Continuous Delivery for Android Applications

Introduction

I hear that phrase around the office place almost every day. Sometimes it seems it's more of fashion statement than an actual description. And yet many companies claim that they do continuous delivery, but what exactly is that?

Even at Sky, where I currently work, we struggle to perfect it to the level in which we would be all satisfied.

Sky has dozens of different teams, APIs, platforms, products and business propositions. Managing efficient development is such an overpopulated space will always be an difficult task.
That's why we need to automate as much as possible so we can concentrate on these differences to get out the best from each developer in our teams.

In this article I will try to simulate issues we face on daily basis, but on a much smaller scale.

I will try to tackle this from bottom up by using 2 tools which I always find useful. storytelling and code writing, the rest is just my poor attempt to be an article writer and mostly to fill the gaps in between.

Those who get bored quickly might just download the sources, but you will still need to do some work outside of the code itself to make it all happen:
https://github.com/beherithrone/calculator-cd-android

I'll start with an fictional company called "Skies" whose owners have an ambitious plan to build the most amazing apps ever. They manage to hire first developer called "Hero".

Prerequisites

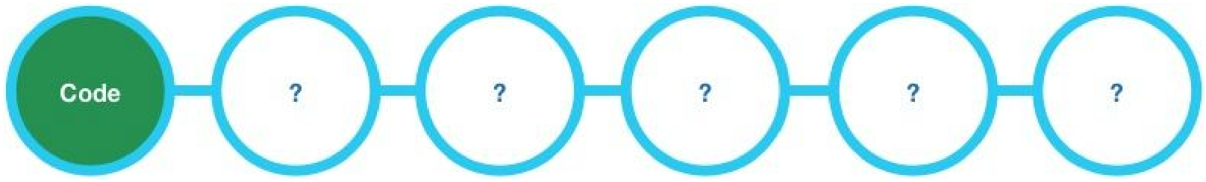
- Company wants to build amazing Calculator app initially
- Company has zero infrastructure
- Company has some money to spend, but would like to spend little and then scale it up if necessary when it comes to tools
- Company wants to invest in building development teams

Requirements

- App should be "deployable" at any state
- Every commit to master should be deployable to stakeholder without manual step
- Initial setup should be as cheap as possible (= free)

At the end of this epic journey you should have good understanding of not only Continuous Deployment, but also why and how to get there.

“Once upon a time there was a developer called Hero who had to face 6 challenges in order to please his demanding manager and reach the holy grail of Continuous Delivery. At first he had to write some code”



Code

Code is of course at the heart of any software development so we want to make sure the environment to write is as optimized as possible.

IDE

There are several choices including Android Studio, Eclipse, Intellij and so on.

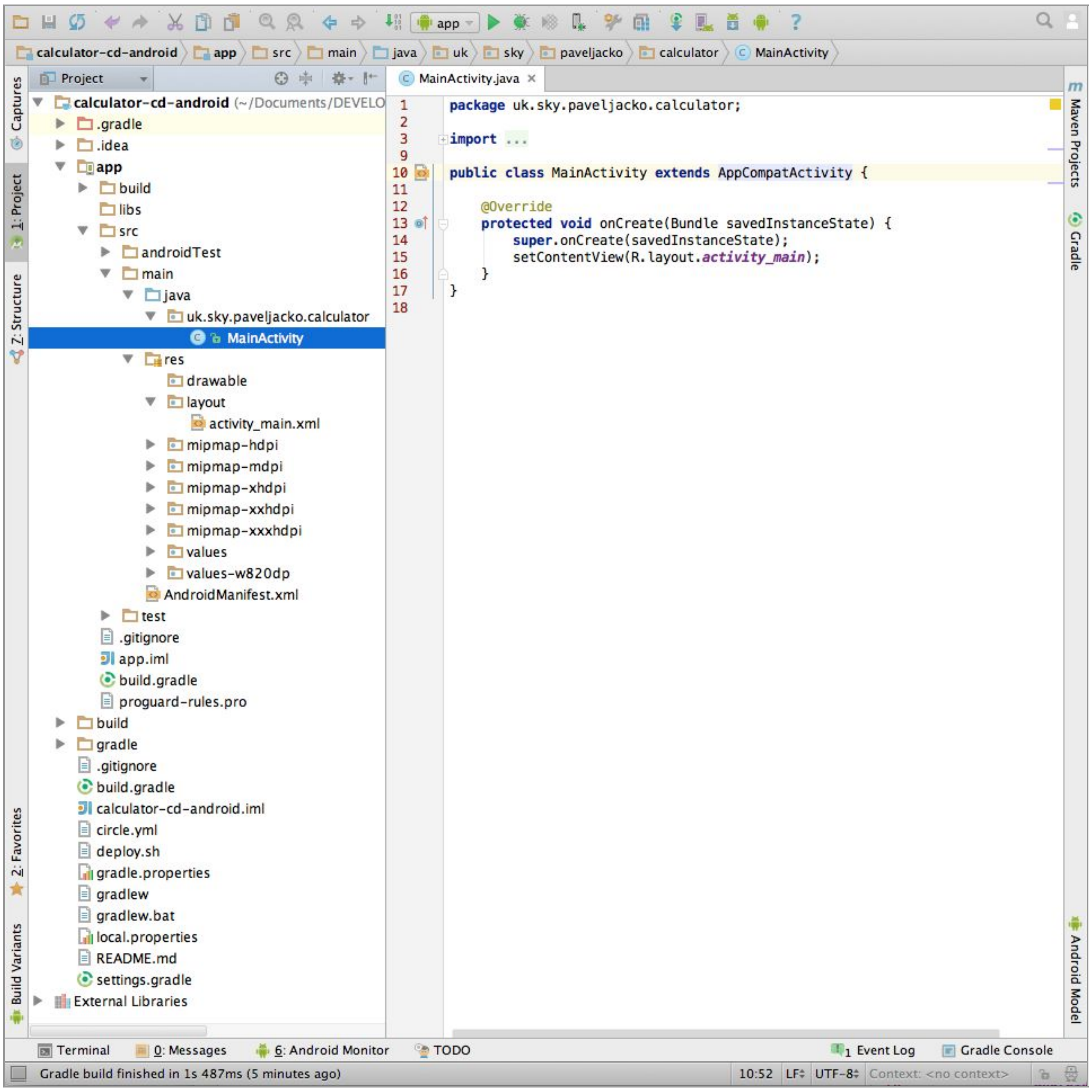
I use Android Studio
http://developer.android.com/tools/studio specifically 1.4 RC1 simply because it does what it says on the tin.

Project

Ok, let’s set ourselves up with a simple project. We’ll start with a blank activity project from Android templates:

```
Android Studio => File => New Project => Next.. => Blank Activity => Finish
```

After the completed setup our project structure should look similar to this:



```java
package uk.sky.paveljacko.calculator;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Gradle build finished in 1s 487ms (5 minutes ago)

We’ll build a very simple UI. Calculator

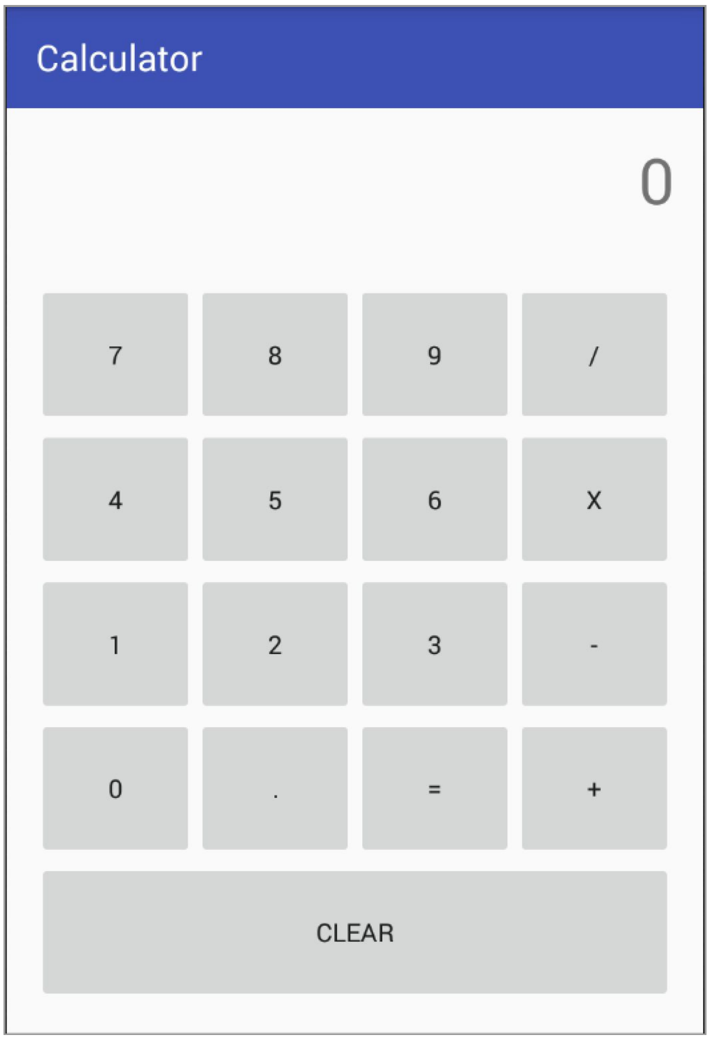
replace content of ***activity_main.xml*** with layout below:

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
```

```xml
android:paddingTop="@dimen/activity_vertical_margin"
app:layout_behavior="@string/appbar_scrolling_view_behavior"
tools:context=".MainActivity"
tools:showIn="@layout/activity_main">

<TextView
   android:id="@+id/txt_result"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:layout_weight="1"
   android:gravity="right"
   android:text="0"
   android:textSize="35sp" />

<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:layout_weight="1"
   android:orientation="horizontal">

   <Button
      android:id="@+id/btn_7"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_weight="1"
      android:text="7" />

   <Button
      android:id="@+id/btn_8"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_weight="1"
      android:text="8" />

   <Button
      android:id="@+id/btn_9"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_weight="1"
      android:text="9" />

   <Button
      android:id="@+id/btn_divide"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_weight="1"
      android:text="/" />

</LinearLayout>

<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
```

```
    android:layout_weight="1"
    android:orientation="horizontal">

    <Button
        android:id="@+id/btn_4"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="4" />

    <Button
        android:id="@+id/btn_5"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="5" />

    <Button
        android:id="@+id/btn_6"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="6" />

    <Button
        android:id="@+id/btn_multiply"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="X" />

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:orientation="horizontal">

    <Button
        android:id="@+id/btn_1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="1" />

    <Button
        android:id="@+id/btn_2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="2" />
        <Button
            android:id="@+id/btn_3"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="3" />

        <Button
            android:id="@+id/btn_minus"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="-" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:orientation="horizontal">

        <Button
            android:id="@+id/btn_0"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="0" />

        <Button
            android:id="@+id/btn_comma"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="." />

        <Button
            android:id="@+id/btn_equals"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="=" />

        <Button
            android:id="@+id/btn_plus"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="+" />
    </LinearLayout>

    <Button
        android:id="@+id/btn_clear"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="Clear" />

</LinearLayout>
```

This is fairly ugly, but it’s good enough for our purposes.
If you run the example it should look similar to this:



The calculator does nothing right now. It’s time to add some code.

