BASE PATTERNS

SOURCE BRANCHING

Source branching is a system where we take a copy of the code base where we have to work and we work with that copy, which will be called branch. A branch is a series of commits and the last one is the head. When two commits are made in parallel a branch splits. When a change is made in other branch it can be merged into with some effort to solve the changes in parallel.

MAINLINE

The mainline is the current state of the team's code. Every new implementation starts from it and when it is finished it will be merged into it. Most of the patterns we will see here are based on mainline.

HEALTHY BRANCH

A healthy branch is the term we use when referring to a branch that has a successful build and few, if any, bugs. Keeping the health of a branch is almost compulsory as it eases identification of errors, increase team trust and productivity. A healthy branch must have solid Self Testing code in order to seek for bugs and problems that can arise.

INTEGRATION PATTERNS

MAINLINE INTEGRATION

Mainline integration is a reflection of the current software teams'. Using a mainline production is improved and easily coordinated and as his name suggest, is based on merging the branch of each teammate in the mainline when required.

FEATURE BRANCHING

Based on mainline integration, a branch is opened when starting a new feature and merged when finished. Two procedures can be followed when Feature Branching: Low-Frequency or High-Frequency integration.

INTEGRATION FREQUENCY

Integration Frequency it is about how frequently merging is done over a period of time. This topic is examined under 2 main headings called "Low-Frequency and High-Frequency". These titles are up to the choice of the teams

LOW FREQUENCY INTEGRATION

Meaning, to isolate himself/herself(team members) from the mainline is to start on his local and continue for a long time. Surely it will bring more complicated conflicts.

HIGH FREQUENCY INTEGRATION

With 'High Frequency' we increase the frequency of merges, this means more pushes but it will reduce any risk. Meaning if there is a conflict back there teams can find it quickly.

SELF TESTING CODE

Sometimes our code may have some issues and sometimes we aren't able to solve them. If there are situations like that self-testing code can probably handle these issues, it can quickly find bugs and issues for us.

CONTINUOUS INTEGRATION

Provides teams High Frequency, and it provides less work to deal with. Continuous Integration is about 'how frequently we will merge'. If teams want to use this topic they have to make integrations at least once a day if teams can manage it and if they have 'Self-Testing Code' they won't bother with issues and this means quick finishes. These are the advantages of Continuous Integration.

PULL REQUEST

If someone makes some changes to the mainline after pulling it and if he/she wants to merge that code with mainline. Her/his code needs to be checked and gets accepted otherwise the code needs to be fixed or something like that.

PRE-INTEGRATION REVIEW

Before pushing something to the mainline, the developer sends the commit to a review, where other members can check it and make some comments and changes until they decide the commit is good enough, and then, it will be placed on mainline.

It is popular for open source, where there is less control on the code because of the occasional contributors.

Principal problem -> **Integration friction** (activities that make integration take time or be an effort to do).

Modularity also impacts integration. Often teams need to use source branching because the lack of modularity leads to it.

THE PATH FROM MAINLINE TO PRODUCTION RELEASE

RELEASE BRANCH

Isolate code when ready to for a release on its own branch dedicated only to remove any remaining defect. Once this is done, it is ready for production.

There can be only one branch or more if several versions of the product are in use.

MATURITY BRANCH

Maturity branches provide a way of tracking the most up to date version of the source.

Once a version of a code base reaches a certain level of readiness, it's copied into the specific branch so it shows each version of the code that reaches a stage in the release workflow.

Variation: Long Lived Release Branch

It is a mixture of a release branch and a maturity branch. It is a reused release branch that is maintained instead of creating a new one for each release. This approach is only suitable for products with a single release in production at a time.

ENVIROMENT BRANCH

An environmental branch is a branch that contains commits that apply to the source code to reconfigure the product to run in a different environment.

HOTFIX BRANCH

If a serious bug appears in production, then it needs to be fixed as soon as possible. A branch is created to work on this bug with the highest priority.

RELEASE TRAIN

Releases are made on regular intervals of time. Metaphor of trains departing from a train station where the developers can choose what of the trains take to implement a given feature. Whenever a train releases (hot - freeze) the branch becomes a release branch, in which not new features can be added but only fixes, which will be cherry-picked by the new branch.

Variation: Loading feature trains.

In this scenario, several trains may accept changes in the features at the same time. This scenario is useful when a non-finished feature is pushed from one branch to the next one before the departing. Also, several teams can work on features from different trains at the same time.

RELEASE-READY MAINLINE

Simple alternative to the already described patterns where the mainline is used as the release branch, keeping into account it should be healthy enough.

OTHER PATERNS

Experimental Branch: Used to implement several experimental approaches to solve a given problem. The code would not be directly integrated to the mainline and it is highly possible to be abandoned.

Collaboration Branch: Useful to share parts of the work when code is not visible to other members of the team. The branch is temporal and may be pull and pushed from different collaborators repositories.

Team Integration Branch: Branch used by the sub-teams of the project as a mainline to integrate code before sending to the releasing branch. Especially useful when sub-groups have different integration frequency.