Development and unit testing

Executive summary

- Effective unit testing is achieved by a use case "Definition of Done."
- Pega has tools built in to make unit testing easier, but these are only useful if you have the discipline to use them.
- UX/UI, performance, and security hardening are not afterthoughts – they have to be designed and tested at the unit level.
- Quality is driven by constant focus on the cause of defects.
- Unit testing is not optional. It is the fastest, cheapest way to find defects. Poorly conducted testing causes major downstream issues.
- Automate unit testing with Pega® Automated Unit Testing.

Use case Definition of Done (DoD)

Many studies have shown that the sooner defects are found in the development process, the easier and cheaper they are to fix. The cheapest time to fix a defect is during specification and design, the second cheapest time is during unit testing. Therefore, it makes sense to have a formal process to build in quality checks before we start to build and to have a strong unit test that takes advantage of Pega's built in capabilities.

This does not happen by accident – all projects start with the intention of conducting great testing, but testing is often the first thing reduced when there is time pressure.

Reducing developer testing does not save time or effort. In many cases, it transfers time from a first delivery to a test phase, which significantly increases the overall effort. This is the IT equivalent of not spending money on foundations so you can build a taller tower – ultimately it will all come crashing down.

To ensure we conduct good testing, we need a formal DoD – basically a checklist for each use case that establishes what must be done to be considered finished. This DoD can be tweaked for each project (some Business Rules Engine (BRE) projects may not have UI for example), but the majority of the DoD will be constant across projects.

Each use case should have a DoD. But you may decide to merge some use cases together if it makes sense to build and test them as a single unit. This is quite rare, as it tends to indicate the use case boundary is set incorrectly.
The DoD includes reviews from the LSA, BA, client SME, product owner, and potentially testers. As each step is completed, the percentage complete is based on the number of completed steps. For example, seven out of 20 steps completed equals 35 percent completion. This avoids the "80 percent complete syndrome," where a piece of work becomes 80 percent complete very quickly and then stays that way for a long time.

If extra time or effort is needed to complete a step, it is better to have it take the time during an early stage instead of waiting until the end and having poor quality delivery.

Unit testing coverage target

The format of the unit test template is less important that the fact that it is done. Testing should aim to ensure that for every screen-based system:

- Fields are filled in with both valid and invalid values.
- Boundaries and limits are tested for numerical fields.
- Interface and connectors are checked at minimum to a dummy interface and preferably to a real working interface.
- Both positive and negative routes are tested. (i.e. What happens if we do the wrong thing?)
- All buttons and links work.
- All dropdown values are tested.
- All drop down values drive processing, regardless of source.
- All values are hardcoded inside Pega.
- There are sample values from an external database.
- PegaRULES logs are reviewed for potential errors. (Note that PegaRULES logs are different than Alert and Security logs.)

Unit testing format

See appendices for standard format; however, it is common to use a clients preferred UT format. This can be acceptable as long as the format doesn't introduce unnecessary overhead.

Guardrails/alert logs/PAL traces

Pega includes several tools to assist with unit testing.

- PegaUNIT
- Guardrails
- PAL (Performance Analyzer)
- Alert logs

- Security Logs – This will identify issues like XSS, violation of CSP, etc. PDN reference: https://pdn.pega.com/performance-alerts-security-alerts-and-autonomic-event-services/performance-alerts-security-alerts

Test data and interfaces

If a system involves any form of cross-system data synchronization, request a common data set across the development and testing environments. For example, you might request 100 production records depersonalized or a similar set of synthetic data, as appropriate.

For simple case-based management, you do not need a common data set for development and testing because ad hoc data can be made up by developers as needed.

Unit testing for defect fixes

Any defects raised should be triaged and assigned to the developer who worked on the use case as per the defect process. This may be a multi-step triage process, depending on project phases and size. For example, a BA may review the defect to determine if it is a valid defect or a requirement change, whereas a technical lead may triage to assign a fix.

If a defect fix fails retest, it requires more investigation. Establish a regular (daily or bi-weekly) defect review meeting that includes the LSA, BA, EL, and developers to review the root cause of the failure.

Often the root cause is not poor development or testing; it's usually bad test data, invalid tests, system outages, etc. These are all worth investigating to avoid the perception that there is poor development quality. Actions must be taken to avoid these problems from happening again. However, if the failure is from something we could have trapped during unit testing, there should be an investigation to see what can be done differently next time.

This meeting can initially be quite painful but, by focusing regularly on quality, it will soon become shorter and easier.

It can be difficult to get the retest fail rate to zero, as there are usually some constraints on unit testing due to data, systems availability, etc. – but aim to drive this below five percent of defects.

PegaUnit

PegaUnit allows simple unit tests to be automated. Unit tests can be chained together to provide a level of test coverage.

This is not a replacement for a full automated test script like Selenium, but it can be done at an earlier stage in the project lifecycle.

Development and unit testing

Checklist for success

- Agree on the format of DoD.
- Ensure everyone understands their roles in the DoD.
- Establish a defect triage process.
- Establish a code review process.
- Establish a tracking mechanism to report on DoD.
- Establish a playback and review schedule around the DoD.
- Monitor DoD completion daily to see if the steps are followed.

Obtain written approval for any exceptions to the DoD. If things are eliminated there will be downstream implications.