Replace code in ***MainActivity.class*** with code below:

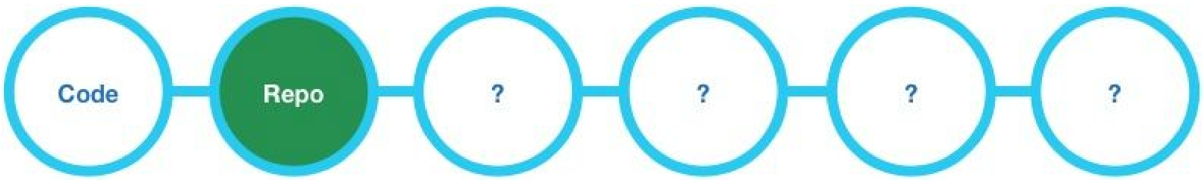
```
package uk.sky.paveljacko.calculator;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

   String result = "";
   Float value = 0.0f;
   Operator lastOperator;
   TextView textView;
   private boolean needsClear = true;

   ButtonHolder[] numbers = new ButtonHolder[] {
           new ButtonHolder(R.id.btn_0, 0),
           new ButtonHolder(R.id.btn_1, 1),
           new ButtonHolder(R.id.btn_2, 2),
           new ButtonHolder(R.id.btn_3, 3),
           new ButtonHolder(R.id.btn_4, 4),
           new ButtonHolder(R.id.btn_5, 5),
           new ButtonHolder(R.id.btn_6, 6),
           new ButtonHolder(R.id.btn_7, 7),
           new ButtonHolder(R.id.btn_8, 8),
           new ButtonHolder(R.id.btn_9, 9)
   };

   ButtonHolder[] operators = new ButtonHolder[] {
           new ButtonHolder(R.id.btn_divide, Operator.Divide),
           new ButtonHolder(R.id.btn_minus, Operator.Minus),
           new ButtonHolder(R.id.btn_multiply, Operator.Multiply),
           new ButtonHolder(R.id.btn_equals, Operator.Equals),
           new ButtonHolder(R.id.btn_plus, Operator.Plus)
   };

   ButtonHolder[] actions = new ButtonHolder[] {
           new ButtonHolder(R.id.btn_divide, Operator.Clear),
           new ButtonHolder(R.id.btn_minus, Operator.Comma),
           new ButtonHolder(R.id.btn_clear, Operator.Clear)
   };

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
       setSupportActionBar(toolbar);

       textView = (TextView) findViewById(R.id.txt_result);

       for(final ButtonHolder buttonHolder : numbers) {
           findViewById(buttonHolder.id).setOnClickListener(new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                   addNumber(buttonHolder.value);
               }
```

```
        });
    }

    for(final ButtonHolder buttonHolder : operators) {
        findViewById(buttonHolder.id).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                addOperation(buttonHolder.operator);
            }
        });
    }

    for(final ButtonHolder buttonHolder : actions) {
        findViewById(buttonHolder.id).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                doAction(buttonHolder.operator);
            }
        });
    }
}

private void addNumber(int number) {
    if(needsClear) {
        result = "";
        needsClear = false;
    }
    result += number;
    updateResultText(result);
}

private void doAction(Operator operator) {
    switch (operator) {
        case Comma:
            result += ".";
            break;
        case Clear:
            result = "0";
            needsClear = true;
            value = 0.0f;
            lastOperator = null;
            updateResultText(result);
            break;
    }
}

private void addOperation(Operator operator) {
    needsClear = true;
    calculate();
    lastOperator = operator;
    updateResultText(value.toString());
}
```

```
    private void calculate() {
        if(lastOperator == null) {
            value = Float.parseFloat(result);
            return;
        }

        switch (lastOperator) {
            case Plus:
                value += Float.parseFloat(result);
                break;
            case Minus:
                value -= Float.parseFloat(result);
                break;
            case Multiply:
                value *= Float.parseFloat(result);
                break;
            case Divide:
                value /= Float.parseFloat(result);
                break;
        }
    }

    private void updateResultText(String value) {
        textView.setText(value);
    }

    private class ButtonHolder {
        public int id;
        public Operator operator;
        public int value;
        public ButtonHolder(int id, Operator operator) {
            this.id = id;
            this.operator = operator;
        }
        public ButtonHolder(int id, int value) {
            this.id = id;
            this.value = value;
        }
    }

    private enum Operator {
        Plus,
        Minus,
        Divide,
        Multiply,
        Equals,
        Comma,
        Clear
    }
}
```

Build it and run it.
Great, we have a running and fully functional Calculator app!

"Hero succeeded in his first quest and now he can expand his Hero team with good developers, but they cannot work together efficiently"



Repository

These days, building any larger scale project without some sort of versioning system is as suicidal as eating a bunch of naga chilli hot wings day before your most important interview. You will regret it, the question is just when. It will most likely be when your interviewer asks "Do you have any questions?"

We have a couple of options: Git, Svn, CVS or Mercurial
I don't remember the last time I used SVN or Mercurial so I'll stick with git and Github hosting specifically.

Github

Github - it's not just a code repository. It also acts as a ticket management tool as well. At Sky, we recently migrated from Enterprise JIRA to github issues on some project and feedback has been positive so far.

I assume every developer has an github account these days. It's the nerdy equivalent of Facebook, just without the annoying videos of cute babies pooping around.

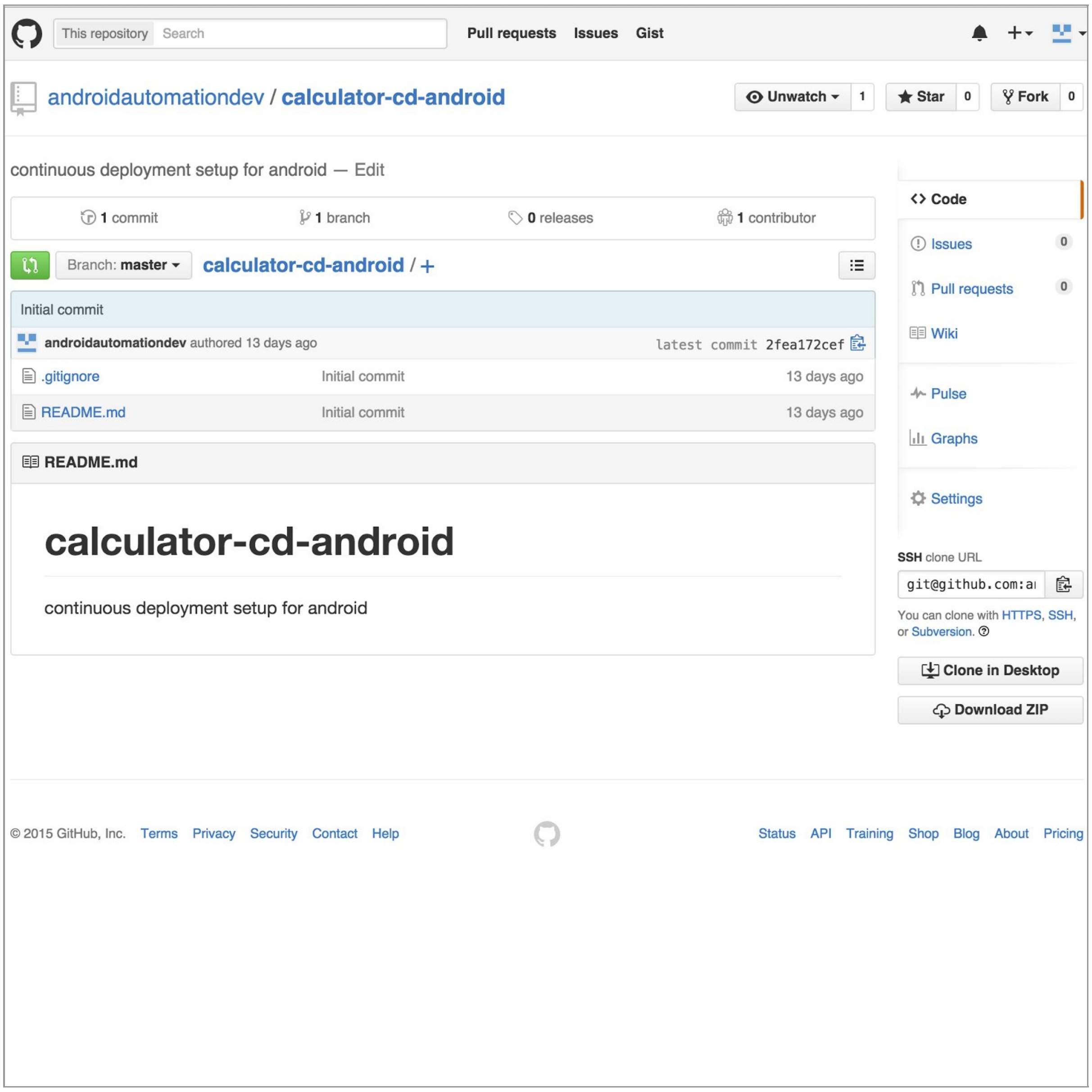
If you haven't already:

```
Walk out of your cave => Github => Sign up => Verify your email
```

Now, let's create a new placeholder for our project:

```
Github => Repositories => New => calculator-cd-android => Create repository
```

At Sky we use the naming convention [name-subname-platform], but you can choose a different one.
I'll call this project *calculator-cd-android* - how inventive is that!


This repository Search

Pull requests Issues Gist

androidautomationdev / calculator-cd-android

Unwatch 1 Star 0 Fork 0

continuous deployment setup for android — Edit

1 commit 1 branch 0 releases 1 contributor

Branch: master calculator-cd-android / +

Initial commit

androidautomationdev authored 13 days ago latest commit 2fea172cef

.gitignore	Initial commit	13 days ago
README.md	Initial commit	13 days ago

README.md

calculator-cd-android

continuous deployment setup for android

Code

Issues 0

Pull requests 0

Wiki

Pulse

Graphs

Settings

SSH clone URL

git@github.com:a

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

© 2015 GitHub, Inc. Terms Privacy Security Contact Help

Status API Training Shop Blog About Pricing

SSH

Now it's time to clone your repo and for that you will need to set-up your SSH keys.

While you could use HTTP authentication to get what you need from the repo, we'll need SSH keys in later steps, so we better to do our homework right away (that is if you haven't done it already)

To setup a SSH keys just follow:

```
Github => Settings => SSH Keys => Add SSH Key
```

github has very simple and useful guide to do that:
https://help.github.com/articles/generating-ssh-keys/

Once done, you can finally clone your repo.

> Note: make sure you copy SSH url not HTTPS!

As we already have our project set up locally the only thing left for you to do is to glue your local code to the remote repo and upload the changes.

