Branching strategies for developing new features within the context of Continuous Delivery

Konrad Schneid[1]

Abstract: This paper evaluates based on current literature, whether the versioning strategies "branch by feature" and "develop on mainline" can be used for developing new software features in connection with Continuous Delivery. The strategies will be introduced and possible applications for Continuous Delivery will be demonstrated and rated. A solution recommendation is finally given. It becomes evident that develop on mainline is the more recommendable method in form of "features toggles" or in case of bigger changes in form of "branch by abstraction" within the context of Continuous Delivery.

Keywords: Continuous Delivery, Branch by Feature, Develop on Mainline, Feature Toggles, Branch by Abstraction

1 Problem formulation

Different attempts exist around developing new software features. In practice, several developers are involved in programming software. They work in parallel on developing the software. In order to perform the parallel development of different software versions, version control systems (VCS) like Git are used [CS14, p.27]. VCS offer various versioning strategies for developing new features. The development of different features should be separated to provide these independently to the production system. It must be guaranteed that features which are still in development do not disturb the productive operation, but can still be developed. Two appropriate strategies are branch by feature and develop on mainline [HF10, Chap.14]. This paper researches whether and how these strategies can be used within the context of Continuous Delivery (CD).

One important aspect of CD is Continuous Integration (CI) [HF10, p.24]. The goal of CI is to improve the software quality by integrating every change of code. This means that software changes have to be continually integrated, tested and built. The central question which will be answered in the following, is how this aspect can be guaranteed during developing new features. Both strategies, branch by feature and develop on mainline, will be seen from this angle in the following chapters. According to RODRÍGUEZ ET AL. both methods are used in practice. But there is no recommendation which strategy is more suitable. This question also shall be answered in the context of this paper. CHEN also sees a need for further research in the area of software development within the CD process. [Ch15, p. 54; Ro16].

[1] Fachhochschule Münster, Münster, Germany, konrad.schneid@fh-muenster.de

2 Branch by Feature

This chapter explains the strategy branch by feature and evaluates the possible application for CD.

In VCS there is one mainline, also known as trunk or master. Branching means to deviate from the mainline and to create a new branch. Branch by features means, that every new feature has to be developed in an own branch. As soon as the implementation of the feature is finished, it will be integrated into the mainline (see figure 1). The mainline can be held in a release status. Thus branches allow an isolated development of new features. The productive environment is not interrupted because only the mainline is live. [HF10, p.410; PS13, p.135]

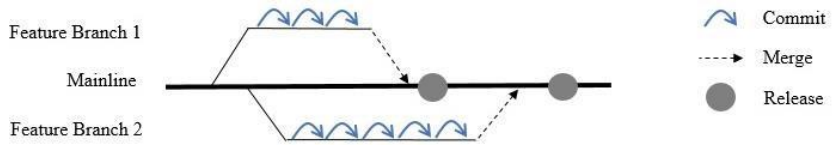

Fig. 1: Schematic diagram of branch by feature

2.1 Impact for Continuous Delivery

As described in chapter 1, CI is one important aspect of CD. Feature branches allow an isolated development. This is a problem if this strategy is used for CD. CI cannot be ensured because there is no integration of the individual branches. For this reason, this strategy is the wrong approach according to HUMBLE & FARLEY: "[…] branching by feature is really the antithesis of continuous integration […]" [HF10, p.412]. [Wo15]

If the definition of CI is not interpreted as strictly as defined by Fowler, the strategy for CD could be possible. Let's assume that a weekly integration is sufficient for a project. This is the case if a company which has used the strategy branch by feature till now would like to implement CD. Thus the developers are not completely derailed from their usual workflow and a higher acceptance can be created for CI. CI only works if the developers let themselves in for it and accept it. [HF10, p.57; Fo06].

In order to use the advantages of CD, the strategy must be supplemented with some rules. Feature branches should be short-lived and exist for at most a few days. As a consequence, the so-called "merging hell" is reduced. This arises if many long-lived feature branches exist and are integrated only after weeks or even months into the mainline. That means merging gets extremely complex and risky because the source code of the branches and the mainline strongly diverge. A further aspect to reducing the "merging hell" is to transfer every change of the mainline promptly to each branch. The number of merging conflicts can be minimized. For this reason, refactorings should also be transferred immediately to the mainline. Thus the number of feature branches should not exceed the number of the features to be developed and a new feature branch should

only be created if a feature was implemented successfully into the mainline. A merge from a feature branch into the mainline is only allowed if the feature branch was tested successfully before. [HF10, S.410]

This procedure can be used as the first step if CD should be introduced and the strategy branch by feature should be maintained although it is not fully suitable for CD. The following table shows the pros and cons of the strategy branch by feature.

