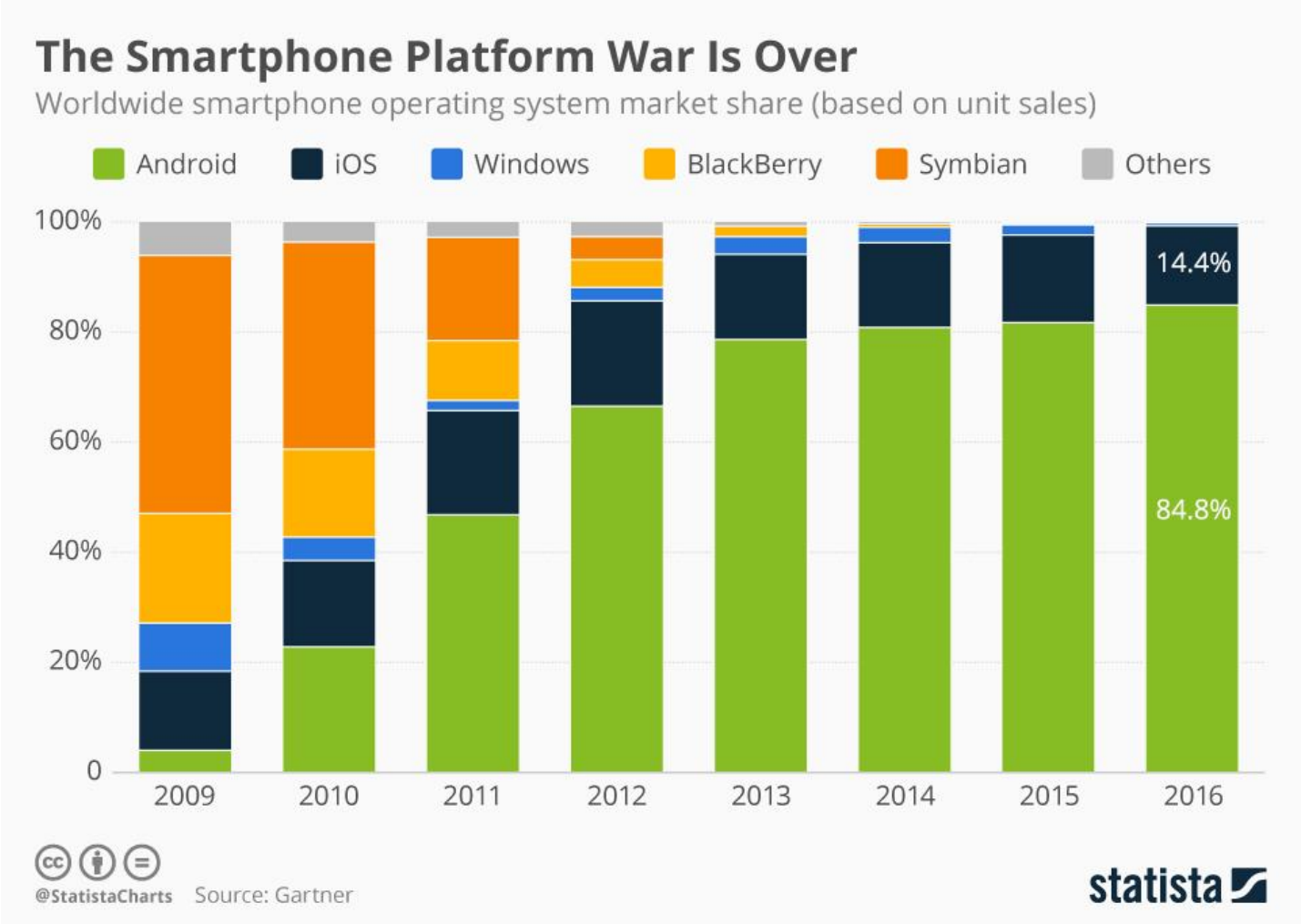
V&V support for Android application development