You can do it via command line by executing this in your root project folder:

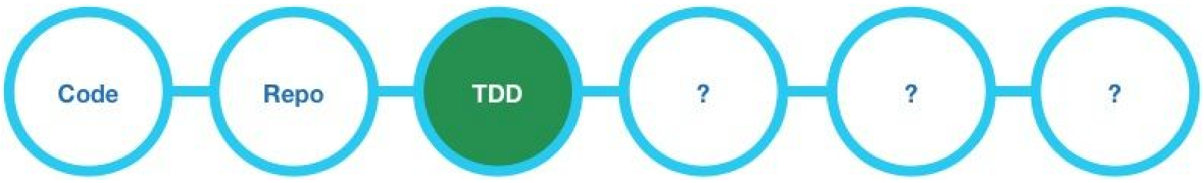
```
$ git init
$ git add .
$ git commit -m "Initial commit"
$ git remote add REMOTE_SSH_REPOSITORY
$ git push origin master
```

> NOTE: if you get git command not found you need to install git client:
> https://git-scm.com/downloads

Now you're ready to go and everybody can contribute to your master branch!
All your team members can collaborate on one master branch.

```
$ git clone REMOTE_SSH_REPOSITORY
```

"They managed to collaborate together but soon after, the bugs arrived and started to tearing their quest to pieces"



TDD

In order to make sure I do not break existing functionality I need some mechanism to prevent me and others from making silly and often costly mistakes.

Concepts

Anybody who is at least roughly familiar with TDD knows this good old rule:

1. write an unit test
2. make it fail
3. add functionality
4. make it pass

Well, in reality that is not always how it happens and the example above demonstrates this. So we'll add unit tests retrospectively to existing functionality, but we are going to use proper TDD to rewrite it slightly.

Tools

There are plenty of tools around to help you with your TDD initiative:

Robolectric, Mockito, JUnit, Cucumber, MockWebServer, AssertJ

For purpose of this exercise I'll use JUnit and Mockito.

Example

If you run a test on your current app:

```
$ ./gradlew test
```

you'll get :

```
$ BUILD SUCCESSFUL

$ Total time: 23.218 secs
```

Don't be fooled as this just indicates that no tests failed which is correct as they do not exist yet!

NOTE: if you set project with exact version of Android Studio as I did you might have one mocked test present called ***ExampleUnitTest.java***. You can safely delete it.

Let’s try write some code TDD style. Actually we are going to refactor current code to match our TDD needs.
First make sure you select correct test artifact in Android Studio.

This will make sure you compile your Unit Tests with your code.

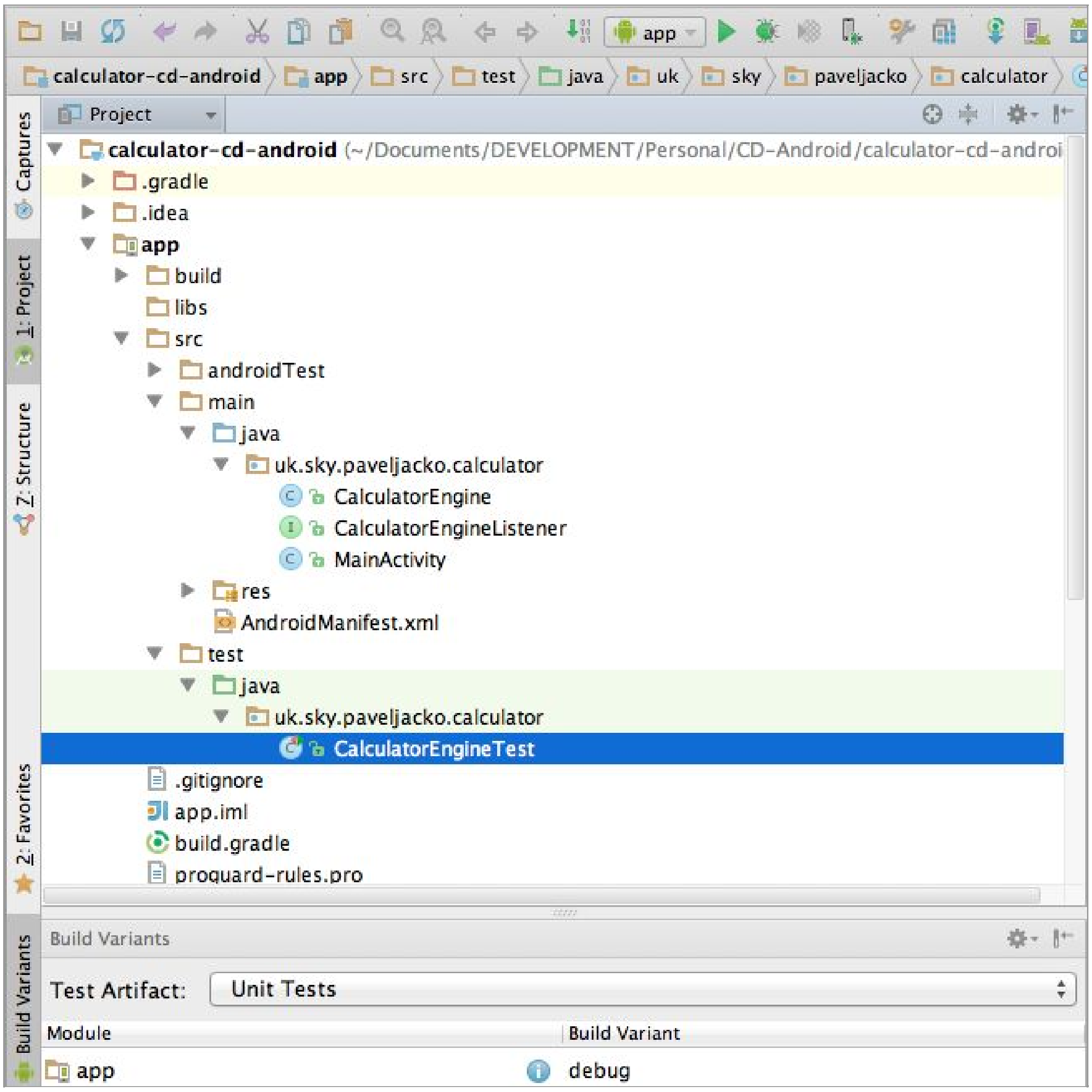
Now, what do we want to unit test? Well let’s go back to your ***MainActivity.java***
All logic is currently embedded there and we can see some calculations happening as well as some UI manipulations.

What if we want to create another mini widget which would do math calculations as well? This would require us to duplicate same code, so let’s extract some piece of functionality into a separate class. Welcome to the world of decoupling.

Writing an Unit Test

Create a new class called ***CalculatorEngine.java***
and a new test class called ***CalculatorEngineTest.java***

Your setup should look like this:



Test Artifact: Unit Tests

Module	Build Variant
app	debug

Now let’s think about what do we want the calculator engine to do:

- It should do calculations
- it should *NOT* do UI manipulations

Let’s think of an interface for this new class.
We already have some functions embedded in ***MainActivity.java*** :

```
private void addNumber(int number)
private void addOperation(Operator operator)
private void doAction(Operator operator)
private void calculate()
```

addNumber(), *addOperation()* and *doAction()* sound like a good public API for our ***CalculatorEngine.java*** while *calculate()* might stay hidden behind.

We also need to communicate back to an interested party (in this case an Activity). We can do it via several methods. In this example, I'll use a simple listener in which the Activity can implement and the calculator engine can use.
Create ***CalculatorEngineListener.java*** interface and add only one method:

```
void updateResult(String value);
```

Your Activity can implement this interface and pass itself down to ***CalculatorEngine.java*** for future observation via it's constructor.

So our final setup might look like this:

CalculatorEngine.java

```
package uk.sky.paveljacko.calculator;

public class CalculatorEngine {

   private String result = "";
   private Float value = 0.0f;
   private Operator lastOperator;
   private boolean needsClear = true;

   private final CalculatorEngineListener listener;

   public CalculatorEngine(CalculatorEngineListener listener) {
      this.listener = listener;
   }

   public void addNumber(int number) {
   }

   public void addOperation(Operator operator) {
   }

   private void calculate() {
   }

   public void doAction(Operator operator) {
   }

   public enum Operator {
      Plus,
      Minus,
      Divide,
      Multiply,
      Equals,
      Comma,
      Clear
  }
}
```

CalculatorEngineListener.java

```
package uk.sky.paveljacko.calculator;

public interface CalculatorEngineListener {
  void updateResult(String value);
}
```

MainActivity.java

```
package uk.sky.paveljacko.calculator;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements CalculatorEngineListener {

  private CalculatorEngine calculatorEngine;
  private TextView textView;

  ButtonHolder[] numbers = new ButtonHolder[]{
        new ButtonHolder(R.id.btn_0, 0),
        new ButtonHolder(R.id.btn_1, 1),
        new ButtonHolder(R.id.btn_2, 2),
        new ButtonHolder(R.id.btn_3, 3),
        new ButtonHolder(R.id.btn_4, 4),
        new ButtonHolder(R.id.btn_5, 5),
        new ButtonHolder(R.id.btn_6, 6),
        new ButtonHolder(R.id.btn_7, 7),
        new ButtonHolder(R.id.btn_8, 8),
        new ButtonHolder(R.id.btn_9, 9)
  };

  ButtonHolder[] operators = new ButtonHolder[]{
        new ButtonHolder(R.id.btn_divide, CalculatorEngine.Operator.Divide),
        new ButtonHolder(R.id.btn_minus, CalculatorEngine.Operator.Minus),
```

```java
            new ButtonHolder(R.id.btn_multiply, CalculatorEngine.Operator.Multiply),
            new ButtonHolder(R.id.btn_equals, CalculatorEngine.Operator.Equals),
            new ButtonHolder(R.id.btn_plus, CalculatorEngine.Operator.Plus)
    };

    ButtonHolder[] actions = new ButtonHolder[]{
            new ButtonHolder(R.id.btn_clear, CalculatorEngine.Operator.Clear),
            new ButtonHolder(R.id.btn_comma, CalculatorEngine.Operator.Comma),
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        textView = (TextView) findViewById(R.id.txt_result);
        calculatorEngine = new CalculatorEngine(this);

        for (final ButtonHolder buttonHolder : numbers) {
            findViewById(buttonHolder.id).setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    calculatorEngine.addNumber(buttonHolder.value);
                }
            });
        }

        for (final ButtonHolder buttonHolder : operators) {
            findViewById(buttonHolder.id).setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    calculatorEngine.addOperation(buttonHolder.operator);
                }
            });
        }

        for (final ButtonHolder buttonHolder : actions) {
            findViewById(buttonHolder.id).setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    calculatorEngine.doAction(buttonHolder.operator);
                }
            });
        }
    }

    @Override
    public void updateResult(String value) {
        textView.setText(value);
    }

    private class ButtonHolder {
        public int id;
        public CalculatorEngine.Operator operator;
        public int value;

        public ButtonHolder(int id, CalculatorEngine.Operator operator) {
            this.id = id;
            this.operator = operator;
        }

        public ButtonHolder(int id, int value) {
            this.id = id;
            this.value = value;
        }
    }
}
```

MainActivity.java has now been simplified and is only responsible for making sure it's views are being updated and clicking of the buttons gets passed down to the calculator engine.

