Outline

Development Scenario

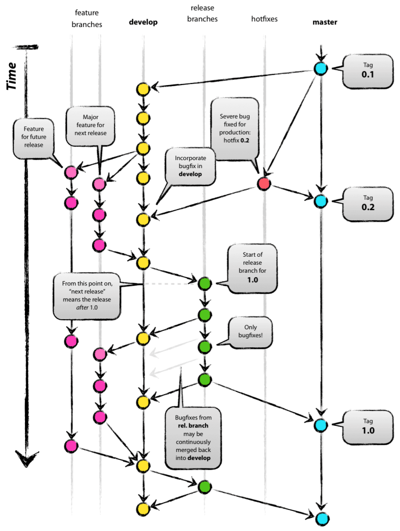
Imagine a small software team working on a mobile grocery list application. The application already has the capability of capturing simple shopping lists and checking items off the lists.

- Jeremiah is working from home (remotely) on allowing lists to be shared between users. This feature will take about four days to implement and he is about halfway through completing it.
- Emma works at the office and is adding functionality for assigning list items to store aisles. This will take two days and she is about to complete it. This feature mostly involves changes to the back-end data store.
- Releases are deployed to the Google Play store which, in turn, prompts users to update the application on their devices.

Git Workflows or Branching Strategies

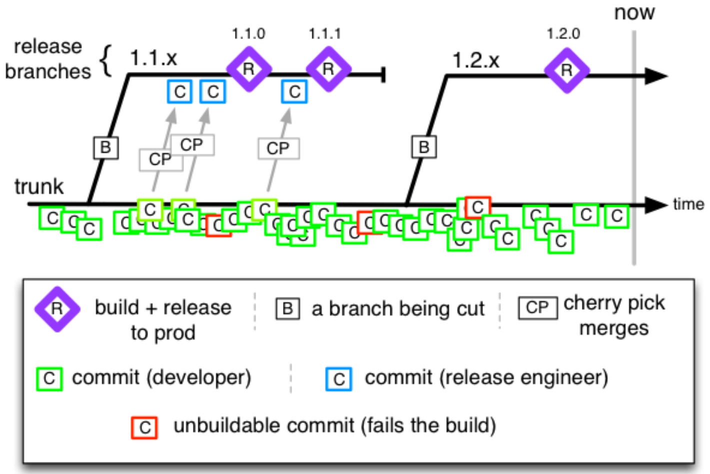
- Many different ways to collaborate using Git
- Two key workflows
  - GitFlow
  - Trunk-Based Development (TBD)

GitFlow Overview


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

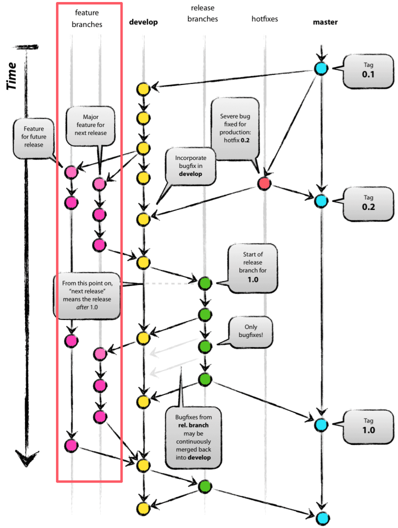
Vincent Driessen
http://nvie.com/posts/a-successful-git-branching-model/

Trunk-Based Development (TBD) Overview



Paul Hammant
http://paulhammant.com/2013/12/04/what_is_your_branching_model/

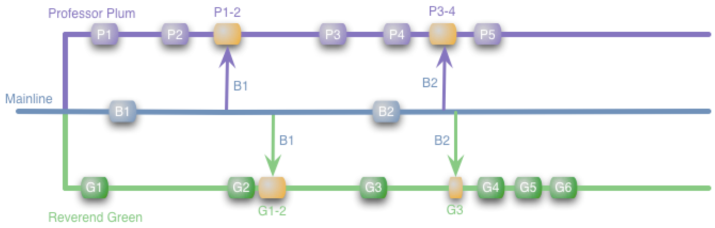
GitFlow — Long-Lived Feature Branches



feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Vincent Driessen
http://nvie.com/posts/a-successful-git-branching-model/

