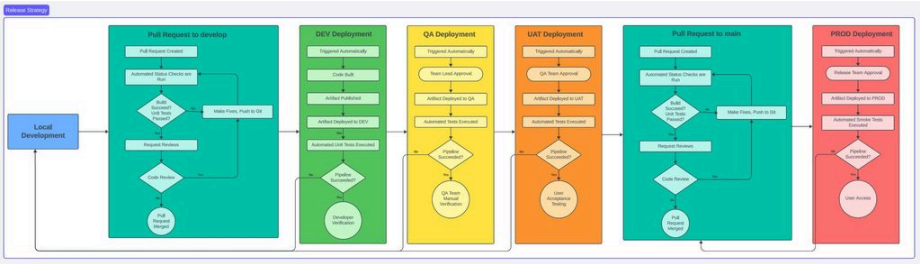
Complete Development Cycle

Full Cycle



1. Developer Machine (Local Development)

- **Accountable:** Developer
- **Responsible:** Developer
- **Consulted:** Product Owner, Team Lead

When starting a new development, this is the expected steps to follow locally:

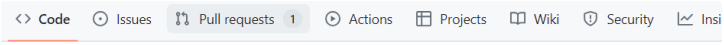
1. Open Git Bash or Command Prompt.
2. If this is the first time accessing this repo, change directory to your `repos` folder and clone the repo:
   `git clone <repo-url>`
3. Change directory to repo root folder.
4. Switch to `develop` branch:
   `git checkout develop`
5. Update to latest:
   `git pull origin develop`
6. Create feature branch from develop locally:
   `git checkout -b <user-alias>/<workitem>`
7. Build new code and tests.
8. Test the code locally.
9. Register all the code changes with Git:
   `git add -A`
10. Commit the code changes:
    `git commit -m "<message>"`
11. Push to Git Remote
    `git push origin <user-alias>/<workitem>`
    NOTE: Developers are encouraged to run Steps 6 to 8 regularly.

2. Pull Request Creation to `develop`

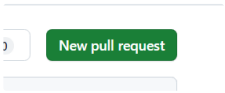
- **Accountable:** Developer
- **Responsible:** Developer
- **Consulted:** Development Team
- **Informed:** Product Owner, Team Lead

Once development is complete, the developer should go to Github website.

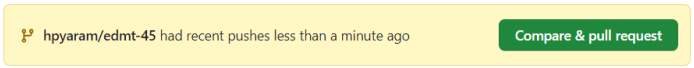
1. Go to the repo URL.
2. Switch to Pull Requests page:



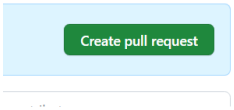
3. Click on `New pull request` button on top right:



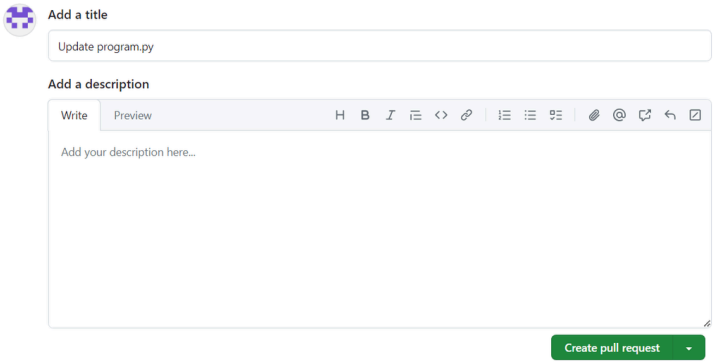
NOTE: Alternatively, if you see a pop up similar to below, click on `Compare & pull request` and skip to step 6.



4. Select `base` branch as `develop` and `compare` branch as the branch defined in the previous section. (`<user-alias>/<work-item>`)
5. Review all code changes and then click on `Create pull request`.



6. Enter a title for the change, this will be used to identify the changes so try to make it succint and unique.



In the description section, describe all the changes and their impact. This can also be used to define what tests were run and how someone else can verify the functionality.

7. Click on `Create pull request` .

3. Pull Request Stage

- **Accountable:** Developer
- **Responsible:** Product Owner
- **Consulted:** Development Team
- **Informed:** Team Lead

After the Pull Request is created, it's on the developer to request reviews from all the relevant stakeholders.

In the meantime, automation in the Pull Requests should be running the following:

- Building the merged code
- Running Unit Tests
- Running any other Status checks programmed for the branch.

NOTE: If there are any merge conflicts, the automation will not be triggered until a new commit resolving the conflicts has been pushed. Also, the automation will reset and rerun every time a new commit is pushed to the source branch.

The Pull Request will be ready to merge if and only if all the following conditions have been met:

- All Status Checks have passed.
- There is at least one reviewer approving the Pull Request for merge.
- All required approvers or their representatives have approved the request.
- No reviewer has blocked the Pull Request from merging.
- All conversations on the Pull Request have been resolved.