Now it's time to test ***CalculatorEngine.java***, but you might have noticed that it does *nothing*! That's exactly what we want. Remember write your test first, then implement functionality.

Our ***CalculatorEngine.java*** is a black box which has 3 public inputs and 1 output in the form of a listener. We can use any listener as long as it implements the ***CalculatorEngineListener.java*** interface. For the sake of simplicity our ***MainActivity.java*** is an actual listener.
Calling public API methods is easy, but how exactly are we supposed to test the actual listener? We have a couple of options:

- Fakes
- Mocks

Mockito

Mocking is one of the most powerful concepts available in TDD. It allows you to mock and inject dependencies without altering the tested class internals. This acts as a little endoscope allowing the doctor to see what's going on in a patient's body without performing horrifying butchering up front, unless of course, he's a Jack the Ripper type of character, in which case I would withdraw the word "horrifying".

Mockito is my favourite "endoscope" in this case.

In order to use it, we have to add it to our ***build.gradle*** dependencies:

```
testCompile "org.mockito:mockito-core:1.+"
```

Once we have synced, let's write our first test:

```
package uk.sky.paveljacko.calculator;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.runners.MockitoJUnitRunner;

@RunWith(MockitoJUnitRunner.class)
public class CalculatorEngineTest {

   @Mock
   private CalculatorEngineListener listener;

   private CalculatorEngine sut;

   @Before
   public void startUp() {
       sut = new CalculatorEngine(listener);
   }

   @Test
   public void testAddNumber() throws Exception {
       //when
       sut.addNumber(5);
       Mockito.verify(listener).updateResult("5");
       sut.addNumber(6);
       Mockito.verify(listener).updateResult("56");
       sut.addNumber(1);
       Mockito.verify(listener).updateResult("561");
   }

}
```

If you run this test (right click on ***CalculatorEngineTest.java*** and press run test…) you will get a failure simillar to this:

```
Wanted but not invoked:
listener.updateResult("5");
```

Adding Functionality

Now let's add some functionality to the empty method we're trying to test. Let's replace it with:

```
public void addNumber(int number) {
   if(needsClear) {
       result = "";
       needsClear = false;
   }
   result += number;
   listener.updateResult(result);
}
```

now run it again and see the result:

```
Process finished with exit code 0
```

We’ll add more tests and cover with additional missing functionality until we reach state where we were before we applied TDD. A functional app.

The final code of our test would be:

CalculatorEngineTest.java

```
package uk.sky.paveljacko.calculator;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.runners.MockitoJUnitRunner;

@RunWith(MockitoJUnitRunner.class)
public class CalculatorEngineTest {

   @Mock
   private CalculatorEngineListener listener;

   private CalculatorEngine sut;

   @Before
   public void startUp() {
       sut = new CalculatorEngine(listener);
   }

   @Test
   public void testAddNumber() throws Exception {
       sut.addNumber(5);
       Mockito.verify(listener).updateResult("5");
       sut.addNumber(6);
       Mockito.verify(listener).updateResult("56");
       sut.addNumber(1);
       Mockito.verify(listener).updateResult("561");
   }

   @Test
   public void testPlusOperator() throws Exception {
       sut.addNumber(6);
       Mockito.verify(listener).updateResult("6");
       sut.addOperation(CalculatorEngine.Operator.Plus);
       sut.addNumber(4);
       Mockito.verify(listener).updateResult("4");
       sut.addOperation(CalculatorEngine.Operator.Equals);
       Mockito.verify(listener).updateResult("10.0");
   }

   @Test
   public void testMinusOperator() throws Exception {
       sut.addNumber(7);
       Mockito.verify(listener).updateResult("7");
       sut.addOperation(CalculatorEngine.Operator.Minus);
       sut.addNumber(3);
       Mockito.verify(listener).updateResult("3");
       sut.addOperation(CalculatorEngine.Operator.Equals);
       Mockito.verify(listener).updateResult("4.0");
```

```
    }

    @Test
    public void testMultiplyOperator() throws Exception {
        sut.addNumber(2);
        Mockito.verify(listener).updateResult("2");
        sut.addOperation(CalculatorEngine.Operator.Multiply);
        sut.addNumber(8);
        Mockito.verify(listener).updateResult("8");
        sut.addOperation(CalculatorEngine.Operator.Equals);
        Mockito.verify(listener).updateResult("16.0");
    }

    @Test
    public void testDivideOperator() throws Exception {
        sut.addNumber(1);
        Mockito.verify(listener).updateResult("1");
        sut.addNumber(5);
        Mockito.verify(listener).updateResult("15");
        sut.addOperation(CalculatorEngine.Operator.Divide);
        sut.addNumber(2);
        Mockito.verify(listener).updateResult("2");
        sut.addOperation(CalculatorEngine.Operator.Equals);
        Mockito.verify(listener).updateResult("7.5");
    }

    @Test
    public void testCommaOperator() throws Exception {
        sut.addNumber(1);
        Mockito.verify(listener).updateResult("1");
        sut.doAction(CalculatorEngine.Operator.Comma);
        Mockito.verify(listener).updateResult("1");
        sut.addNumber(5);
        Mockito.verify(listener).updateResult("1.5");
    }

    @Test
    public void testClearOperator() throws Exception {
        sut.addNumber(1);
        Mockito.verify(listener).updateResult("1");
        sut.addNumber(1);
        Mockito.verify(listener).updateResult("11");
        sut.doAction(CalculatorEngine.Operator.Clear);
        Mockito.verify(listener).updateResult("0");
    }
}
```

And

CalculatorEngine.java

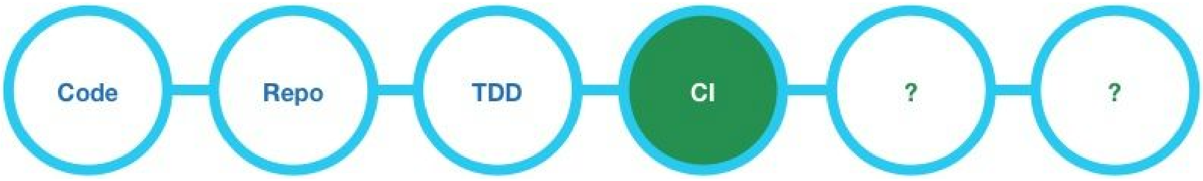
```
package uk.sky.paveljacko.calculator;

public class CalculatorEngine {

    private String result = "";
    private Float value = 0.0f;
    private Operator lastOperator;
    private boolean needsClear = true;

    private final CalculatorEngineListener listener;
```

```java
public CalculatorEngine(CalculatorEngineListener listener) {
    this.listener = listener;
}

public void addNumber(int number) {
    if(needsClear) {
        result = "";
        needsClear = false;
    }
    result += number;
    listener.updateResult(result);
}

public void addOperation(Operator operator) {
    needsClear = true;
    calculate();
    lastOperator = operator;
    listener.updateResult(value.toString());
}

private void calculate() {
    if(lastOperator == null) {
        value = Float.parseFloat(result);
        return;
    }

    switch (lastOperator) {
        case Plus:
            value += Float.parseFloat(result);
            break;
        case Minus:
            value -= Float.parseFloat(result);
            break;
        case Multiply:
            value *= Float.parseFloat(result);
            break;
        case Divide:
            value /= Float.parseFloat(result);
            break;
    }
}

public void doAction(Operator operator) {
    switch (operator) {
        case Comma:
            result += ".";
            break;
        case Clear:
            result = "0";
            needsClear = true;
            value = 0.0f;
            lastOperator = null;
            listener.updateResult(result);
            break;
    }
}

public enum Operator {
    Plus,
    Minus,
    Divide,
    Multiply,
    Equals,
    Comma,
    Clear
```

```
  }
}
```

Re-run all tests, commit this to your repo and have a nice cup of coffee/tea/beer/lucozade, you've passed half way through!

"Hero team stabilized their app, but as more developers started committing at the same time, tests started failing on developers' machines after synchronization."



Continuous Integration

The biggest issue of collaborating developers is surprisingly collaboration itself. How can you make sure that changes introduced by one developer and similar changes done by other developer at the same time will work well together?
It's time to fix this, and to do that we need some help. Say 'hello' to Continuous Integration. The idea is that every commit to your branch would trigger a CI build which executes your tests and, therefore, your code is, well… "continuously integrated".

Tools

There are plenty of CI options from local ones like:
Jenkins, Hudson, Bamboo
To Hosted ones like:
CircleCI, Travis, Ship.io and so on.
Full list:
https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software

All these hosted services are fairly new and choosing the right one probably comes down to your requirements, current state of the tool as well as some personal preferences.
I've done a lot with Jenkins in the past, but recently I've got tired maintaining it. What can I say.. I'm getting older and lazier.

I've been using CircleCI for some time and I like it's "no maintenance, no hassle" benefits so that I can concentrate on what I get paid to do (development).

CircleCI

CircleCI offers a fairly good amount of flexibility, but most importantly these 2 key characteristics:

- Reliability
- Scalability

In our current Sky project, we managed to reduce the Jenkins build time from 1 hour to 15 minutes by using multiple containers and clever yml configurations.

CircleCI setup

Go to https://circleci.com/ and sign up.

Here is the cool part...you can log in using your github account instead of creating a new account.

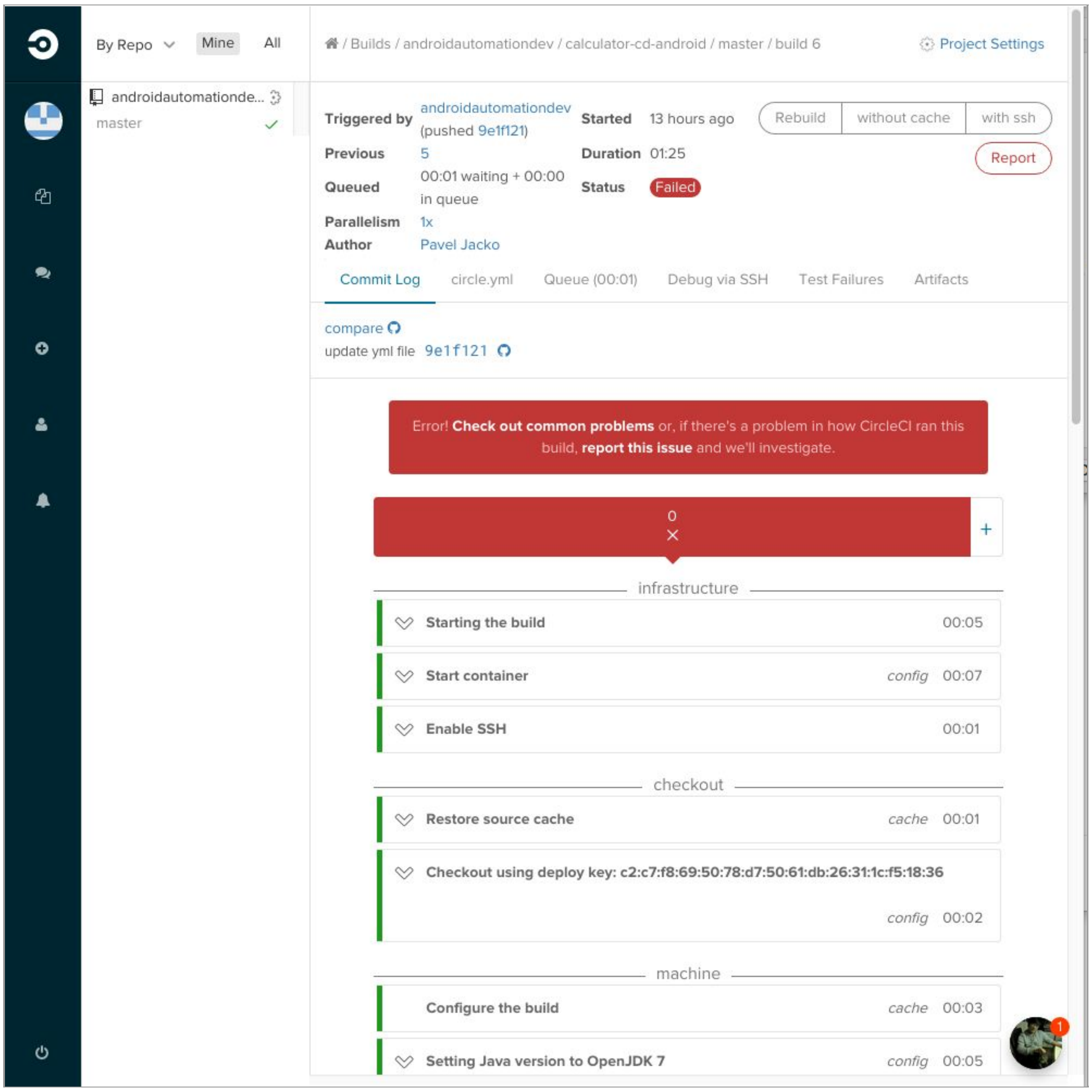
Connect CircleCI with Github:

```
CircleCI => Authorize with CircleCI => Authorize
```