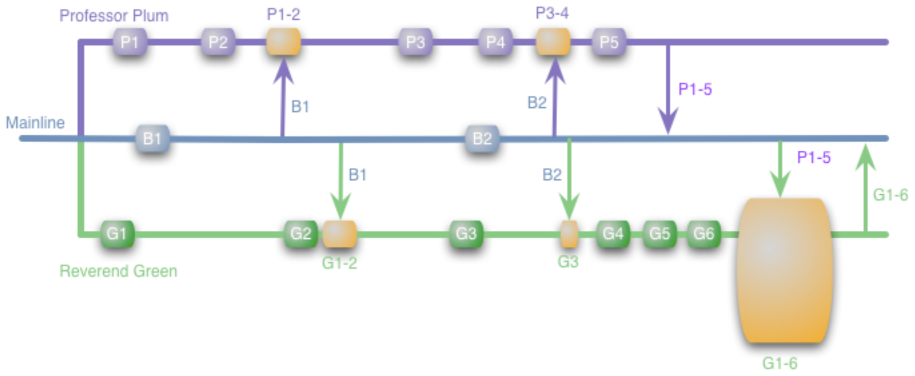
Long-Lived Feature Branches — Local/Main Branch Interaction



Martin Fowler
`http://martinfowler.com/bliki/FeatureBranch.html`

- Branch per user story or feature
- Each developer works on their own feature, isolated from changes elsewhere
- Pull in changes at their own pace
- Features can be cherry-picked for release

Long-Lived Feature Branches cont.



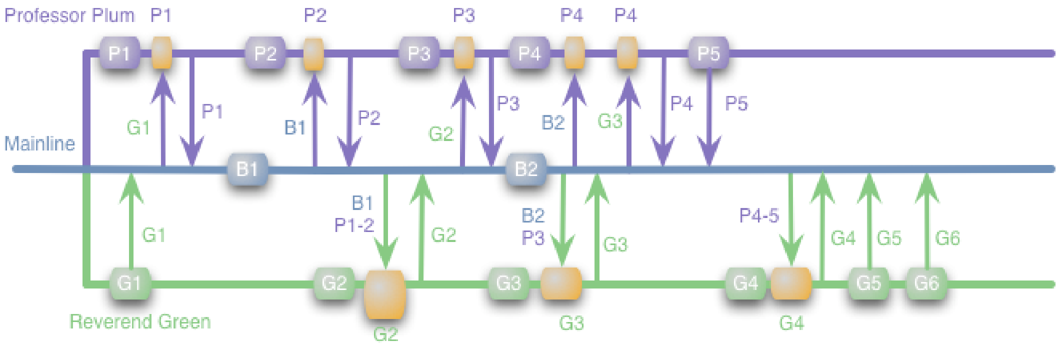
Martin Fowler
http://martinfowler.com/bliki/FeatureBranch.html

Big Scary Merge - G1-6 with P1-5

Dangers in having long-lived, non-trunk branches

- Complexity - technical and cognitive
- Merge conflicts
    - Textual
    - Semantic conflicts (eg. function renames) deter refactoring
- Isolated features, interaction problems discovered late

TBD and Continuous Integration (CI) — Local/Main Branch Interaction



Martin Fowler
http://martinfowler.com/bliki/FeatureBranch.html

TBD and CI continued

- Characterised by short-lived feature branches or commits direct to trunk
- Small batches and frequent merges
- Feature branches cut directly from master and return as *pull requests* into master
- Incomplete features must be masked

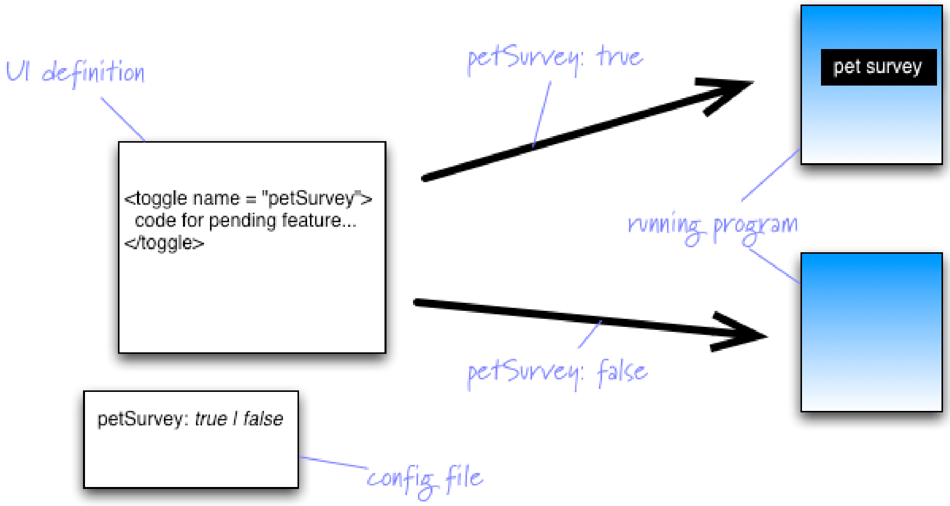
Advantages of TBD and CI

- Shortens the feedback cycle
- Promotes communication, increases visibility and collaboration
- Highlights issues