David Sik

BME AUT
Department of Automation and Applied Informatics

Introduction

Smartphone market

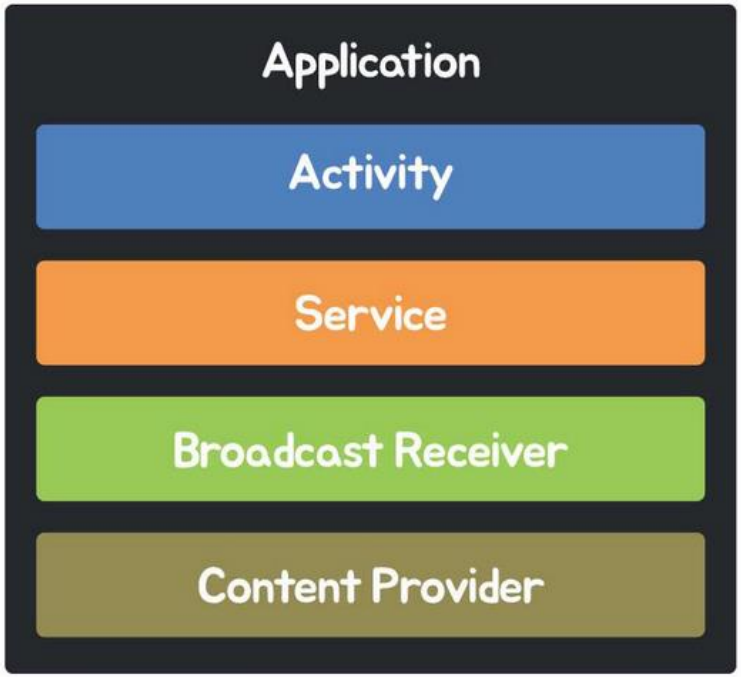


Android

- Google, 2008
- Open, Linux-based OS
  - Complex architecture
  - Linux kernel
  - Dalvik VM
- Play Store
  - 3.3 million app
- Lot of manufacturers, devices
  - Phone, tablet, watch

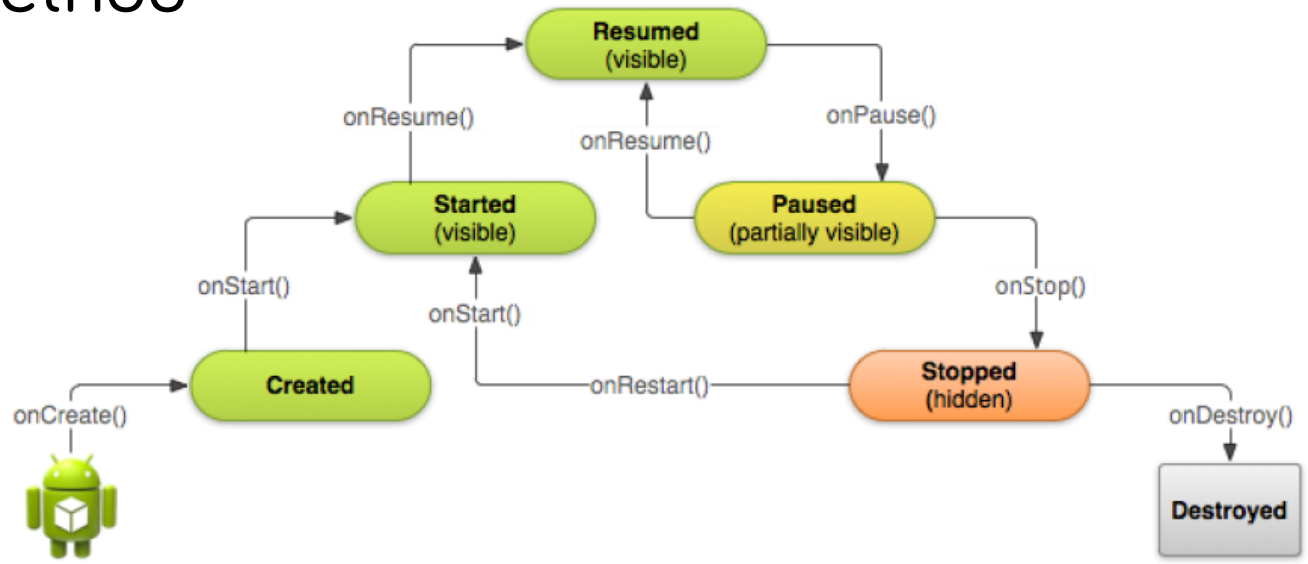
Android application

- Component(s)
  - Activity
  - Service
  - Broadcast Receiver
  - Content Provider
- Communication between the components
- Java classes
- XML, RAW resources



