Continuous Integration, Delivery, and Deployment

Stephen Levitt

School of Electrical and Information Engineering
University of the Witwatersrand

2020
1. Continuous Delivery

2. Git Workflows

3. Typical Feature Branching (as in GitFlow)

4. Trunk-Based Development and Continuous Integration
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 their devices.
Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.

— Martin Fowler
Why implement Continuous Delivery?

What concerns do you have around such an approach?

What needs to be in place to allow a development team to practise continuous delivery?

- What infrastructure is involved deploying a new feature coded on a developer’s laptop into a production environment?
- How should the team collaborate?
- How do you deal with the fact that at any given point in time some features are not complete?
Benefits of CD

- Faster reaction times
- Reduced risk; increased stability
- Exposed inefficiencies and costs $\Rightarrow$ increased productivity
- Flexible release options
## Software Delivery Performance

### Source:
Accelerate: State of DevOps 2018 by DORA

<table>
<thead>
<tr>
<th>Aspect of Software Delivery Performance</th>
<th>Elite&lt;sup&gt;a&lt;/sup&gt;</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deployment frequency</strong>&lt;br&gt;For the primary application or service you work on, how often does your organization deploy code?</td>
<td>On-demand (multiple deploys per day)</td>
<td>Between once per hour and once per day</td>
<td>Between once per week and once per month</td>
<td>Between once per week and once per month</td>
</tr>
<tr>
<td><strong>Lead time for changes</strong>&lt;br&gt;For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?</td>
<td>Less than one hour</td>
<td>Between one day and one week</td>
<td>Between one week and one month&lt;sup&gt;b&lt;/sup&gt;</td>
<td>Between one month and six months&lt;sup&gt;b&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>Time to restore service</strong>&lt;br&gt;For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?</td>
<td>Less than one hour</td>
<td>Less than one day</td>
<td>Less than one day</td>
<td>Between one week and one month</td>
</tr>
<tr>
<td><strong>Change failure rate</strong>&lt;br&gt;For the primary application or service you work on, what percentage of changes results either in degraded service or subsequently requires remediation (e.g., leads to service impairment, service outage, requires a hotfix, rollback, fix forward, patch)?</td>
<td>0-15%</td>
<td>0-15%</td>
<td>0-15%</td>
<td>46-60%</td>
</tr>
</tbody>
</table>

<sup>a</sup> Elite performance is defined as the top 25% of organizations.

<sup>b</sup> These timeframes are approximations and can vary based on specific circumstances.
Continuous Integration, Delivery, and Deployment: Continuous Delivery

Source: 2015 State of Dev Ops Report by puppet labs
Continuous Delivery Infrastructure

Continuous Delivery, Delivery, and Deployment

Source: U Meding

http://uwemeding.com/making-the-case-for-continuous-integration/
Continuous Delivery versus Continuous Deployment

Source: Y Sundman

http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment
Continuous Delivery Factors/Ingredients and Effects

- Comprehensive, fast and reliable test and deployment automation
- Trunk-based development and continuous integration
- Application code and app and system configuration all in version control

Together, the factors on the left model continuous delivery which leads to...

- Lower levels of deployment pain
- Higher levels of IT performance (higher throughput and stability)
- Lower change fail rates
- Higher levels of org performance (productivity, market share, profitability)

Source: 2015 State of Dev Ops Report by puppet labs
Many different ways to collaborate using Git

Two key workflows
- GitFlow
- Trunk-Based Development (TBD)
Trunk-Based Development (TBD) Overview

Continuous Integration, Delivery, and Deployment: Git Workflows

[Diagram showing trunk-based development with release branches and timeline]

- **R** build + release to prod
- **B** a branch being cut
- **CP** cherry pick merges
- **C** commit (developer)
- **C** commit (release engineer)
- **C** unbuildable commit (fails the build)

Paul Hammant

http://paulhammant.com/2013/12/04/what_is_your_branching_model/
- 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
Big Scary Merge - G1-6 with P1-5
Dangers in having long-lived, non-trunk branches

- Complexity
- Merge conflicts
  - Textual
  - Semantic conflicts (e.g., function renames) deter refactoring
- Isolated features, interaction problems discovered late
TBD and Continuous Integration (CI) — Local/Shared Branch Interaction

Martin Fowler

http://martinfowler.com/bliki/FeatureBranch.html
Characterised by short-lived feature branches and frequent, small merges due to daily (or near-daily) merges with trunk (contrast to GitFlow)

Feature branches cut directly from master and return as pull requests into master

CI server isolated builds used to verify pull requests before they are merged in master

Incomplete features must be masked — Feature Toggles and Branch by Abstraction

How do we handle code reviews in this situation?
Advantages of TBD and CI

- Communication - increased visibility and collaboration
- Merge conflicts are relatively easy to resolve
- Each commit gives birth to a release candidate
Feature Flags or Toggles

Provide the ability to easily turn application features on and off

- Hard code the feature choice

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

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

- Use a command-line argument
  ```java
  // Java
  if args.contains("--withOneClickPurchase") {
      purchasingCompleting = new OneClickPurchasing();
  }
  ```

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

- Change is time-consuming
- Lots of developers already depend on the code that is the subject of the change
- No commit that is pushed to/merged with master should jeopardize the production-ready status of the branch of the trunk
Branch by Abstraction Approach

- Introduce an abstraction around the code to be changed
- Write a new implementation for the abstraction which caters for the changes
- Migrate code to the new implementation
- Remove the old implementation
- Remove the abstraction
Branch by Abstraction - Step 1

Continuous Integration, Delivery, and Deployment: TBD and CI
Branch by Abstraction - Step 2

Continuous Integration, Delivery, and Deployment: TBD and CI
Branch by Abstraction - Step 3

Continuous Integration, Delivery, and Deployment: TBD and CI
Branch by Abstraction - Step 4

- Client Code
- Abstraction Layer
- Flawed Supplier
- New Supplier

Continuous Integration, Delivery, and Deployment: TBD and CI
Branch by Abstraction - Step 5

Continuous Integration, Delivery, and Deployment: TBD and CI