Pros	Cons
Features in development do not cause problems during running time	Short-lived feature branches often not possible
Traceability of feature development	Merging effort
Each feature is independently in own branches	Integration delay

Table 1: Advantages and disadvantages of branch by feature

3 Develop on Mainline

In the following the strategy develop on mainline will be introduced and evaluated in the context of CD.

With this strategy the features are implemented directly on the mainline (see figure 2). Therefore, all changes are immediately available for all developers. Branches do not exist during development. Consequently, there are no merging problems. Since all developers work in parallel on the mainline, additional strategies are needed to develop the feature programs on the mainline in parallel. It must be guaranteed that the features, which are in development, still do not disrupt the productive operations. To develop directly on the mainline there are two approaches feature Toggles and branch by abstraction. These are described in the two following subchapters. [HF10, p. 405]

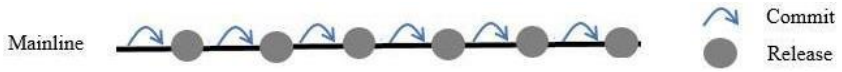

Fig. 2: Schematic diagram of develop on mainline

3.1 Feature Toggles

Feature toggle (synonymous feature flags, flipper, feature switches) is a programming technique that a feature in development can be turned on and off during the running time of the software. Toggles can only be in the state “on” or “off”. The software uses toggles during the running time for the evaluation and for the application and activates the feature or deactivates it depending on their condition. In case a new feature causes any problems, it can be switched off directly during running time. The simplest implementation variant of feature toggle is the use of if-then-else-statements. HODGSON differs toggles in four categories which are subdivided by the factors

lifetime and dynamics of a toggle. [Ho16; Eb15]

Release toggles allow to implement unfinished and untested source code directly on the mainline. These toggles must not be switched on during running time. The state of this toggle is typically static. According to HODGSON's recommendation release toggles should not exist longer than two weeks. HODGSON suggested to remove release toggles after the final acceptance of the new feature in the code to ensure manageability and prevent deactivating the feature inadvertently. [Ho16; Eb15]

Another application example of toggles is not just to hide unfinished code but also to use it depending on user groups or other environmental factors. This variant is known under the name experiment toggles. New features can be tested only with a small user group and released after a successful test stage for all users according to the "Dark Testing method" or using "A/B-Testing". The toggle condition is set dynamically and according to HODGSON has a lifetime of a few hours up to several weeks depending on the frequency of the use. Popular services using this method are Facebook and Flickr according to own information. [Ha09; Ho16; Ta15]

Permissioning toggles are very similar. With these toggles features are provided only for specific user groups. Selected beta or premium users can use features which are disabled for standard users. In contrast to experiment toggles the features are not randomly provided to users. They are explicitly turned on for specific users. The lifetime of premissioning toggles are appropriately long-lived. HODGSON himself has estimated the lifetime for several years. [Ho16]

EKART recommends a static configuration for features toggles. Thus toggles can be treated according to the "Configuration-as-Code" principle, which allows versioning and provides transparency. For extended configurations EKART recommends distributed "Key-Value Stores". [Ek16]

Feature toggles do not increase the complexity of testing according to NEELY & STOLT, although new and old features are in the same mainline. The number of test cases to be written is just as high as in case of using feature branches because of the combination difficulty of different features is the same. NEELY & STOLT go even further and argue that testing is easier with feature toggles than with feature branches: "[…] the combinatorial criticism would be the same problem with feature branches, and with feature toggles it is simpler as you toggle on and off in test code" [NS13, p.124]. The only additional effort is to specify which toggles can be turned on simultaneously. [NS13, p.124]

On the contrary, BIRD expresses the following criticism on feature toggles: "Feature flags make the code more fragile and brittle, harder to test, harder to understand and maintain, harder to support, and less secure." [Bi14]. According to BIRD it is dangerous to put untested code with uncertain effects in production. As an example he mentions an unintentional change in the business process of a financial institution. [Bi14]

TANG ET AL describe the successful application of feature toggles at the social

network Facebook. Through an administration interface (Gatekeeper) features can be adaptably released and selectively to certain user groups ("cohorts"). First, a new feature can be released with the Gatekeeper for one percent of the internal employees. The percentage is continuously increased with trouble-free use. Only after a successful internal test the feature is released for about five percent of the users of a specific region. In case this phase is also successful, the new feature is eventually released in further steps continuously worldwide. [Ta15]

