugs hidden. **Remedial Steps:** Introduce concepts like boundary value analysis, equivalence partitioning and decision trees so that versatility of test data can be introduced to the suite increasing the coverage and touching various aspects of the application in different ways.	**Problem:** Test Scheduler, the tests are currently scheduled to run every build deploy to a shared development environment. Since this is a shared development environment, breaking changes can be introduced if they are not pre-verified on the PR creation stage. **Remedial Steps:** Allow PR merges only if the tests pass for the custom branch for that PR.
Problem: No retrospective meetings. These are important discussions that enable us to identify what went good/bad and what should be improved. **Remedial Steps:** Have a retrospective meeting	**Problem:** API Documentation, If there is no API documentation that's up to date - onboarding of new resources or even reducing tech debt by involving existing developers would become a challenge. **Remedial Steps:** Make a process of API documentation update with every release.
Problem: Test suites run every night against the dev environment. Testing should be done against a stable environment and not an environment where consistent code commits are happening.	**Potential Concern:** This is not mentioned, but if there are 30 API endpoints that are tightly integrated in the system, then performance becomes a major factor in the project as the throughput times eventually

Remedial Steps: Push for a more stable staging environment for automated test execution.	affect the data serving mechanism. **Remedial Steps:** Having performance test suite enabled for the API endpoints.
Potential Problem: If all tests run every night, even the ones that are not required, these are unnecessarily creating delays because of more time consumed and using more than required resources. **Remedial Steps:** Run tests only which are necessary for the deployment. And segregate tests into tiers. Tier 1 - daily, Tier 2 - alternatively and so on.	

Note: Since there is no mention of how the development environment is set up, it is always a best practice to follow a containerized approach of development so that a fix or an enhancement works fine on everyone's computers with the consistent set of dependencies and development environment.