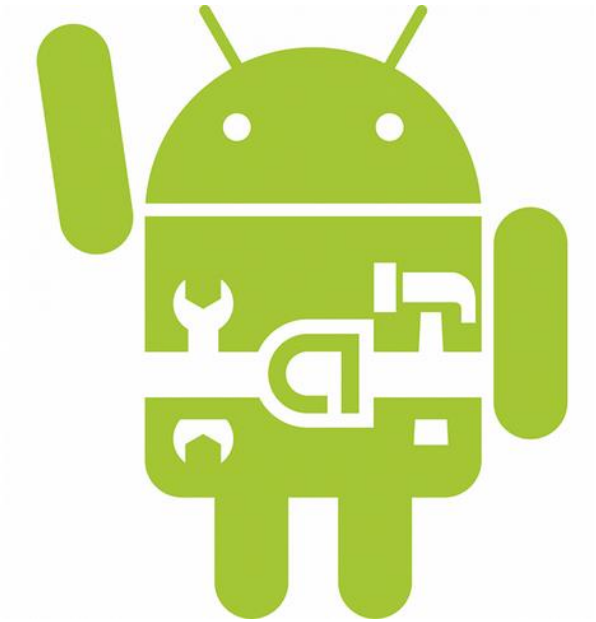
Android application – Activity

- User interface
- Interaction, gestures
- Lifecycle callback methods
- No main() method
- Views
  - Button
  - EditText
  - TextView
  - etc.



Android application development

- Java or Kotlin based native development
  - (+Crossplatform, NDK)
- Java classes with XML layouts
- Further resources: images, DB
- Manifest file
- Build process with Gradle
  - Java->Class->Dex
- APK output
- Play Store or private distribution



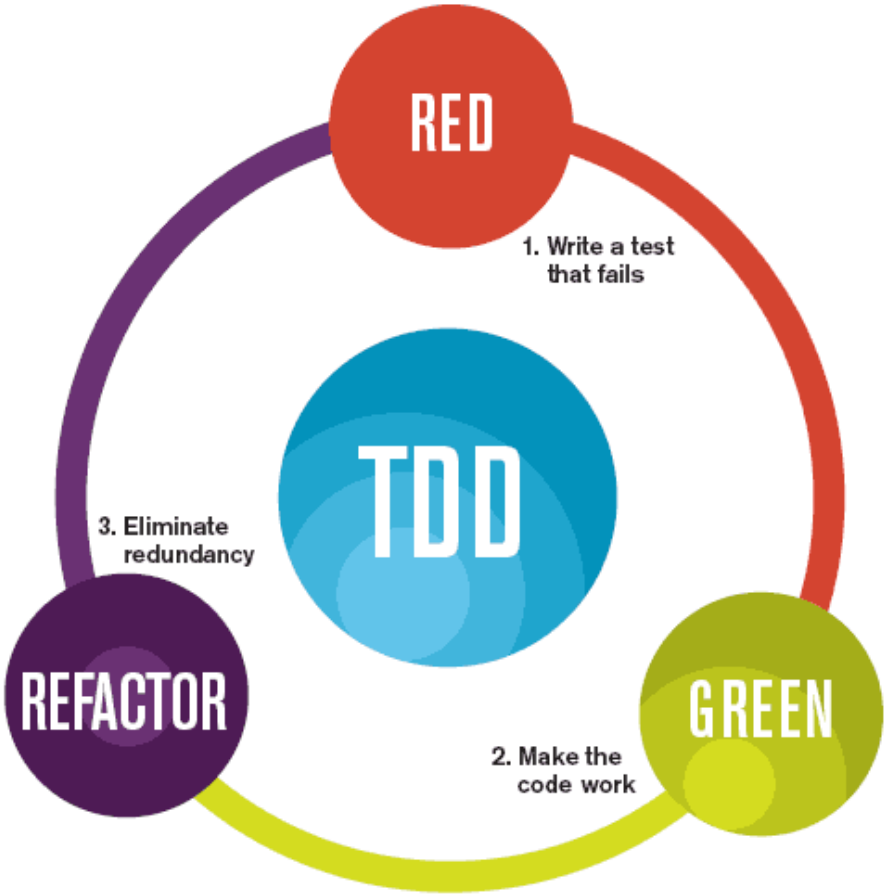
Android Studio

- 2013, IntellJ IDEA based
- Seperate business logic and UI development
- Built-in Emulator
- Test support features, plugins
- Build process tools
- Git support
- Publish, distribution