When the Pull Request is ready for merge, the developer has to go in and click on `Merge`. Alternatively, if `Auto-merge` has been turned on, the Pull Request will merge automatically. (This option needs to be turned on by the developer for each Pull Request created.)

4. DEV Deployment

- **Accountable:** Product Owner
- **Responsible:** Developer
- **Consulted:** Development Team

- **Informed:** Team Lead

Whenever the PR merges into `develop`, a Github Actions Workflow (pipeline) deploying to DEV environment is triggered automatically.

This pipeline is expected to accomplish the following:

- Build the code.
- Run Unit tests.
- Deploy it to the DEV environment.
- Run automated smoke or regression tests.

Once this deployment completes, the developer has to verify the functionality on the environment manually and then inform their Team Lead and the Product Owner if everything looks good.

5. QA Deployment

- **Accountable:** Product Owner
- **Responsible:** QA Team
- **Consulted:** Team Lead
- **Informed:** QA Lead

If the deployment to DEV completes without issues, the pipeline will automatically trigger a deployment to QA environment. The deployment will enter a “waiting” state requiring approval from someone in the Team Lead group.

When a Product Owner requests approval for a deployment to QA environment, it is the responsibility of the Team Lead to ensure all Governance rules and standards are being followed in the Pull Request. If everything looks good and the developer has verified functionality, the Team Lead can approve the deployment to QA environment.

Once approved, the pipeline will deploy the code to QA Environment and run any automated tests as configured by the QA team to ensure there’s no regression in the product. After deployment, the QA team can access the environment and manually verify that the functionality is working as expected. Should be verified with the Product Owner.

6. UAT Deployment

- **Accountable:** Product Owner
- **Responsible:** Developer
- **Consulted:** Development Team
- **Informed:** Product Owner, Team Lead

Once the QA Environment deployment completes, the pipeline will automatically trigger a deployment to UAT. This deployment will also enter a “waiting” state and will require an approval from someone in the QA team.

After the QA team has verified functionality and is confident in the code, they will have to approve the deployment to UAT.

NOTE: If there is high confidence in the automated tests written by the QA team in catching regression, this approval requirement can be removed, and the time-to-PROD can be reduced even more.

Once approved, the pipeline will deploy the code to UAT and run any automated user scenario tests to ensure any customer functionality has not been impacted. Any Load and Stress Testing can be executed at this state to catch any regression in performance.

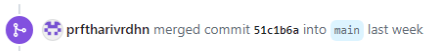
7. Pull Request to `main`

- **Accountable:** Product Owner
- **Responsible:** Developer
- **Consulted:** Release Team
- **Informed:** CAB Team, Team Lead

This is where the Product Owner goes back to the developer to request a PR to promote to `main`.

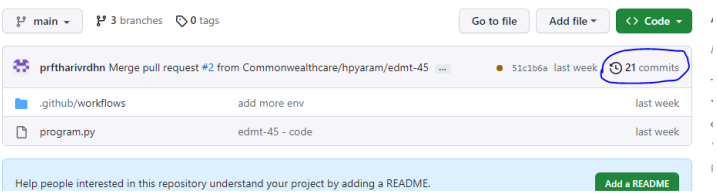
For the developer, we go back to local machine and follow the steps defined below:

1. Note the commit generated by the merged PR in Section 3 above:
   The final message in the PR would look similar to this:

   If this works, skip to step 3.

   Alternatively, you can go to the `develop` branch on Git and click on the history to look at the commits:

2. In the list of commits, look for the one merging the PR and copy the commit code beside it:

3. Open Git Bash or Command Prompt.
4. Change directory to repo root folder.
5. Switch to `develop` branch:
   `git checkout develop`
6. Update to latest:
   `git pull origin develop`
7. Switch to `main` branch:
   `git checkout main`
8. Update to latest:
   `git pull origin main`
9. Create feature branch from `main` locally:
   `git checkout -b release/<workitem>`
10. Cherry-pick the commit into `main` using the commit-id from Steps 1 & 2:
    `git cherry-pick <commit-id>`
11. Push to Git Remote
    `git push origin release/<workitem>`
12. Create a Pull Request using the steps from Section 2 with the base branch as `main`.

At this point, the Product Owner has to present the change to the CAB meeting and get them to approve the promotion to PROD.

8. PROD Deployment

- **Accountable:** Product Owner
- **Responsible:** Release Team
- **Consulted:** Team Lead
- **Informed:** CAB Team

The PROD deployment is triggered automatically after the PR is merged into `main`. The deployment will require the approval from a Release Team before the deployment actually promotes the code to PROD.

This approval requirement should not be removed. There needs to be at least one manual approval required before the code is promoted to PROD.

Automated Smoke Tests are run after the deployment is completed.