Once logged in, you can set your current master branch to build by CircleCI after each commit:

```
CircleCI => Add new projects => Select your account => Build project
```

Once done you should see something like this:



Triggered by: androidautomationdev (pushed 9e1f121)
Started: 13 hours ago
Previous: 5
Duration: 01:25
Queued: 00:01 waiting + 00:00 in queue
Status: Failed
Parallelism: 1x
Author: Pavel Jacko

compare
update yml file 9e1f121

Error! **Check out common problems** or, if there's a problem in how CircleCI ran this build, **report this issue** and we'll investigate.

0

infrastructure

```
Starting the build                                                  00:05
Start container                                             config  00:07
Enable SSH                                                          00:01
```

checkout

```
Restore source cache                                        cache   00:01
Checkout using deploy key: c2:c7:f8:69:50:78:d7:50:61:db:26:31:1c:f5:18:36
                                                            config  00:02
```

machine

```
Configure the build                                         cache   00:03
Setting Java version to OpenJDK 7                           config  00:05
```

Your project will fail initially because we’re missing the yml file and CircleCI is trying to use the default one which is not what we need.

YML File

The yml file will act as a configuration file for CircleCI.

Let’s override it with our own implementation.
In your root project directory create the file ***circle.yml*** with this content:

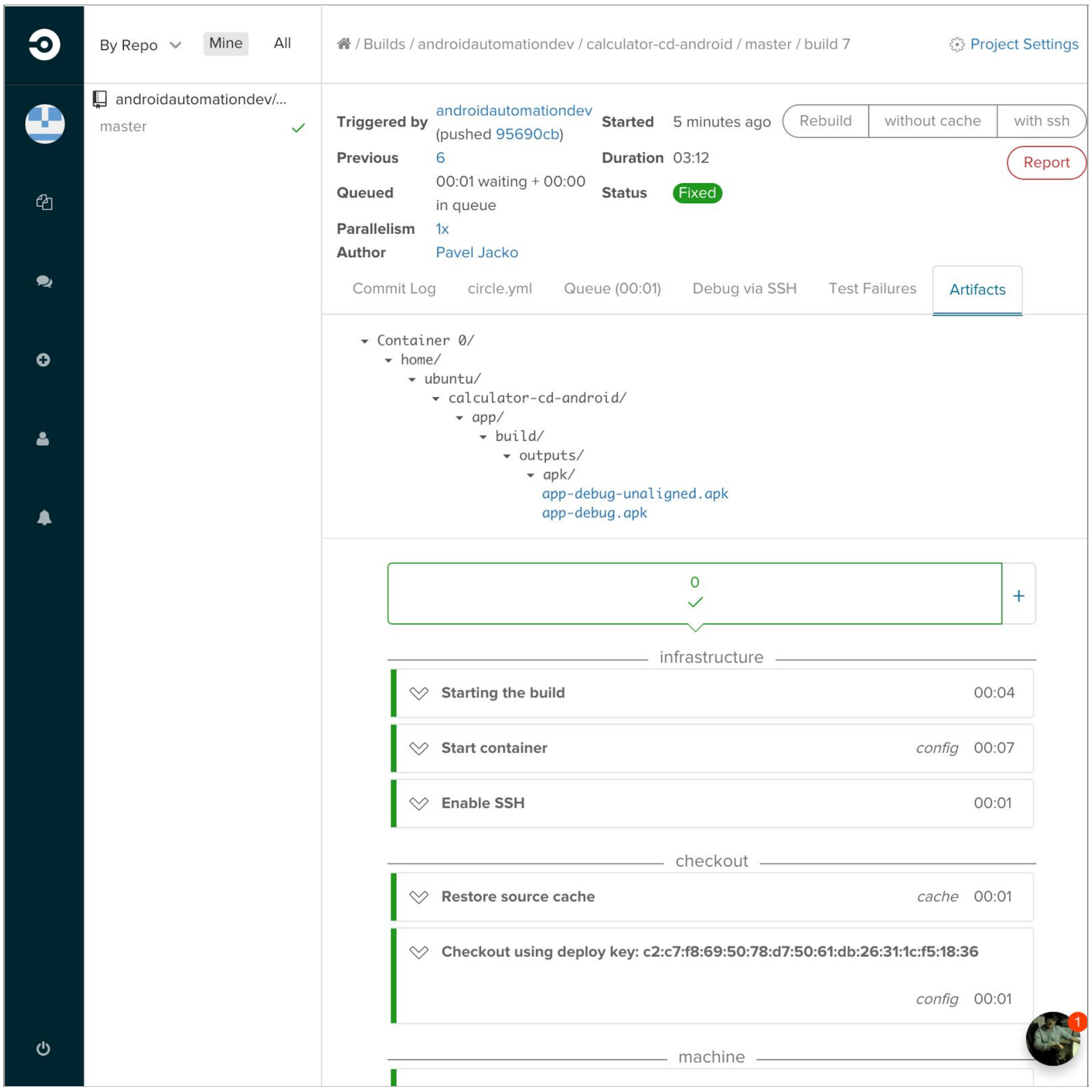
```
machine:
 java:
   version: openjdk7
 ruby:
  version: 2.1.1
 environment:
   GRADLE_OPTS: '-Dorg.gradle.jvmargs="-Xmx2048m machine:
  java:
   version: openjdk7
  environment:
   GRADLE_OPTS: '-Dorg.gradle.jvmargs="-Xmx2048m -XX:+HeapDumpOnOutOfMemoryError"'

dependencies:
  pre:
   - echo y | android update sdk --no-ui --all --filter
platform-tools,tools,android-23,build-tools-23.0.1,extra-android-m2repository,extra-android-support,extra-goog
le-m2repository

   - chmod +x gradlew

  cache_directories:
   - ~/.android
   - ~/.gradle
  override:
   - ./gradlew dependencies

test:
  override:
   - ./gradlew test --stacktrace;
   - ./gradlew assembleDebug --stacktrace;

general:
  artifacts:
   - "**/build/outputs/apk"
   - "**/build/outputs/mapping/**/**/mapping.txt"
```

In short, this file does the following:

- updates all dependencies
- runs all the tests
- saves the build artifacts (apk files)

Then commit and push your changes to the master branch.
CircleCI will automatically trigger its master branch which should pass successfully.

The outcome should look similar to this:


By Repo Mine All

/ Builds / androidautomationdev / calculator-cd-android / master / build 7 Project Settings

androidautomationdev/...
master ✓

Triggered by	androidautomationdev (pushed 95690cb)	Started	5 minutes ago
Previous	6	Duration	03:12
Queued	00:01 waiting + 00:00 in queue	Status	Fixed
Parallelism	1x		
Author	Pavel Jacko		

Rebuild | without cache | with ssh

Report

Commit Log | circle.yml | Queue (00:01) | Debug via SSH | Test Failures | Artifacts

```
▾ Container 0/
  ▾ home/
    ▾ ubuntu/
      ▾ calculator-cd-android/
        ▾ app/
          ▾ build/
            ▾ outputs/
              ▾ apk/
                app-debug-unaligned.apk
                app-debug.apk
```

0 ✓

infrastructure

Starting the build		00:04
Start container	config	00:07
Enable SSH		00:01

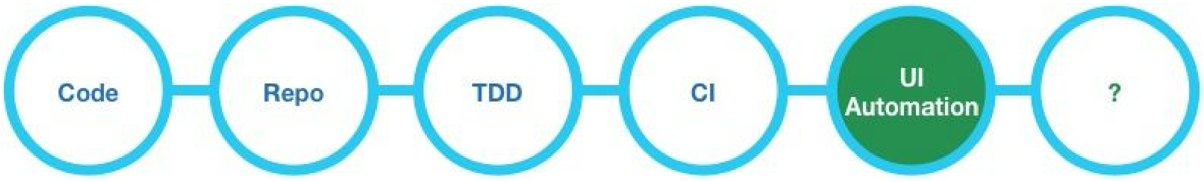
checkout

Restore source cache	cache	00:01
Checkout using deploy key: c2:c7:f8:69:50:78:d7:50:61:db:26:31:1c:f5:18:36	config	00:01

machine

And there you go, you now have Continuous Integration running and every commit will be evaluated by an automated CI job.

“Unit Tests were passing and the quest continued, but once the app was delivered to stakeholders it would not behave as Hero’s demanding manager wished.”



UI Automation

Let’s say your Product Owner wanted to make sure that when a user interacts with the calculator, that the result display would state *“Result is: 80.0”* instead of *“80.0”* (for whatever weird reason) and he is furious! How could this be missed?
The issue is that this was never covered by unit tests. This is a very UI specific message and had nothing to do with our ***CalculatorEngine.java*** functionality.

In order to cover User journeys we want to setup something which would simulate our demanding manager and all our potential customers. That requires a different type of testing. The one which interacts with final UI and observe results on UI “customer” scale rather than on internal component “unit” scale.

Welcome to the world of UI automation.

Let’s make it clear from the start: UI automation is hard. Not the writing of it, but everything else surrounding it including maintenance, collaboration, dependencies, change of requirements and so on.
I would compare it to an old light switch. You have to remember one cardinal rule: “*It is either working 100% or 0%*”. There is no inbetween, no compromise, no “just the little”. Your switch is either on or off. If you commit your efforts to automation and you don’t use this rule, you will waste time, effort and money.
This is definitely the most challenging part of Continuous Delivery.