Feature Flags or Toggles

- Useful technique for hiding partly built features
- Use as a last resort — rather *break features down* in smaller, useful, pieces

UI Toggles



```
<toggle name = "petSurvey">
  code for pending feature...
</toggle>
```

```
petSurvey: true / false
```

Martin Fowler
https://martinfowler.com/bliki/FeatureToggle.html

Logic Layer Toggles

Provide the ability to easily turn application features on and off

- Hard code the feature choice

```
function reticulateSplines(){
  let useNewAlgorithm = false;
  // useNewAlgorithm = true; // UNCOMMENT IF YOU ARE WORKING ON THE NEW
      SR ALGORITHM

  if( useNewAlgorithm ) {
    return enhancedSplineReticulation();
  else {
    return oldFashionedSplineReticulation();
  }
}
```

Logic Layer Toggles cont.

- Use a command-line argument

```
// Java
if args.contains("--withOneClickPurchase") {
  purchasingCompleting = new OneClickPurchasing();
}
```

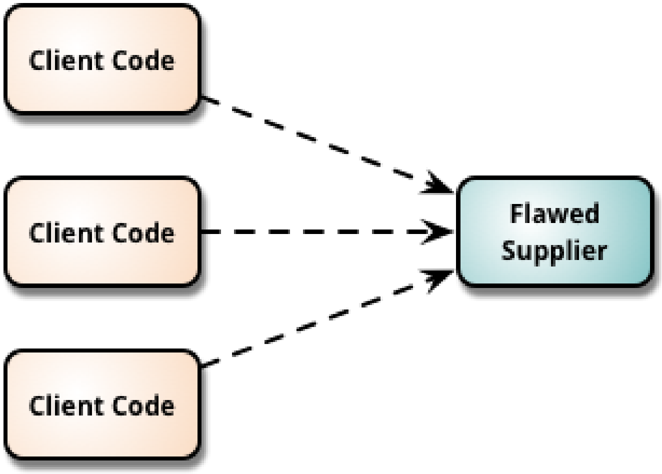
- Store feature configuration in a file and read it in at run-time
- Use existing libraries

Branch by Abstraction

For making a large-scale change to a software system in gradual way

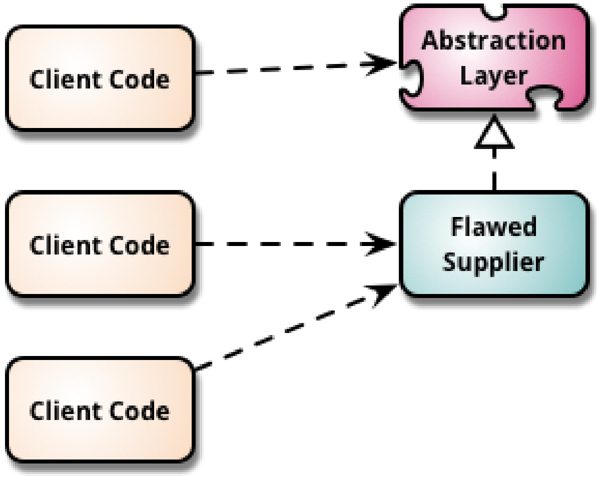
- Change is time-consuming
- Lots of developers already depend on the code that is the subject of the change
- As always, no commit should break the trunk — change must be incremental