Approach & Architecture

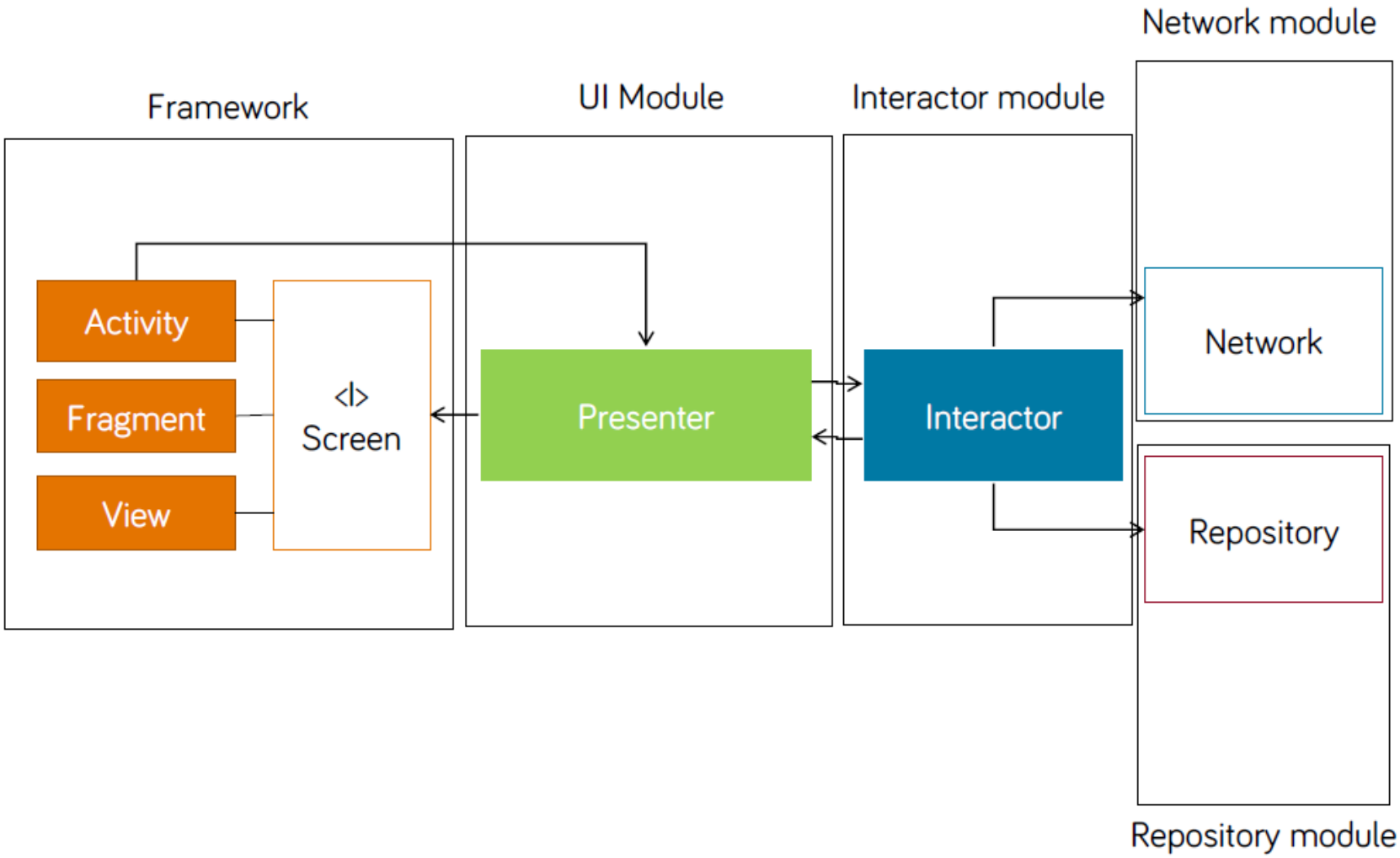
Test Driven Development

- 1. Write a test to check the code. Fail.
  - > No code yet.
- 2. Write the code to make the test Pass.
- 3. Refactor the code, reorganize it, etc. (performance)
  - > The tests remains Pass.