Tools

Appium, Calabash, Android Espresso

Hosted Providers:
Sauce labs, Xamarin test cloud, AWS, Perfecto Mobile

The first and most important question you need to ask yourself is: “What are you trying to do with UI automation?”

Save development time? Build a stable app? Eliminate manual testing?

I see one main reason to introduce UI automation and the clue can be found in its name. It’s to “automate”

Don’t forget one of the requirements we had at the beginning:

- Every commit to master should be deployable to stakeholders without manual steps

As Ford's introduction of the assembly line revolutionized the automotive industry, UI automation can revolutionize the process and speed of deploying apps to your customers.

Faster deployment = faster feedback = faster response.

Just like Darwin's theory of evolution says, it's the ability to adapt to an ever changing environment that ensures survival.

Thus it's the most responsive businesses which have the best chances of survival in the jungle of ever changing technology evolution and competition.

By automating most of the usual time consuming journeys across the app you can shrink your manual testing to exploratory testing at the point of major releases. Any hotfix or small change could be dealt with automation alone.

While this is, of course, a theory, in practice this is much harder to achieve. Nevertheless, we'll try to attempt to introduce UI automation into our app.

Android Espresso

What is it? Well according to uncle google:
The Espresso testing framework, provided by the Android Testing Support Library, provides APIs for writing UI tests to simulate user interactions within a single target app.

Before we start using it, we need to add new dependencies to our **build.gradle** file:

```
androidTestCompile 'com.android.support:support-annotations:23.0.1'
androidTestCompile 'com.android.support.test:runner:0.4.1'
androidTestCompile 'com.android.support.test:rules:0.4.1'
androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.1'
```

and update our config section:

```
defaultConfig {
  applicationId "uk.sky.paveljacko.calculator"
  minSdkVersion 16
  targetSdkVersion 23
  versionCode 1
  versionName "1.0"

  testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
}
```

Now it's time to add a new class to our src/androidTest folder. Let's call it:
UseCalculatorBehaviorTest.java

NOTE: Android Studio might have already created a class for you called:

ApplicationTest.java. You can delete it.

Let’s add some initial code to it:

```
package uk.sky.paveljacko.calculator;

import android.support.test.rule.ActivityTestRule;
import android.support.test.runner.AndroidJUnit4;
import android.test.suitebuilder.annotation.LargeTest;

import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;

import static android.support.test.espresso.Espresso.onView;
import static android.support.test.espresso.assertion.ViewAssertions.matches;
import static android.support.test.espresso.matcher.ViewMatchers.withId;
import static android.support.test.espresso.matcher.ViewMatchers.withText;


@RunWith(AndroidJUnit4.class)
@LargeTest
public class UseCalculatorBehaviorTest {

   public static final String INITIAL_TEXT = "1";

   @Rule
   public ActivityTestRule<MainActivity> mActivityRule = new ActivityTestRule<>(
           MainActivity.class);

   @Test
   public void checkInitalValue() {
      onView(withId(R.id.txt_result)).check(matches(withText(INITIAL_TEXT)));
   }
}
```

Now if you run it (right click => run…) you will see the actual test run, but it will fail with a message:

```
android.support.test.espresso.base.DefaultFailureHandler$AssertionFailedWithCauseError:
'with text: is "1"' doesn't match the selected view.
Expected: with text: is "1"
```

Now if you have a keen eye you might spot the reason why:

```
public static final String INITIAL_TEXT = "1";
```

That’s not what we want. We expect “0”. Let’s update it and run it again:

```
Success
Running tests
Test running startedFinish
```

Good. We now have the correct UI expectation and app which satisfies that; however, there is one requirement which has not been satisfied.

Going back to the failed product requirement:
“Product Owner wanted to make sure that when a user interacts with the calculator, that the result display would state “Result is: 80.0” instead of “80.0”

So let’s update our automation to match it. Add a new test to your ***UseCalculatorBehaviorTest.java***:

```
@Test
public void checkOutcomeValue() {
   onView(withId(R.id.btn_1)).perform(click());
   onView(withId(R.id.btn_5)).perform(click());
   onView(withId(R.id.btn_multiply)).perform(click());
   onView(withId(R.id.btn_2)).perform(click());
   onView(withId(R.id.btn_equals)).perform(click());
   onView(withId(R.id.txt_result)).check(matches(withText("Result is: 30.0")));
}
```

Running this will result in an expected failure. Now it’s time to update our ***MainActivity.java*** with additional code. Replace *updateResult* method content with this new line:

```
@Override
public void updateResult(String value) {
   textView.setText("Result is: " + value);
}
```

Now you run it and bingo! All is green. Go and grab yourself a refreshment.

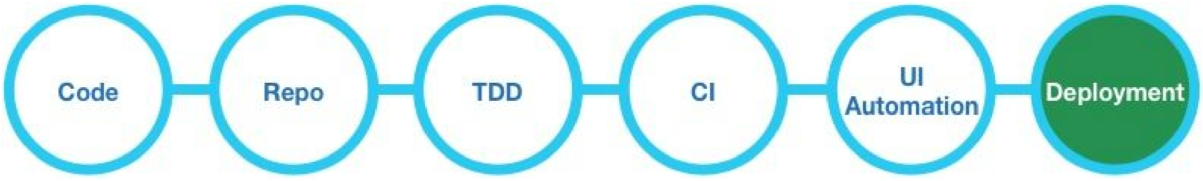
CircleCI Automation

Now before we commit this, we want to make sure the same tests can be executed on CircleCI. So it’s time to update the **circle.yml** file. Override the test section with this code:

```
test:
 override:
   - $ANDROID_HOME/tools/emulator -avd testing -no-window -no-boot-anim -no-audio:
                  background: true
                  parallel: true
   - ./gradlew test --stacktrace;
   - ./gradlew assembleDebug --stacktrace;
   - ./gradlew assembleDebugAndroidTest --stacktrace;
```

Now it’s time to commit=>push and if everything goes well you should see the build pass green. And we’re almost there!

“Hero team managed to build a stable app, commit often and according to product requirements, but delivering the app to stakeholders was still time consuming”



Deployment

So how can we set up something which would allow us to deploy our app directly to our stakeholders, trialists, or anybody interested without developers performing this task manually?

Tools

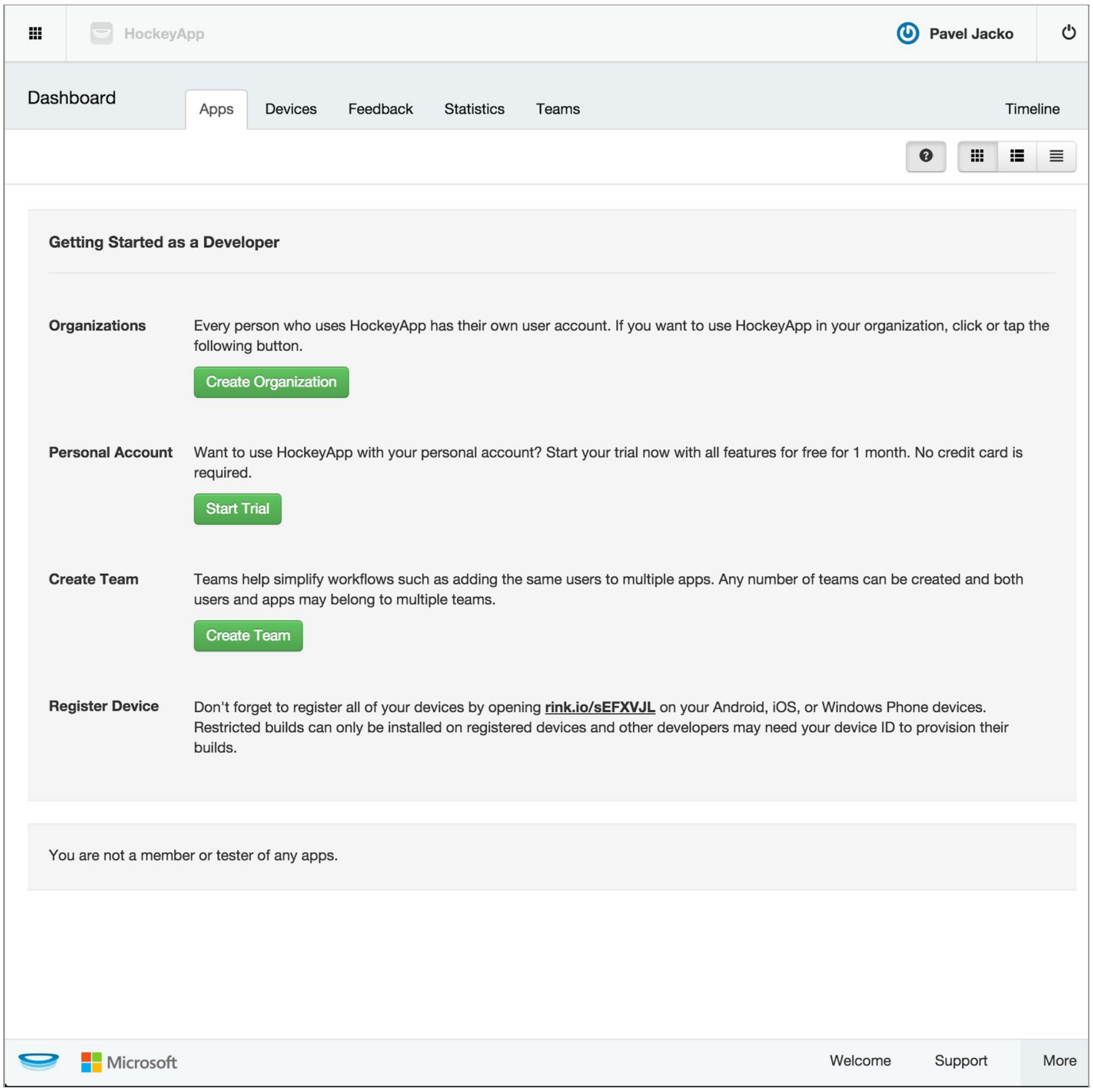
There are couple of tools available to do this job:
HockeyApp, Google Play Alpha/Beta, Ship.io, Fabric, Appaloosa, TestFairy, Apphance etc.
They all vary in features and pricing.

Hockeyapp

I’ve been using Hockeyapp for some time and so far it suits my and my team’s needs.
It’s easy to setup, manage, deployments are almost instantaneous and it supports both Android and iOS.

Setup Hockeyapp account

Navigate to: http://hockeyapp.net/ and sign up
Once you pass all the setup screens you should be able to get to this initial dashboard:


HockeyApp Pavel Jacko

Dashboard Apps Devices Feedback Statistics Teams Timeline

Getting Started as a Developer

Organizations Every person who uses HockeyApp has their own user account. If you want to use HockeyApp in your organization, click or tap the following button.

Create Organization

Personal Account Want to use HockeyApp with your personal account? Start your trial now with all features for free for 1 month. No credit card is required.

