Fast and Efficient Development for Small Teams

Mike Waud and Eric Palakovich Carr

LEGACY | The Schroeder Institute FOR TOBACCO RESEARCH AND POLICY STUDIES

TASKS

Tickets / Tasks / Stories

These are the backbone of any good process, and are key to being fast and efficient. A poorly made ticket slows everything down the further it goes down the pipeline. Time invested here decreases time spent in all other parts of the pipeline.

Tickets should be created by the Product Owner to express the business need. These are then vetted by the project manager, and finally discussed and refined with the development team to hash out the details.

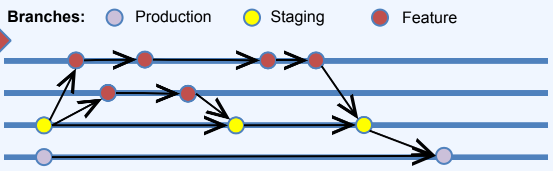
Follow these ticket creation guidelines:

⌘ Include complete acceptance criteria
⌘ Include testable acceptance criteria
⌘ Focus on "what", not "how"
⌘ **Smaller = faster & efficient**
⌘ Don't be afraid to break a large ticket into smaller tickets

Development

A single, small, well defined ticket frames this step. This allows the developer to focus on the "how" of implementing the ticket.

Repo Strategy
Your source repository for a project should at least have a production and staging branch, as well as a feature branch for each ticket.

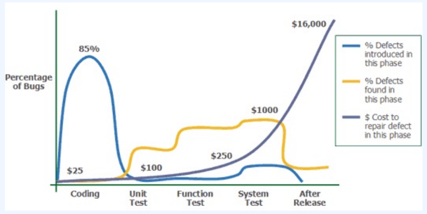


Out-source Tedious (a.k.a. slow) Tasks to Python

⌘ Data Migrations – South, Alembic
⌘ Deployment – Fabric, collectstatic (Django), django-storages
⌘ Testing – unittest, mock, nose, factory_boy
⌘ Package management – virtualenv and pip

QA / Testing / Reviews

Finding bugs early is ideal, so QA is almost every member's responsibility. Multiple QA steps throughout means less **money**, **time,** and **firefighting** spent fixing bugs after a release.



Automated Tests
Since most bugs are introduced during development, automated tests are invaluable at catching bugs early while they're still cheap to fix. Plus, **automation = fast & efficient**.

Feature Tests
The acceptance criteria on a ticket are the basis for these tests. These should be performed by manual feature tests done by someone other than the developer.

Feature tests should be done in isolation from all other features not yet on staging. We use "Feature Boxes", which act like staging servers for a single feature and have parity with production.

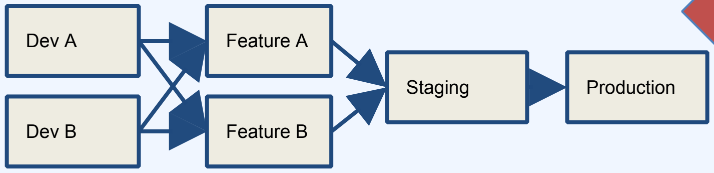
Integration / System tests
These are performed before a release, and once again should be performed by a combination of automated and/or manual tests. This should cover all new tickets in a release.

Code Reviews
Code review's biggest strength is finding design weaknesses. **Every** ticket should have one, and the bigger the ticket the earlier a code review should happen. We use pull requests in Github. **Github rocks for code reviews!**

Deployment

We deploy multiple times, first to a feature server and then later to staging, putting the new code through its paces each time. This gives us confidence that the final deploy to production won't lead to any firefighting. Remember, **firefighting != fast & efficient**.

Pre-production Environments
Dev – The developers local machine.
Feature – Used for testing a single ticket.
Staging – Used for testing all tickets in a release.



Monitoring & Reporting

Never let a user find a bug before you do. If you want to avoid late night firefighting sessions, use services that monitor and report when your systems aren't healthy.

Monitoring:

⌘ Use logging judiciously
⌘ Pingdom for system health
⌘ New Relic or Munin for performance monitoring

Reporting:

⌘ Data integrity
⌘ Key Performance Indicators (KPI) for all servers
⌘ Errors

Roles

Product Owner

Manages the vision for the project, and creates the initial form of a ticket.

Project Manager

Works with the team to refine tickets, and watches them pass through the entire pipeline. Poor performance here != fast & efficient.

Developer

Creates the software and automated tests that satisfy a ticket. Code reviews software from other developers.

Tester

Runs automated and manual tests at various stages to ensure quality.

Release Manager

Documents and releases new features to production.