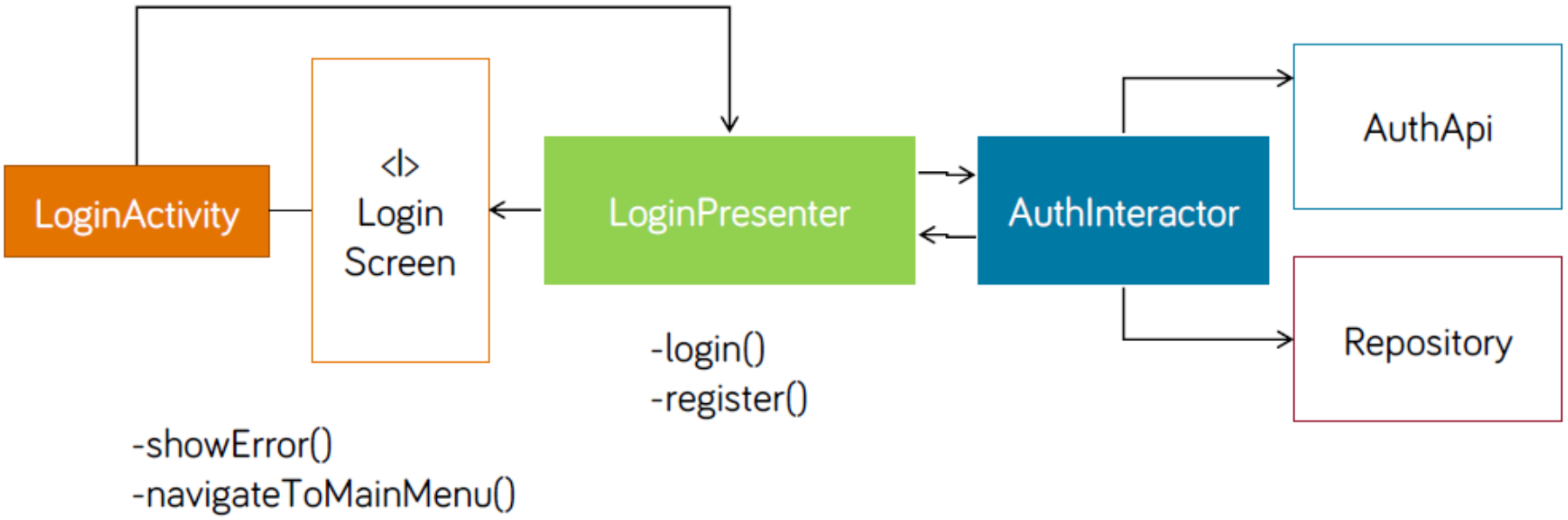


The mantra of Test-Driven Development (TDD) is "red, green, refactor."