Start Trial

Create Team Teams help simplify workflows such as adding the same users to multiple apps. Any number of teams can be created and both users and apps may belong to multiple teams.

Create Team

Register Device Don't forget to register all of your devices by opening **rink.io/sEFXVJL** on your Android, iOS, or Windows Phone devices. Restricted builds can only be installed on registered devices and other developers may need your device ID to provision their builds.

You are not a member or tester of any apps.

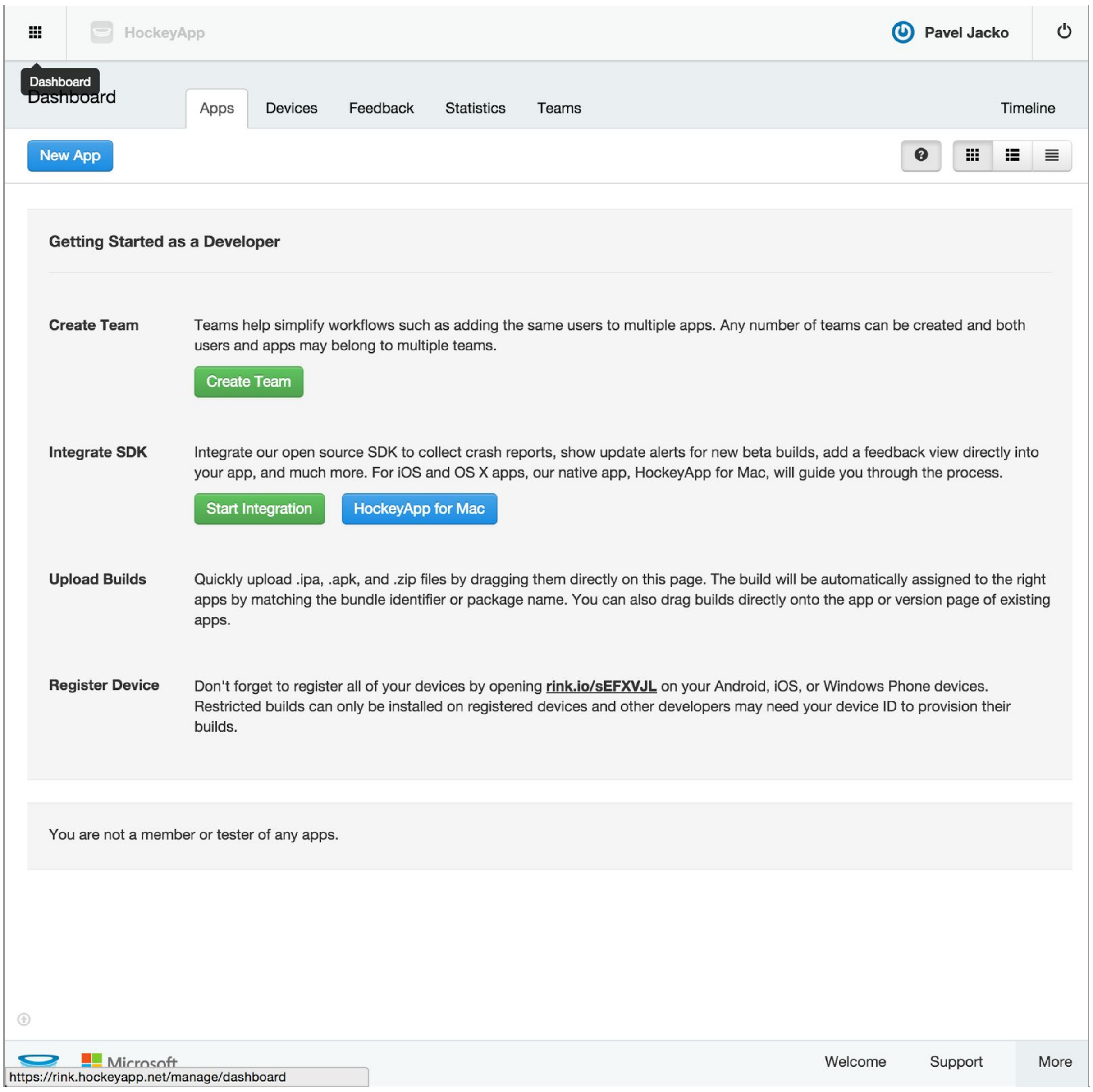
Microsoft Welcome Support More

Create a new Team (you can called it *“Stakeholders”* for example):

HockeyApp => Dashboard => Create Team => invite members => Create

Create a new Organization:

HockeyApp => Dashboard => Create Organization => Save



Getting Started as a Developer

Create Team Teams help simplify workflows such as adding the same users to multiple apps. Any number of teams can be created and both users and apps may belong to multiple teams.

Create Team

Integrate SDK Integrate our open source SDK to collect crash reports, show update alerts for new beta builds, add a feedback view directly into your app, and much more. For iOS and OS X apps, our native app, HockeyApp for Mac, will guide you through the process.

Start Integration HockeyApp for Mac

Upload Builds Quickly upload .ipa, .apk, and .zip files by dragging them directly on this page. The build will be automatically assigned to the right apps by matching the bundle identifier or package name. You can also drag builds directly onto the app or version page of existing apps.

Register Device Don't forget to register all of your devices by opening rink.io/sEFXVJL on your Android, iOS, or Windows Phone devices. Restricted builds can only be installed on registered devices and other developers may need your device ID to provision their builds.

You are not a member or tester of any apps.

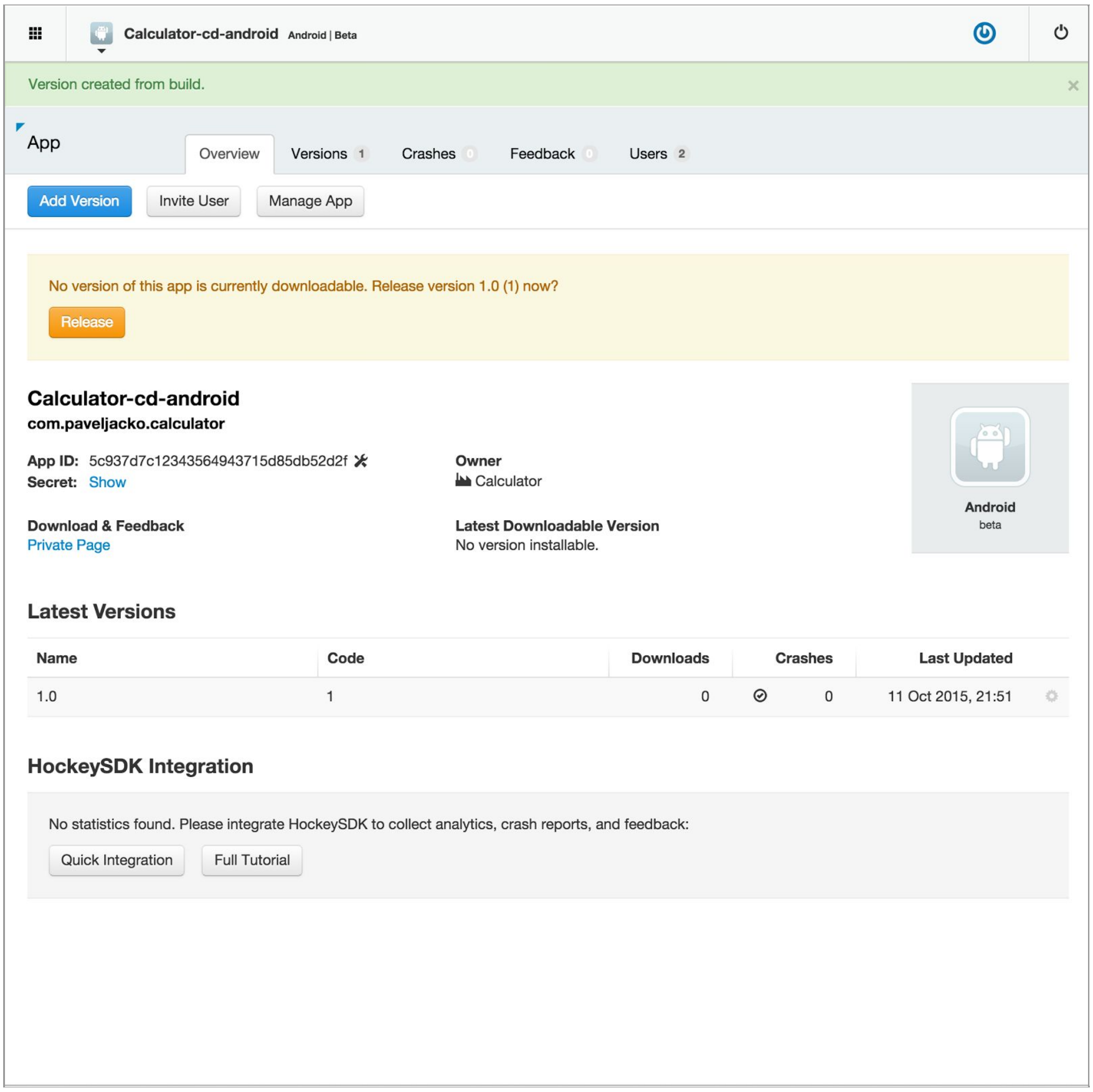
Now go to your CircleCI builds and download your latest build. It should be called *app-debug.apk*:

CircleCI => pick latest build => Artifacts => app-debug.apk

Now go back to Hockeyapp, hit the “New app” button and drag and drop your apk over there:

HockeyApp => Dashboard => New App => drag and drop downloaded apk => Next

Once uploaded, you can perform a release.


Calculator-cd-android Android | Beta

Version created from build.

App

Overview | Versions 1 | Crashes 0 | Feedback 0 | Users 2

Add Version | Invite User | Manage App

No version of this app is currently downloadable. Release version 1.0 (1) now?

Release

Calculator-cd-android
com.paveljacko.calculator

App ID: 5c937d7c12343564943715d85db52d2f
Secret: Show

Owner
Calculator

Download & Feedback
Private Page

Latest Downloadable Version
No version installable.

Android
beta

Latest Versions

Name	Code	Downloads	Crashes	Last Updated
1.0	1	0	0	11 Oct 2015, 21:51

HockeySDK Integration

No statistics found. Please integrate HockeySDK to collect analytics, crash reports, and feedback:

Quick Integration | Full Tutorial

Downloading the app

Getting the app from Hockeyapp is extremely easy, even for a non technical person (ahem, that is once they manage to register).

Open the https://hockeyapp.net/apps in your phone browser and download the Android version. Sign in/up with one of the invited emails you've added to your team in the previous step and accept the invitation.

Versioning

Those with a keen eye may have noticed one important value. “Code” which currently stands at 1. This is a very important property. In order to allow people to install a new version of an app on top of \an old one, each new app has to have a higher version code.