3.2 Branch by Abstraction

Another possibility to run develop on mainline is the use of the pattern branch by abstraction defined by HAMMAT. In comparison to feature toggles this pattern is used for bigger changes, when it is not possible to implement these in small incremental steps. [Ha07]

With this pattern an abstraction layer is put on the part of the software, that should be changed. The abstraction layer points at the old implementation and allows a parallel development of the new feature. It can be set during the deployment or running time to which code the abstraction layer refers. As soon as the new development is finished, the abstraction layer refers to the new code and will be removed together with the old code if no problems appear. The CI principle is guaranteed. Only if the user has to choose the implementation, the abstraction layer will not be removed. According to HUMBLE & FARLEY the pattern is also used to transfer a monolith code base into modular built up software. In that case the old implementation is running parallel to the new, modular built code with the same functionality. [HF10, p.351]

To test the functionality of the new implementation, the pattern “verify branch by abstraction” can be used. In that case a toggle follows the abstraction layer and can refer to the new implementation (see figure 3). Both implementations are used with the same input data. The test fails if the result is not the same. This prevents that the business behavior of an application changes with the new implementation. This strategy is only possible, if there is no change in the functionality. [Sm13; HF10, p.351]

Fig. 3 Schematic diagram of verify branch by abstraction

According to HUMBLE & FARLEY a difficulty of the abstraction layer is to find an entry point in the source code which has to be isolated. If no entry point can be found, the code base has to be refactored. Nevertheless, HUMBLE & FARLEY consider it easier to handle this problem than that of branch by feature. [HF10, p.351]

HUMBLE sums up the advantages of branch by abstraction that way: “your code is working at all times throughout the re-structuring, enabling continuous delivery” [Hu11].

3.3 Impact for Continuous Delivery

Since the whole development is done on the mainline, CI is ensured. To develop new features in parallel, both methods, feature toggles and branch by abstraction, can be used. In this way new features can be transported to production without disturbing it. Correspondingly the strategy develop on mainline is very suitable for the development of new features in the context CD. This strategy represents a necessary condition for CI to HUMBLE & FARLEY: “In fact, it is an extremely effective way of developing, and the only one which enables you to perform continuous integration” [HF10, p.405].

Pros	Cons
No merging problems and integration delay Testing during running time Separation of deployment and release “Big Bang release” is avoided	Maintainability of software is complicated when features are cross-cutting

Table 2: Advantages and disadvantages of develop on mainline

4 Hybrid solutions

As described in the previous chapters, develop on mainline suits better than branch by feature in the context of CD. Nevertheless, branch by feature should not be excluded generally. It can make sense in hybrid scenarios. But this should only be practiced for small critical hotfixes according to HUMBLE & FARLEY. Instead of rollbacks the authors recommend rollforwards. This results from the fact, that the deltas are small between two releases. [HF10, p.351]

Another use case to HUMBLE & FARLEY, which allows hybrid scenarios or makes them even necessary, concerns software programmed after the "Big Ball of Mud" pattern. It can be difficult to put an abstraction layer over an entry point. If such an entry point (typically in form of an interface) is not found, the code has to be refactored. In that case, branches can be used. [HF10, p.351]

In order to increase the acceptance of develop on mainline, hybrid scenarios are conceivable in the transitional phase.

5 Recommendation

As a conclusion of this paper, a solution is recommended and further practical examples are introduced.

KRUSCHE & ALPEROWITZ describe an experiment to lead students to CD by using the strategy branch by feature. Over 50% of the student did not have any experience with

CD. As a result of the experiment students actually want to use this strategy for further projects. That paper does not mention that this method leads to “merging hell” and no CI is possible. This could be because the projects were very small and there have been just a few merging conflicts. In the paper there is no information about the size of the group. [KA14, p.337]

It becomes evident that a first acceptance for CD can be created by using the strategy branch by feature. In small project teams (up to 3 people) this strategy can be sufficient if the rules from chapter 2.1 are followed. The most important point is that only short-lived feature branches are used. But in general this strategy is not suitable for CD.