Clean Architecture



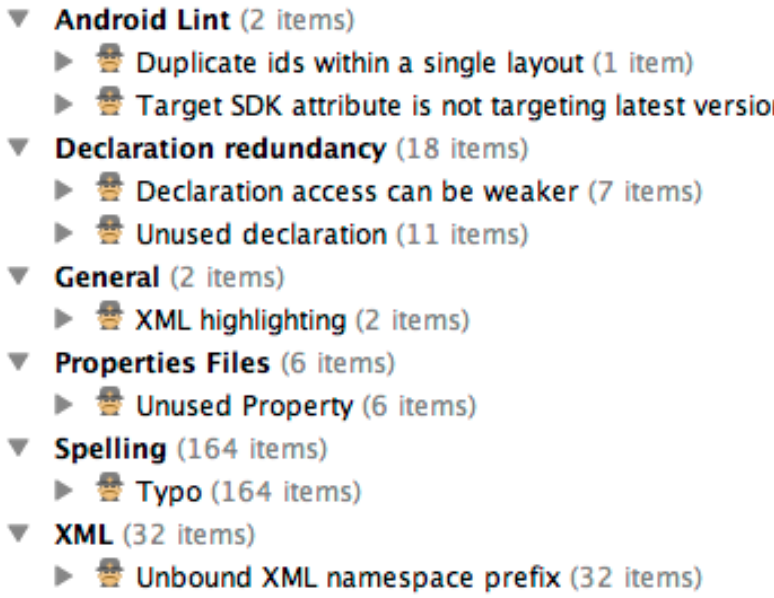
Clean Architecture - Example



Static code analysis on Android

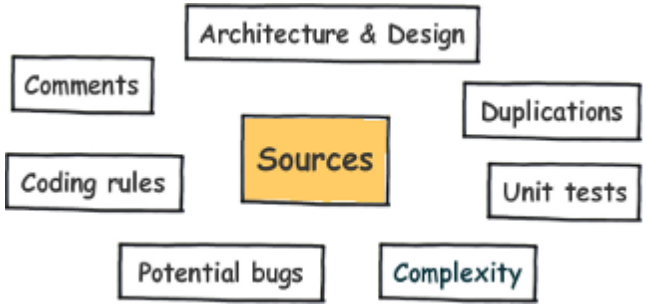
Lint

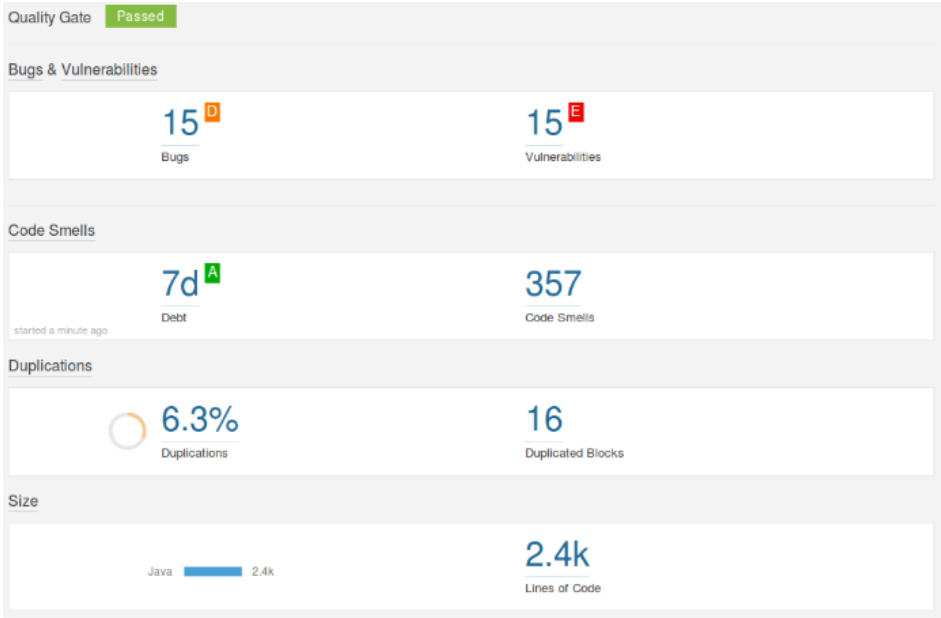
- Android
  - Spell checking
  - Examining Android XML Files
  - Attributes not available in older versions
  - Hardcoded texts
  - Warns for localization, eg. SimpleDateFormat
  - Layout inflating without parent
  - Padding margin symmetry
  - Unused resources
  - Visibilities, eg. where it can be stricter
- Coding style
- Control flow, Data flow
- Declaration redundancy
- Import
- Possible bugs
- XML syntax



Sonarqube

- Source code based analysis
- Community support
- Modes
  - > Analysis
  - > Preview
  - > Incremental
- Integration into CI
- Outputs
  - > Into DBs
  - > Online dashboards





Testing on Android

Testing on Android

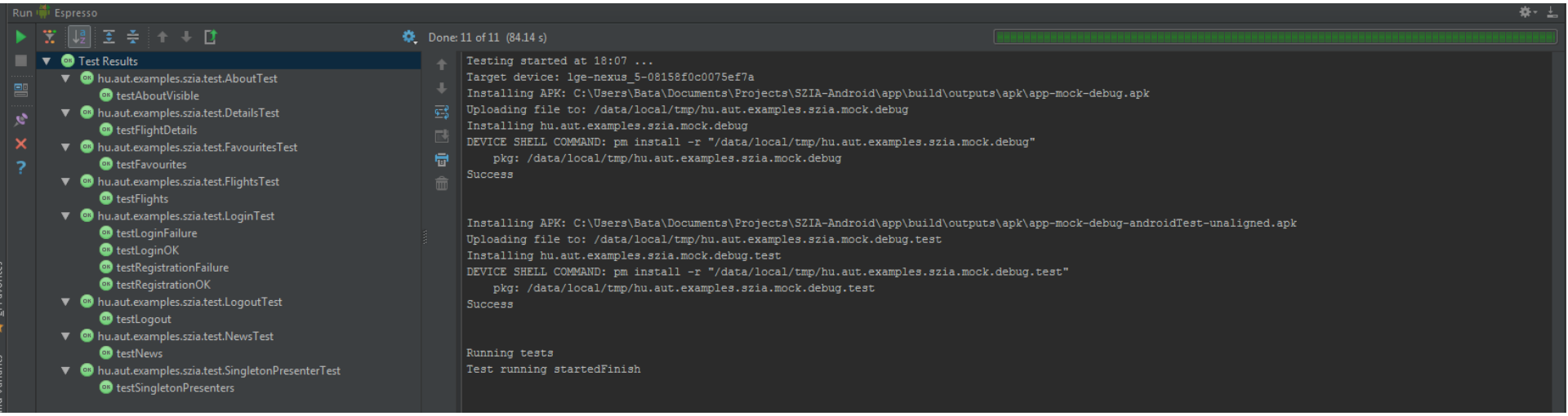
- On Android device
  - The developer's machine controls the test
  - Tests run on real/emulated Android device
  - \+ Test in the environment
  - \- Slow, complicated
- On developer machine
  - Tests run by the JVM
  - The Android specific element are simulated
  - \+ Fast, no device dependency
  - \- Not full range