If you compare this to your ***build.gradle*** file you'll notice the same version information:

```
defaultConfig {
    applicationId "uk.sky.paveljacko.calculator"
    minSdkVersion 16
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"
}
```

Sure, you could update this number manually every time you're about to do a commit to master but that go against our first requirement:

"Every commit to master should be deployable to stakeholders without manual steps"

So we need bit more clever setup. This is where gradle and git comes to rescue. Idea is that each commit in git is countable and that sounds like perfect auto increment tool.

we'll add this code to the bottom of our app/build.gradle:

```
def computeVersionCode() {
    try {
        def p = Runtime.getRuntime().exec("git rev-list --all --count")

        def result = p.waitFor()
        if (result != 0) {
            return 0 // no git revisions
        }
        return p.getInputStream().readLines().get(0).toInteger()
    } catch (ignored) {
        return 0;
    }
}

def computeVersionName() {
    def command = Runtime.getRuntime().exec("git rev-parse --short HEAD")
    def result = command.waitFor()
    return (result == 0) ? computeVersionCode() + "-" + command.inputStream.text.trim() : "nogit"
}
```

and the replace to versioning lines at the top of the file:

```
versionCode computeVersionCode()
versionName computeVersionName()
```

Additionally, to test it before we upload it to hockeyapp let's add little logging at the bottom:

```
task printVersion() {
    print "\n**************** APP BUILD VERSION ***********\n";
    print "Version Code: " + computeVersionCode() + "\n";
    print "Version Name: " + computeVersionName() + "\n";
    print "**************************************************\n\n"
}
```

Now you can run any gradle command and you should see our initial log message similar:

```
$ ./gradlew test

**************** MOBILE BUILD VERSION ***********
Version Code: 7
Version Name: 7-95690cb
**************************************************
```

My log indicates that I've done 7 commits on the master. each commit will bump up the version code. Version code must be an integer but version name is string so we also included part of the commit hash to be able to easily identify commit in the git history.

Ok, so far all good and we have last step in front of us. we need to automate the whole upload process.

Connecting with CircleCI

At first, we need to generate HockeyApp token, go to :

```
HockeyApp => Account Settings => API Tokens => Create
```

This should generate an (very long digit) API token number for you at the bottom of the page.

Now copy that and go back to your CircleCI:

```
CircleCI => Project Settings => Environment Variables
Name: HOCKEYAPP_TOKEN
Value: [add your api token value here]
=> Save variables
```

Now it's time to add support for Hockeyapp deployment into your build:

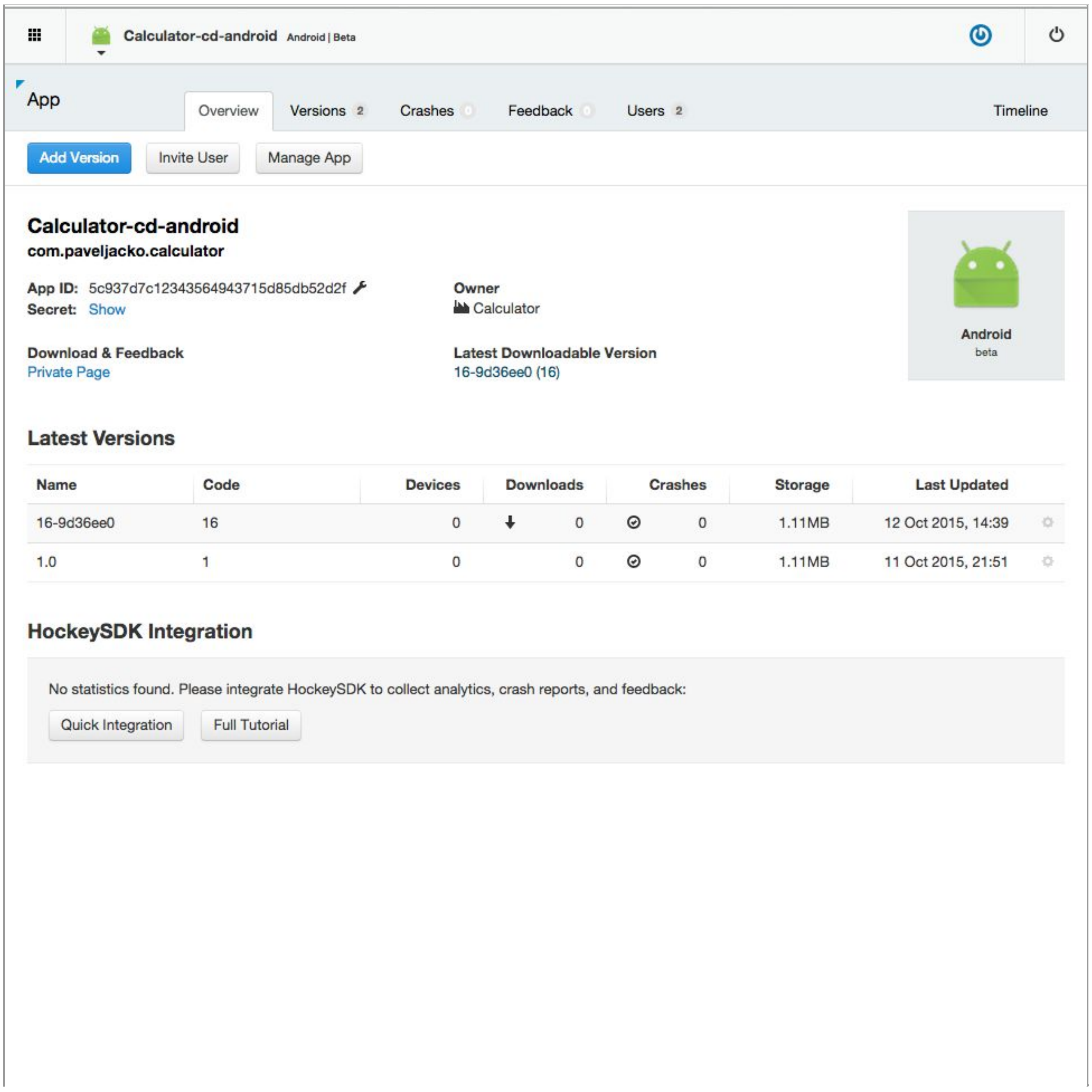
To do this we'll make some updates to our ***circle.yml*** file. To make things more readable we'll move hockeyapp script into separate file.
Create ***deploy.sh*** file in the root project directory and paste this code into it:

```
PRODUCT_NAME=hockey
NOTIFY="True"
NOTES="Build uploaded via the upload API"

echo "Downloading File..."
echo "Archives: ${CIRCLE_ARTIFACTS}"

if [ "$1" ]
then
  NOTES="$1"
fi

if [ ! -f "app/build/outputs/apk/app-debug.apk" ]
then
  echo "app/build/outputs/apk/app-debug.apk not found!"
else
  echo "Uploading to HockeyApp..."
  /usr/bin/curl "https://rink.hockeyapp.net/api/2/apps/upload" \
       -F ipa=@"app/build/outputs/apk/app-debug.apk" \
      -F "status=2" \
       -F "notify=1" \
       -F "notes=Some new features and fixed bugs." \
       -F "notes_type=0" \
       -H "X-HockeyAppToken: ${HOCKEYAPP_TOKEN}" \
fi
```

Then add additional lines at the bottom of your ***circle.yml*** file:

```
deployment:
 master:
   branch: master
   commands:
     - ./deploy.sh
```

Now, if you commit/push this back to master repo and wait couple of minutes you should be able to see magic in action. If you navigate to hockeyapp you should be able to see new app version:



Calculator-cd-android
com.paveljacko.calculator

App ID: 5c937d7c12343564943715d85db52d2f
Secret: Show

Owner
Calculator

Download & Feedback
Private Page

Latest Downloadable Version
16-9d36ee0 (16)

Latest Versions

Name	Code	Devices	Downloads	Crashes	Storage	Last Updated
16-9d36ee0	16	0	0	0	1.11MB	12 Oct 2015, 14:39
1.0	1	0	0	0	1.11MB	11 Oct 2015, 21:51

HockeySDK Integration

No statistics found. Please integrate HockeySDK to collect analytics, crash reports, and feedback:

And once you've done that your stakeholders can see a new app through their Hockeyapp console and download it straight away.

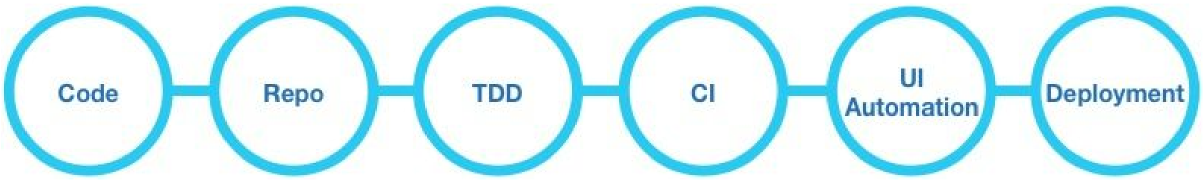
Every new commit to master will automatically trigger the whole process. If it fails, no deployment will happen and developers would be forced to fix the issue as soon as possible. This forces the team to maintain a healthy codebase at all times.

To recap, we went from updating code and committing it to having it available to stakeholders within 10 minutes through unit tests and UI automation coverage.

I have good news for you:

1. We've completed our quest!
2. The fact that you are still reading this means you finished mentally unharmed.

"Hero and his team managed to reach the holy grail that they were looking for - the mighty Continuous Deployment! Hero's team got pay rises. Hero managed to get married during his annual paid holiday in Las Vegas, and they all lived happily thereafter..at least until the next recession"



Conclusion

Continuous Delivery is not easy.
It reminds me of an F1 racing car. It's fast, it's powerful, it can deliver great results, but all that comes at a price.
You have to maintain it and treat it with the respect it deserves.
And while it's powerful, it's also very fragile. You can't just put your keys into an ignition and drive it off to your local Tesco grocery shop. You need to know what you're doing and you need to know how to tweak it just right for your needs and current environment.
If you do it right, it will reward you with the ability to build products and release them to your potential customers at the fraction of traditional development lifecycle times. It will also greatly contribute towards making your team truly agile *(having a daily standup ≠ being agile)*.
If you do not commit to it fully; however, you might just end up wasting a lot of precious time on an imaginary holy grail which no matter how daunting it sounds, delivers only failed promises.

Notes

This is a very simplistic scenario to demonstrate a basic idea of Continuous Delivery with a working example.
It's real benefits will become more obvious as you start scaling your team and app.

As your app grows you might start to think about implementing tools which could support larger project demands such as:

- Git Flow setup for sub branching
- Using fragments
- Applying MVC/MVP/MVVM architecture

- Using crash logging tools like fabric or even Hockeyapp
- Analytics
- AB Testing
- Google play deployment integration
- Introduce BDD and Gherkin format to your tests
- Dependency Injection tools (Dagger 2)
- More enterprise Automation tool (Calabash, Appium)
- Slack: message integration via CircleCI
- ..and an infinite number of other tools/frameworks/design patterns

Sky has it's well known slogan "Believe in Better", but as far as I know the universe's fate is sealed and pretty grim, so I would probably change it to "**Believe in Continuous Delivery**".

Sources are available here:
https://github.com/beherithrone/calculator-cd-android

This story and its characters are definitely fictional (as far as I know).

Author: Pavel Jacko

Special thanks to Rhonda Hoonjan for editorial sanitisation.