One disadvantage of branch by feature is that short-lived branches are not always possible. Thus, CI cannot be guaranteed. Even short-lived feature branches result in an integration delay because features are integrated after they were completed. These substantial disadvantages lead to the recommendation that develop on mainline within the context of CD should be used in combination with feature toggles. If changes in the software architecture are intended, branch by abstraction can be used. NEELY & STOLT and MEYER explain the successful use of the strategy develop on mainline with the help of feature toggles. Both describe their positive experience with the strategy using only one mainline. According to the authors the use of feature toggles is mandatory in order to develop new features. NEELY & STOLT used to work with branch by feature before. They switched to feature toggles because the effort of merging became too complex. As described in chapter 3.1., the authors TANG ET AL also report positively in about the use of feature toggle at Facebook.

Another crucial advantage of feature toggles in contrast to feature branches is the possibility to test new features in production and to turn them on and off. This allows for example A/B testing. According to HUMBLE the use of branch by abstraction is recommended especially for bigger changes in context of CD. In order to test the changes, the combination verify branch by abstraction with feature toggles and branch by abstraction should be used. [HF10, p.351; Sm13]

Hybrid scenarios as described in chapter 4 are conceivable but should be used just in special cases. According to HUMBLE & FARLEY this is for example necessary for software that was programmed after the „Big Ball of Mud” pattern. Furthermore, small hotfixes can be implemented as feature branches. [HF10, p.351]

The strategy develop on mainline combined with the techniques feature toggles and branch by abstraction is the most suitable solution for CD. Only in exceptions hybrid scenarios should be used.

References

[Bi14] Bird, J.: Feature Toggles are one of the Worst kinds of Technical Debt, 2014, https://dzone.com/articles/feature-toggles-are-one-worst (retrieved 11.02.2016).

[Ch15] Chen, L.: Continuous Delivery: Huge Benefits, but Challenges Too. IEEE Software, Band 32, pp. 50-54, 2015.

[CS14] Chacon, S.; Straub, B.: Pro Git – Everything you need to know about Git. 2. Auflage, Apress, Berkeley CA, 2014.

[Eb15] Ebbers, H..: Jede Änderung ein Feature, 2015, https://jaxenter.de/jede-aenderung-einfeature-18354 (retrieved 11.02.2016).

[Ek16] Ekart, G..: Feature Toggles Revisited, 2016, http://www.infoq.com/news/2016 /02/featuretoggles (retrieved 11.02.2016).

[Fo06] Fowler, M.: Continuous Integration, 2006, http://www.martinfowler.com/articles /continuousIntegration.html (retrieved 11.02.2016).

[Ha07] Hammat, P..: Introducing Branch By Abstraction, 2007, http://paulhammant.com/blog/branch_by_abstraction.html (retrieved 11.02.2016).

[Ha09] Harmes, R..: Flipping Out, 2009, http://code.flickr.net/2009/12/02/flipping-out/.

[HF10] Humble, J.; Farley, D.: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Addison Wesley, Boston, 2010.

[Ho16] Hodgson, P..: Feature Toggles, 2016, http://www.martinfowler.com/articles /continuousIntegration.html (retrieved 11.02.2016).

[Hu11] Humble, J..: Make Large Scale Changes Incrementally with Branch By Abstraction. 2011, http://continuousdelivery.com/2011/05/make-large-scale-changes-incrementally-with-branch-by-abstraction/ (retrieved 11.02.2016).

[KA14] Krusche, S.; Alperowitz, L.: Introduction of Continuous Delivery in Multi-customer Project Courses. In: Companion Proceedings of the 36th International Conference on Software Engineering, ACM, pp. 335-343, 2014.

[Me14] Meyer, M.: Continuous Integration and Its Tools. IEEE Software, Band 31, Heft 3, pp.14-16, 2014.

[NS13] Neely, S.; Stolt, S.: Continuous delivery? Easy! Just Change Everything (well, maybe it is not that easy). In: Agile Conference, IEEE, pp. 121–128, 2013.

[PS13] Preißel, R.; Stachmann, B.: Git: Dezentrale Versionsverwaltung im Team - Grundlagen und Workflows. 2. Auflage, dpunkt.verlag, Heidelberg, 2013.

[Ro16] Rodríguez, P. et al.: Continuous deployment of software intensive products and services: A systematic mapping study. The Journal of Systems and Software, 2016.

[Sm13] Smith, S.: Application Pattern: Verify Branch By Abstraction. 2013 https://dzone.com/articles/application-pattern-verify (retrieved 11.02.2016).

[Ta15] Tang, C. et al.: Holistic configuration management at Facebook. ACM, New York, 2015.

[Wo15] Wolff, E.: Continuous Integration widerspricht Feature Branches. 2015 https://heise.de/-2736487 (retrieved 11.02.2016).