UI & Screen testing

- **Espresso** offers a variety of static functions to access and validate elements
- Working with the image we can see only
  - Not seeing more than the user

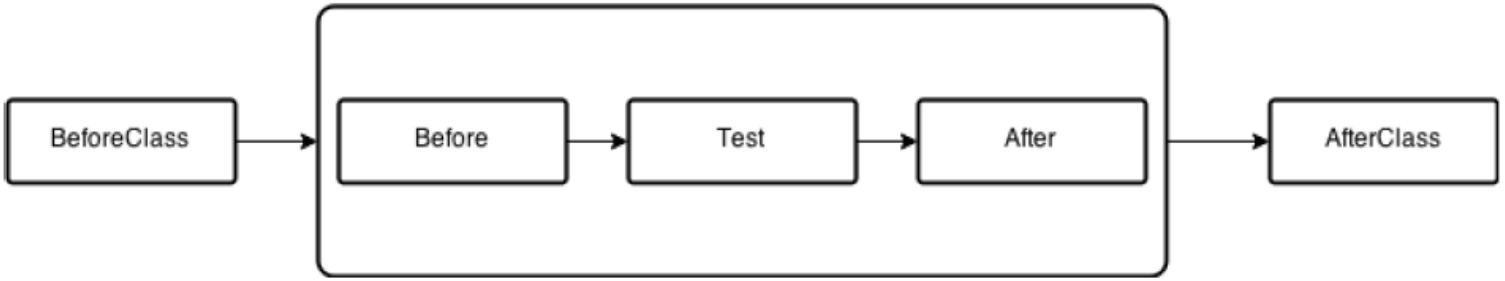
```
onView(withText(„Boarding")).perform(click());
```

```
onView(isAssignableFrom(Toolbar.class))
    .check(matches(withToolbarTitle(is(title))))
```



Unit testing - JUnit

- Open-source, support society
- Java annotations
  - Pojo classes
    - @Test, @Category, @Ignore etc.
  - Setup, Teardown
    - @Before(Class), @After(class)
- Test lifecycle



Unit testing - JUnit

- Assertions
  - True, False, Equals, Same, NotSame, Null, NotNull etc.

```
@Test
public void testAssertEquals() {
    org.junit.Assert.assertEquals(
        "text", "text");
}
```

- Hamcrest
  - AssertThat … both/and/or/is/not etc.

```
assertThat("albumen", both(containsString("a"))
    .and(containsString("b")));
```

- AAA
  - Arrange, Act, Assert

```
@Test
public void stringUpperCaseWithAAA(){
  //Arrange
  String message = "message";

  //Act
  String upperCaseMessage =
          message.toUpperCase();

  //Assert
  assertEquals("MESSAGE",
          upperCaseMessage);
}
```

Unit testing - Robolectric

- Robolectric Unit Test Framework for Android
- Advantages:
  - Compatible with Android SDK components
  - Activity life cycle can be manually controlled
  - The tests run directly on JVM, not device / emulator
  - Tests run relatively quickly, there is a possibility the TDD approach.
  - Can be run as a JUnit test from Android Studio (graphical report, code coverage tool ...)

