Overview

1. Overview

The Modern Development Process (MDP) is a Software Development Process[1] captured from historical industry practices and evolving modern approaches. It's maintained as change is encountered and experienced in the IT industry.

Good "set the stage" references:

- What a modern development organization looks like[2]
- On Modern Software Development[3]

Notes:

1. The first reference covers the **current state of many organizations concepts and tool sets**. (5 minute read)
2. The second reference reviews **past and current approaches** of **A Software Developers Job Description**. (5 minute read)
3. These references set the stage for this MDP document.

The purpose of the MDP is to encourage and support developers by:

- Separating Development from Operations
- Identifying Automation separately from Development and Operations
- Encourage an MDP approach; more passive programming, less active programming

The goals of this process include:

- Get more value to the market quicker
- Reduce risk by addressing security from the start

[1] https://en.wikipedia.org/wiki/Software_development_process
[2] https://www.infoworld.com/article/3230905/application-development/what-a-modern-development-organization-looks-like.html
[3] https://www.rainerhahnekamp.com/en/modern-software-development/

- Empower developers with ability to consider and present all possibilities
- Promote an environment where developers can support any group within or outside the organization
- Identify, develop, and maintain “Best Practices”

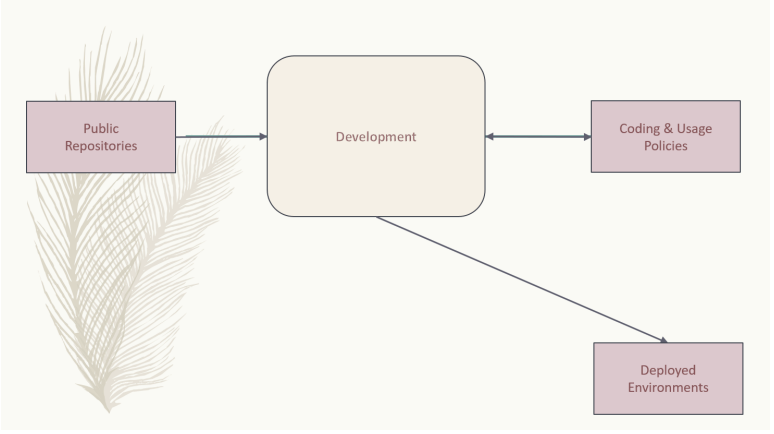
The primary workflow utilized by the MDP is the GitFlow Workflow[4].

The GitFork Workflow[5], which is built on the GitFlow workflow, is an additional development tool to:

1. Promote developer contributions to outside open source projects or internal "other organization" projects
2. Provide a backup capability to the contributor workstations
3. Provides a safe environment for new-contributors to experiment with before acquiring "solution/code review" access
4. Promoted by default configurations for GitHub and GitLab, primary examples of git bare repository management tools.

- This approach has been defined and exercised in the open source development world.

2. Context Diagram



[4] https://datasift.github.io/gitflow/IntroducingGitFlow.html
[5] https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow

2.1. Develop Changes

Developers are focused on development separately from Operations and CI[6] / CD[7] automation. They work directly with **public repositories** and local code identifying best options for development requirements or “asks”.

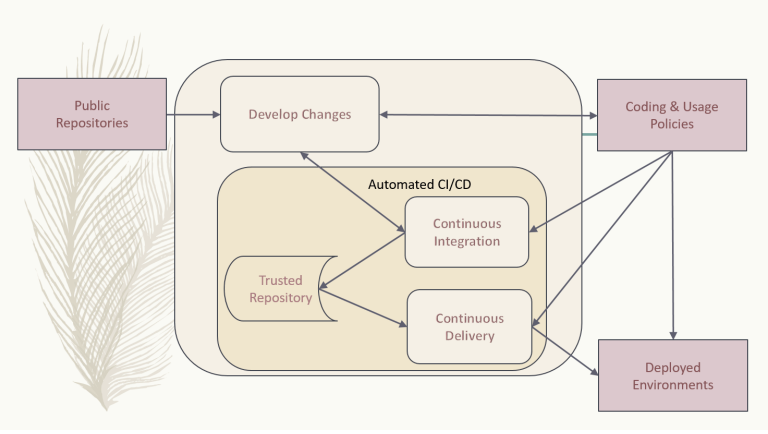
During this DevOps process, development references and challenges **Coding & Usage Policies** per security, licensing, testing, and coding standards.

Development proceeds unencumbered with private, individual developer controlled forked repositories, separate from the automation of CI/CD and stability required for operations.

Developers are free and encouraged to expand and improve CI/CD automation. This includes building, testing, and operational environment challenges as they develop new functionality for customer solutions.

Code Reviews occur with Development Leads prior to initiating the Automated CI/CD.

