Mobile Continuous Delivery is closer, easier (and more fun) than you think!

Karl Krukow
MyDriving Mobile & Azure IOT
Release More Often!

- yearly
- quarterly
- monthly
- bi-weekly
- weekly
Why?
Key Benefits of Continuous Delivery

- Reduced lead-time
- Faster feedback
- Higher-Quality
Emotional cycle of manual delivery

The business case for continuous delivery, Atlassian blog
<table>
<thead>
<tr>
<th>Release Cadence</th>
<th>bi-weekly</th>
<th>weekly</th>
<th>weekly</th>
<th>weekly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. rating</td>
<td>4.5</td>
<td>3.5</td>
<td>4.2</td>
<td>3.5</td>
</tr>
<tr>
<td># Ratings</td>
<td>155,344</td>
<td>2,975,737</td>
<td>71,968</td>
<td>354,415</td>
</tr>
</tbody>
</table>

Sources: https://www.applyzer.com/ & App Store

* = approximation based on current frequency
Ratings as of June 11, 2017
What’s Stopping Us?
Apple review cycle time


http://appreviewtimes.com/
Continuous Integration
August 15, 2015

Setting Up A Continuous Build Environment For Xamarin: Part 1 - Jenkins

Continuous integration is the core foundation of the DevOps lifecycle, as it allows tests to be run and builds to be created every time a member of the development team checks in new code. This allows the team to quickly know if the latest changes have 'broken the build' and depending on how the continuous integration is configured, for that check in or commit to be rejected (essentially allowing the team to pre-emptively avoid major issues from ever making it into the build).

Setting up a continuous integration environment for Xamarin is a non-trivial task because of the number of dependencies that Xamarin requires to build packages (a lot of this is thanks to Apple's requirements that you can only build iOS apps on a Mac). So in this 2 part tutorial series, I'll walk through configuring two different continuous integration solutions for Xamarin. Part 1 will be built using a combination of Jenkins (for Xamarin.iOS and Xamarin.Android) and a hosted TFS build controller (for Windows/Windows Phone). Part 2 will be built using the new 'BuildvNext' tools that are included Visual Studio Online and the newly released Team Foundation Server 2015.

SAMPLE SOLUTION

To give you a sample solution to work with for building these continuous integration environments, I've taken the TaskyPortable solution from the Xamarin Samples GitHub account, made some modifications and uploaded it to my GitHub account. So you just need to download this solution and check it in/commit it to your VSO or TFS Team Foundation Server accounts.
CI-As-A-Service?
Automated UI Testing

• Interact with UI controls in your app using gestures
• Declarative query language to identify views on screen
• Wait for events to occur (e.g., no spinner visible)
• App-lifecycle APIs (start/stop, reset, etc)
• Generate screenshots for test reports

//Example in C#
app.Tap("Help");
app.Tap (e => e.Id ("history-btn"));
app.WaitForElement (e => e.Text ("Ink"));
app.Screenshot ("View the purchasing history");
Demo: Xamarin.UITest
(yes, it works with all apps, not just Xamarin :)
How?
What are all the steps needed to ship?

CI

Experiment
Idea
Change

SCM change
Build sim/device
Unit Tests
Int. Tests

What else?

Continuous Delivery: Anatomy of the Deployment Pipeline
What else?
What are all the steps needed to ship?
DAUNTING
What are all the steps needed to ship?
Build-Test-Distribute with Mobile Center

SCM change → Build → Test → Distribute
Release More Often!

- yearly
- quarterly
- monthly
- bi-weekly
- weekly

https://aka.ms/mobilecenter
Please

Remember to rate this session

Thank you!