TRUNK BASED DEVELOPMENT

Adrián González Guadalupe, Carolina Barrios González,

Carlos Menéndez González, Gonzalo Suárez Losada

1. INTRODUCTION

A version control system tracks and provides control over changes to source code.

Trunk based development is a source control branching model where developers collaborate on code in a single branch called 'trunk'. Any other branch that may be created should live a very short life. This emphasizes continuous integration and delivery while removing the complexities associated with long-lived branched.

2. WHAT IS TBD?

In TBD, as previously said, there is only one branch, 'trunk' where developers commit small, incremental changes directly to it multiple times a day. This means that any minor change in the functionality should be commited and revised.

2.1. KEY PRINCIPLES:

- Continuous Integration: mainline is continuously updated with the latest changes and allows for early detection and resolution of integration issues.
- Small, frequent commits: any minimal change in the functionality should be committed right away. This means also faster review as the changes are not that big.
- Shared ownership: since everyone works on the same trunk, there's greater visibility and transparency in the development process.
- Feature toggles: decouple the deployment of new features from their activation. This allows teams to merge incomplete or experimental features into the mainline without affecting the production environment.
- Automated testing: it is necessary to ensure that each code change is thoroughly tested before integration into the mainline.

3. UNDERSTANDING TBD

3.1. BENEFITS OF TRUNK BASED DEVELOPMENT:

- Reduced Merge Conflicts. In the case of small teams, TBD reduces significantly the occurrence of merge conflicts compared to other branching strategies. This leads to faster development cycles.
- Faster Integration and Feedback Loops. Changes are continuously integrated into the trunk, allowing rapid feedback loops where developers can quickly assess the impact of changes and address any issues very fast.
- Improved Visibility and Collaboration. Since all changes are made directly to the trunk, there is greater visibility into the progress of the project which can facilitate communication.

- Promotes Code Quality and Stability. TBD encourages small, incremental changes and frequent integration, so it is very important to write clean, modular code so code can be tested easily and then this leads to finding issues faster and fewer bugs.

3.2. CHALLENGES OF TRUNK BASED DEVELOPMENT:

- Risk of breaking the build. Since in TBD the developers commit very frequently, there is a higher chance of introducing bugs or breaking the build. This is why automated tests are very important.
- Number of developers. If there amount of developers working on the same codebase is very high, it can be very challenging. It is important that the team understands TBD and also there must be good communication.
- Difficult to roll back changes. This is because each change is made directly to the trunk and there are no separated branches for holding previous versions of code. It is difficult to identify which change caused the problem and to rollback that specific change without affecting other work in progress.

3.3. COMPARISON WITH OTHER BRANCHING STRATEGIES:

In fact, apart from Trunk Based Development, there are a lot of branching strategies and one of them is Feature Branching.

Feature branching is a branching strategy where developers work on isolated branches for each new feature or task they are working on. Whenever a new feature is planned, a new branch is created from the main branch and then the feature is developed until complete and then merged back to the main branch.

Comparing it to Trunk Based Development, the differences can be found in the integration frequency (more frequency in TBD than in Feature branching), visibility and collaboration (developers in TBD see the progress of the rest of the team but in feature branching they work independently without interfering with each other’s work) and parallel development (feature branching allows it on multiple feature since each feature has its own branch while in TBD everyone works in the same codebase and this requires collaboration for multiple developers working on closely related features.

4. RECOMMENDATIONS

4.1. PROFILES NEEDED FOR TBD:

Using TBD is not hard to implement, although it is important to use it properly. That is why both, programmers and software architects, must have certain abilities.

- Regarding programmers, they must be experienced in many different projects. This leads to seniors being the most desired ones as they are the ones who will take more advantage of the flexibility of TBD.
- On the other hand, we have software architects. They must have a clear abstract idea of those concepts to be implemented by programmers, in addition to knowing how to tell them. Last but not least, it is extremely important for them to trust their partners and the code they develop to be fully functional.

4.2. HOW TO ADD VALUE WITH TBD:

By nature, TBD is usually used with those methodologies which match agile developments. For example, extreme programming as it can work with extreme short delivery times. It also provides flexibility to teams so they can work improving their productivity.

It is also worth mentioning pair programming too as it reduces the risk of introducing bugs.

4.3. MORE RECOMMENDATIONS FOR IMPLEMENTING TBD:

It is advisable to be fully prepared to deliver code in any moment of the development, which is archieved by organizing those tasks which will be implemented but not ready yet.

5. CONCLUSIONS

5.1. IS IT WORTH IT?

There is no such thing as the perfect strategy, it will depend on your team and the nature of the project you are developing.

This branching strategy is not for junior developers. It demands a high autonomy that we still have not achieved as students, and so we should aim for another option that helps monitor our work more efficiently, such as GitFlow, where our work is isolated in a single feature branch.

However, there will be someday in which Trunk Based Development will be interesting for us and we should consider implementing it. It is one of the two most common branching strategies together with GitFlow and not for no reason: it is very agile.

5.2. WHEN SHOULD WE USE TRUNK BASED DEVELOPMENT?

Most cases where we should apply TBD are those in which we need to do continuous integration and continuous delivery of our projects. That is the reason you should use this strategy in DevOps environments.

DevOps is not the only scenario where you can use this branching strategy, but it is probably the most common one. In general, as said before, there is no perfect branching strategy. You should choose the one that fits your project. And if your project needs fast and continuous delivery and integration, then Trunk Based Development is your strategy.

6. WEBGRAPHY:

Trunk-based Development | Atlassian

Git Branching Strategies: GitFlow, Github Flow, Trunk Based... (abtasty.com)

¿Qué es DevOps? - Explicación de los modelos de DevOps - Amazon Web Services (AWS)

Trunk Based Development

Baeldung - Trunk-Based Development