3. DevOps Process



[6] https://en.wikipedia.org/wiki/Continuous_integration
[7] https://en.wikipedia.org/wiki/Continuous_delivery

NOTES:

- For the GitFork workflow[8], Dev coding, testing, verifying, etc, happens between step 7 and 8.
- **Rebuild project/repository OFTEN. All repos including workareas are:**
  - less stale
  - Up to date
  - More secure
- Project or repositories that are used by automated **Continuous Integration** are configured to promote code reviews and clean development baseline starting points. These are the repositories developers fork from.
- Developer local (workstation) repos can utilize CI and CD together to promote to a developer controlled environment. These type of pipelines are limited to a maximum set of versions in the **Trusted Repository** as they support development. Otherwise non-development pipelines end at the **Trusted Repository**.

3.1. Continuous Itegration

An automated process that is initiated at the completion of a **git push** to the primary (non-forked) GHE repository **develop** branch.

The build of this process enforces and verifys the **Coding & Usage Policies**.

Provides tested deployment sets to the Trusted Repository:

- All assembly, code building, and know tests, are executed for every build prior to adding to Trusted Repo. *
- Successful build identified with semantic versioning[9].
- Unsuccessful build is reported back to the submitter

[8] https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow
[9] https://semver.org/

NOTES:

1. Input changes to this process are developed and tested in the **Develop Changes** process, subject to code reviews.
2. For the GitFork workflow a pull request initiates this
3. For the GitFlow workflow, a feature branch push initiates this
4. This allows deployment environments, promoted to by **Continuous Deliver** to be focused on validation of new changes.

3.2. Trusted Repository

This is the end point of the **Continuous Integration** process and the starting point of the **Continuous Delivery** process.

In addition to updates made by the **Continuous Integration** process, **Coding & Usage Policies** can report on and remove elements as required.

3.3. Continuous Delivery

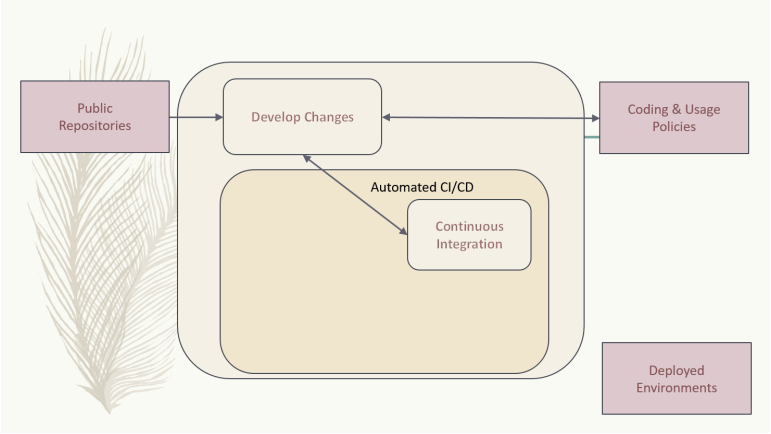
Provides deployments from the **Trusted Repository** for argument selected environments.

Coding & Usage Policies can report on and remove elements as required within this process.

NOTE:

- **Coding & Usage Policies** can report on and remove elements as required outside of the MDP DevOps Process.

4. DevOps Developer



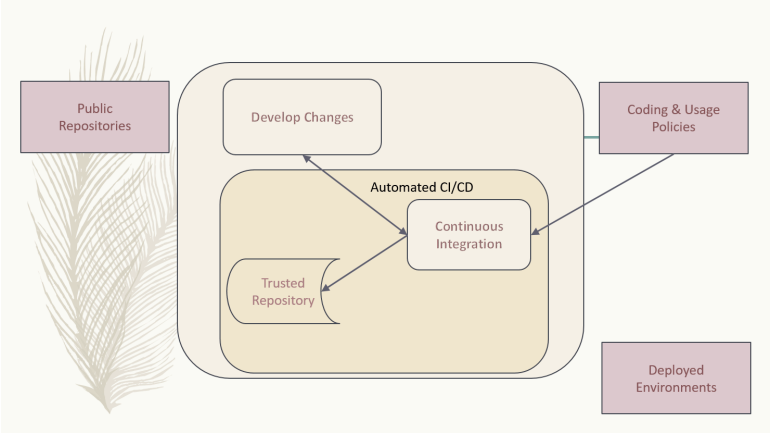
For given “asks”, developer looks for and/or develops working solutions

- Identify/Establish repository or repo:
  - From Public Repository or Team Repository or create new (Team repos available to CI process)
- For GitFlow workflow: **
- For GitFork workflow:
  - Fork from existing projects or repositories for development access to outside or "other group" repositories: (See reference section for related links)
  - Forking Projects – (example using the spoon-knife project),
  - Fork & Pull Workflow (For git beginners)
  - Clone from forked repo
  - Create additional remote to non-forked repo
  - All contributions are done with "Pull requests" rather than feature branches
  - Automation starts when devlead complete code review and pushes to non-forked repo development branch.
  - NOTE: Rebuild local repo often, daily to start
- Verify Coding & Usage Policies:

- Report variance to customers
- Challenge security, licensing, testing, and coding standards as needed

