Too often best practices for testing are an after-thought, or a non-thought with respect to the development and deployment of code.

This generally results in shortcuts or bandaids being applied to the SDLC of an organization which can build up over time and become a major constraint on the future product evolution.

With respect to a PHP driven application, my preference is PHPUnit. So briefly a friendly reminder to distinguish your tests by categorization, without opening up a can-of-worms on how to classify a test; we can do that later.

The simplest and easiest ways to get started with this is to organize your tests within PHPUnit, so you can run specific groups of tests along your deployment cycle.

For example, it's generally a good practice to run your true **unit** tests on every commit. If they're written correctly and concisely this shouldn't introduce a bottleneck on the developers.

Building on that, this is a simple recommended approach for which tests to run along your pipeline:

1. **Unit**
    1. On every `commit`
    2. After/during each `pull-request` being merged
2. **Integration**
    1. On every preprod build; usually I do this with a jenkins/ansible build in a containerized environment that can hit specific api endpoints.
3. **Functional**
    1. On every environment deployment. When deployed to `env` we'll run tests specific for that environment to verify behaviors for things like billing rules, email filters, etc.
4. **UI**
    1. After deployment; when deployed and "launched" we can run automated UI tests to verify behaviors and UI/UX interactions.

2. These are typically lower importance/severity, so they can generally happen post-build.