Branch by Abstraction



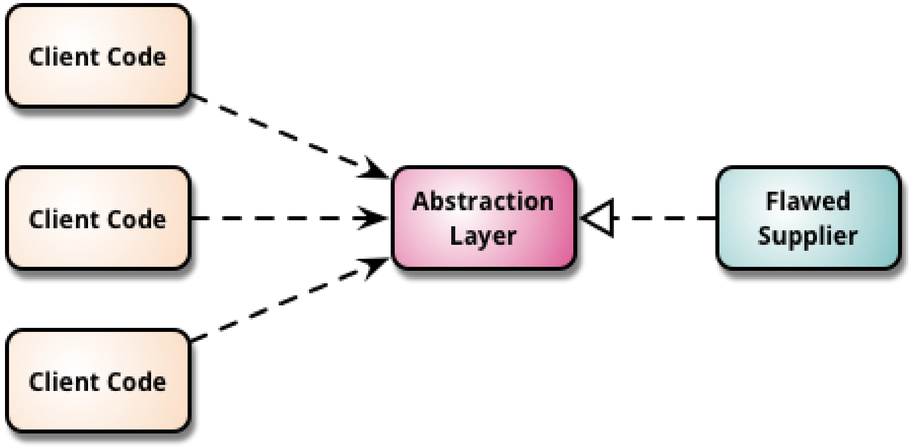
Martin Fowler
https://martinfowler.com/bliki/BranchByAbstraction.html

Branch by Abstraction - Step 1



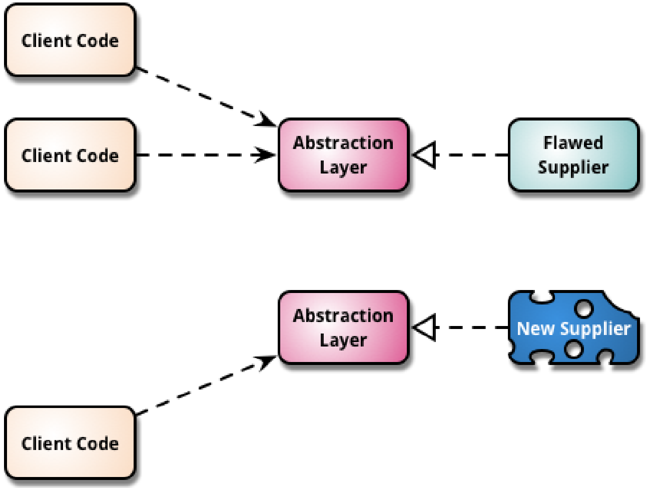
Martin Fowler
https://martinfowler.com/bliki/BranchByAbstraction.html

Branch by Abstraction - Step 2



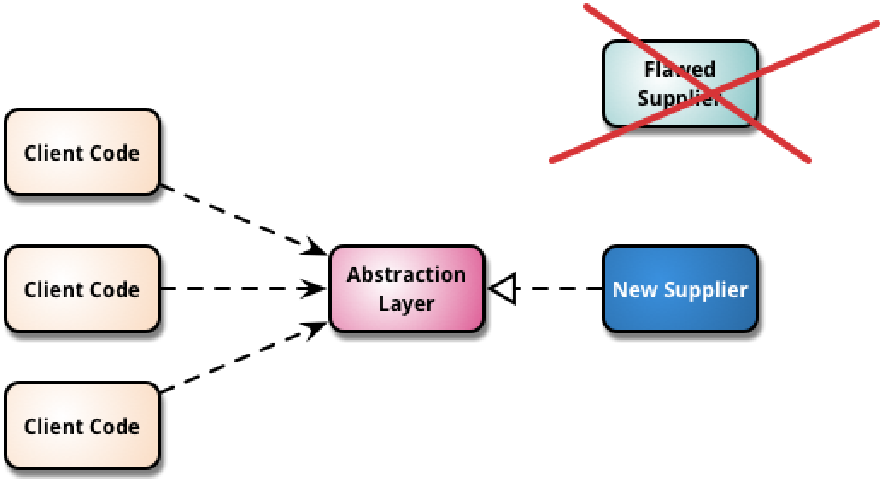
Martin Fowler
https://martinfowler.com/bliki/BranchByAbstraction.html

Branch by Abstraction - Step 3



Martin Fowler
https://martinfowler.com/bliki/BranchByAbstraction.html

Branch by Abstraction - Step 4



Martin Fowler
https://martinfowler.com/bliki/BranchByAbstraction.html

Revisiting the Development Scenario

Imagine a small software team working on a mobile grocery list application. The application already has the capability of capturing simple shopping lists and checking items off the lists.

- Jeremiah is working from home (remotely) on allowing lists to be shared between users. This feature will take about four days to implement and he is about halfway through completing it.
- Emma works at the office and is adding functionality for assigning list items to store aisles. This will take two days and she is about to complete it. This feature mostly involves changes to the back-end data store.
- Releases are deployed to the Google Play store which, in turn, prompts users to update the application on their devices.