- Prepare Changes:
  - Test with all known CI process testing. Add, modify, remove as needed.
- Commit and merge per GitFork workflow (Dev coding, testing, verifying, etc, happens between step 7 and 8)
- Initiate automated CI
  - Push Changes to Non-Forked repository or project

5. Continuous Integration Automated Process



The Continuous Integration process is:

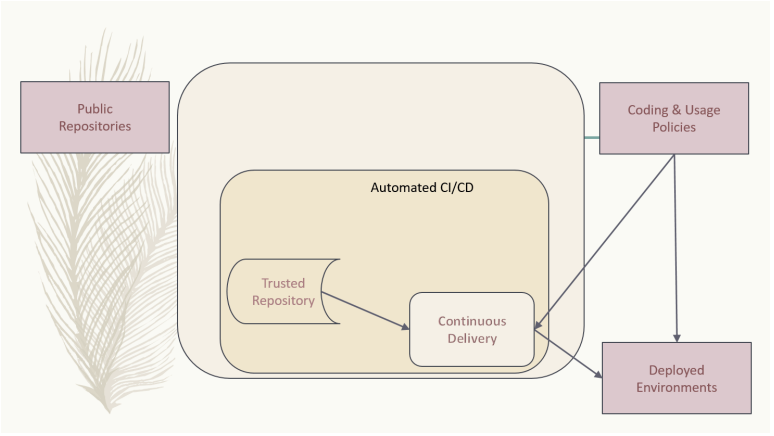
- Initiated by Developer with a repository branch push
- Automatically verified with existing “Coding & Usage Policies”
- Automatic Build
- Automatic Testing
- Build/Test reporting to development
- Successful build identified with semantic versioning [10]

[10] https://semver.org/

- Successful build delivered to Trusted Repository

NOTE:
Coding & Usage Policies verification reporting is run against all elements in Trusted Repository on a regular basis.

6. Continuous Delivery Automated Process



The Continuous Delivery process:

- Starts with up-to-date, verified, built from a Trusted Repository
- Contents included are delivered from CI automated process exclusively
- Uses Builds identified with semantic versioning[11]
- Automatically verified with existing “Coding & Usage Policies”
- Automatic environment deployment report
- Automatically deploys to specified environment

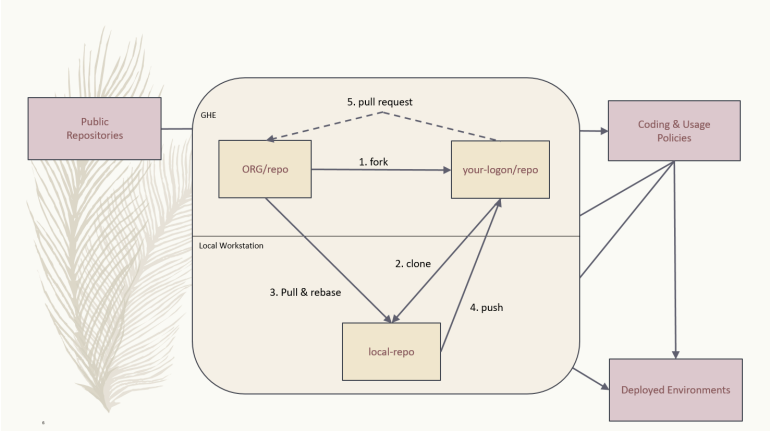
NOTE:
Coding & Usage Policies verification reporting is run against all "deployed to" environments on a regular basis.

[11] https://semver.org/

7. GitFork Related Steps

The GitFork workflow is an additional tool available to development when needed for the reasons mentioned in the Overview section.

The following image identifies the steps specific to the GitFork workflow.



Important:

- Just like the GitFlow workflow, the ORG/repo is the starting point of automated Continuous Integration. For the GitFlow workflow, the feature branch is the delivery element.
- Code reviews are initiated by contributing developers when they initiate "pull requests". These are managed by DevLeads. These are the gateway to controlled GitHub repository updates and non-development CI initiation.
- The use of forked repos only requires ready-only permission for contributing developers.
  - Executing the Install git[12] procedure is required for this.
- The Develop Changes process is separate from:

[12] https://github.com/cmguy/CM-Plan-Site/blob/develop/app/site/git/GitInstallationProcedure.adoc

    - Automation:
        - CI
        - CD
    - Operations

8. References

- Fork & Pull Workflow[13] For git beginners
- Forking Projects[14] – example using the spoon-knife project[15]
- GitHub and Git Foundations (YouTube)[16]
- GitHub vrs GitLab[17]

[13] https://reflectoring.io/github-fork-and-pull/
[14] https://guides.github.com/activities/forking/
[15] https://github.com/octocat/Spoon-Knife
[16] https://www.youtube.com/playlist?list=PL0lo9MOBetEHhfG9vJzVCTiDYcbhAiEqL
[17] https://usersnap.com/blog/gitlab-github/