**Gitflow**

Gitflow это один из вариантов реализации системы контроля версий в команде разработки. Главная идея в том, что основных веток несколько, а остальные являются функциональными. То есть, при разработке проекта, готовые изменения, изначально заливаются в ветку разработки(develop), после этого в ветку релизов, и после этого в главную ветку (main/master). В главной ветке находится то, что непосредственно загружается в прод. Для загрузки изменений в develop, работа изначально идет в ветке задачи(feature), и только после код ревью, develop обновляется. Когда количество новшеств накапливается и принимается решение выпуска релиза, то создается дополнительная ветка(релиз). Данная ветка обычно тестируется тестировщиками и проходит окончательную проверку релиза. После этого, ветка релизов вливается как в main, так и в Develop, для учета всех изменений. В случае если баг найден уже при эксплуатации в проде, то создается отдельная ветка фиксирования бага и её результат заливается как в основную ветку, так и в ветку разработки.

**User Flow**

Данная методология позволяет спроектировать удобный сценарий взаимодействия с продуктом. Юзер-флоу показывает точки входа в сценарий, все переходы и страницы на пути к достижению пользователем его цели. Используется он при проектировании интерфейса, и соответственно при разработке, создание интерфейса будет отталкиваться как раз от него. User Flow делят на 4 части – старт, точка принятия решения, переходы, финал. При разработке User Flow изначально отталкиваются от user story, после создаются правила чтения схемы и в последствии сама схема.

**CI (stages)**

Непрерывная интеграция изначально была разработана как часть экстремального программирования, поэтому она нацелена по сути на постоянный обмен информацией между разработчиками. Если все часто отправляют изменения в основную ветку, то конфликты обнаруживаются на много быстрее. Если разработчики фиксируют изменения каждые несколько часов, конфликт можно обнаружить в течение нескольких часов с момента его возникновения, к этому моменту ничего особенного не происходит, и его легко разрешить. А конфликты, которые остаются незамеченными в течение недель, могут быть очень сложными для разрешения.

Если все в команде интегрируются по крайней мере ежедневно, то основная ветка остается в хорошем состоянии. Однако также необходимо убедиться, что каждый коммит проверен. Обычный способ сделать это — использовать службу непрерывной интеграции (CI Service) , которая отслеживает основную линию. (Примерами CI Services являются такие инструменты, как Jenkins, GitHub Actions, Circle CI и т. д.). Только после того, как эта сборка интеграции станет зеленой, можно считать интеграцию завершенной.

Весь смысл непрерывной интеграции заключается в предоставлении быстрой обратной связи. Поэтому важную роль в этом также играет быстрая сборка. Она обеспечивает проверку возможности реализации данных изменений. Так же, одной из самых важных вещей для сообщения является состояние основной сборки. У служб CI есть панели мониторинга, которые позволяют всем видеть состояние любых сборок, которые они запускают.

Непрерывная интеграция отличается тем, что она определяется как стиль с высокой частотой. Она делает акцент на установлении частоты интеграции как на цели сама по себе, а не на привязке ее к завершению функций или частоте выпуска.

**CD (deploy, update, rollback);**

Непрерывное развертывание— это автоматическая доставка обновления в прод. Преимущественно без участия человека и без согласования.

Типичный CD-конвейер состоит из этапов сборки, тестирования и развертывания. Более сложные конвейеры включают в себя следующие этапы:

Получение кода из системы контроля версий и выполнение сборки.

Настройка инфраструктуры, автоматизированной через подход “инфраструктура как код”.

Копирование кода в целевую среду.

Настройка переменных окружения для целевой среды.

Развертывание компонентов приложения (веб-серверы, API-сервисы, базы данных).

Выполнение дополнительных действий, таких как перезапуск сервисов или вызов сервисов, необходимых для работоспособности новых изменений.

Выполнение тестов и откат изменений окружения в случае провала тестов.

Логирование и отправка оповещений о состоянии поставки.

Данные этапы носят скорее общий характер и конвейер может состоять и из других блоков задач. Соответственно, частота релизов является приоритетом в конвейерах CD.

Для реализации данного вида развертывания должны быть хорошо настроены и полностью автоматизированы тесты, которые проверяют целостность каждого нового релиза перед развертыванием. Комплексное тестирование теоретически должно исключать регрессии в новых функциях после публикации. Однако необходимо также, используя соответствующие метрики для проверки успешности развертывания, настроить быстрые автоматизированные откаты. В этом случае есть возможность управления ошибками, которые не обнаруживаются и попадают в производство.

Безусловным плюсом является то, что конвейеры развертывания срабатывают автоматически при каждом внесении изменений, поэтому разработку можно вести быстрее.