Unit testing - Example

```
@RunWith(RobolectricGradleTestRunner.class)
@Config(constants = BuildConfig.class)
public class MainActivityTest {

    MainActivity mainActivity;

    @Before
    public void setup() throws Exception {
        // Calls the lifecycle: create-start-postCreate-resume
        mainActivity = Robolectric.setupActivity(MainActivity.class);
    }

    @Test
    public void testSubmitButtonIsDisplayed() throws Exception {
        Assert.assertEquals(
                mainActivity.findViewById(R.id.btnSubmit).getVisibility(),
                View.VISIBLE);
    }
}
```

Mocking – Mockito

- Interface / Class substitution
  - > Allows run time creation of mock objects
- Behavior setting
  - > Clearing the response of specific actions
  - > Examination of calls

```
@Test
public void mockitoMockTest(){
   MathService mathService =
                  mock(MathService.class);
   when(mathService.sum(2, 3)).thenReturn(15);


   int mockedResult = mathService.sum(2, 3);

   assertEquals(mockedResult, 15);
}
```

```
@InjectMocks  MathService mathService;
@Spy Square square;

@Test
public void mockitoInvokeTest(){

   MockitoAnnotations.initMocks(this);

   when(square. square(2)).thenReturn(5);

   //using mock object
   int result = mathService.square(2);

   assertEquals(5, result);
}
```

Mocking – Mockito calls

```
@Test
public void mockitoTest(){
    MathService mathService =
                mock(MathService.class);

    when(mathService.sum(2,2)).thenCallRealMethod();
    when(mathService.sum(2, 3)).thenReturn(15);

    int actual = mathService.sum(2, 2);
    int actual = mathService.sum(2, 3);

    assertEquals(actual, 4);
    assertEquals(wrong, 15);
}
```

```
mockedList.add("once");
mockedList.add("twice");
mockedList.add("twice");
mockedList.add("three times");
mockedList.add("three times");
mockedList.add("three times");

//following two verifications work exactly the same - times(1) is used by
default
verify(mockedList).add("once");
verify(mockedList, times(1)).add("once");

//exact number of invocations verification
verify(mockedList, times(2)).add("twice");
verify(mockedList, times(3)).add("three times");

//verification using never(). never() is an alias to times(0)
verify(mockedList, never()).add("never happened");

//verification using atLeast()/atMost()
verify(mockedList, atLeastOnce()).add("once");
verify(mockedList, atLeast(6)).add(anyString());
verify(mockedList, atMost(5)).add("three times");
```

Jacoco

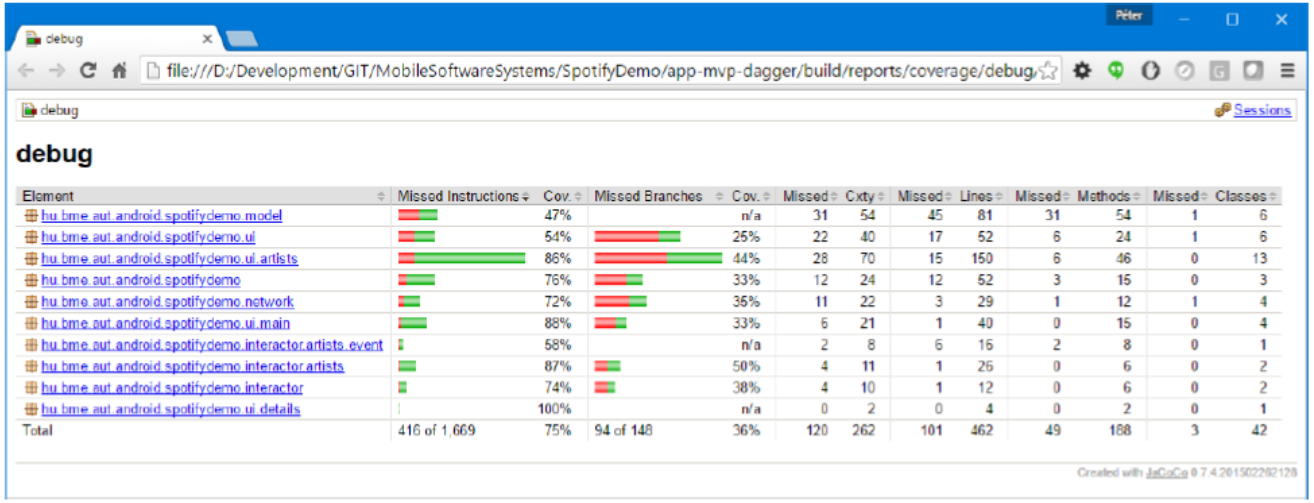
- Code coverage by test tool
- Debug build type:

```
Debug {
    testCoverageEnabled true
}
```

- Reporting

gradlebuild->other->CreateDebugAndroidTestCoverageReport

- HTML output

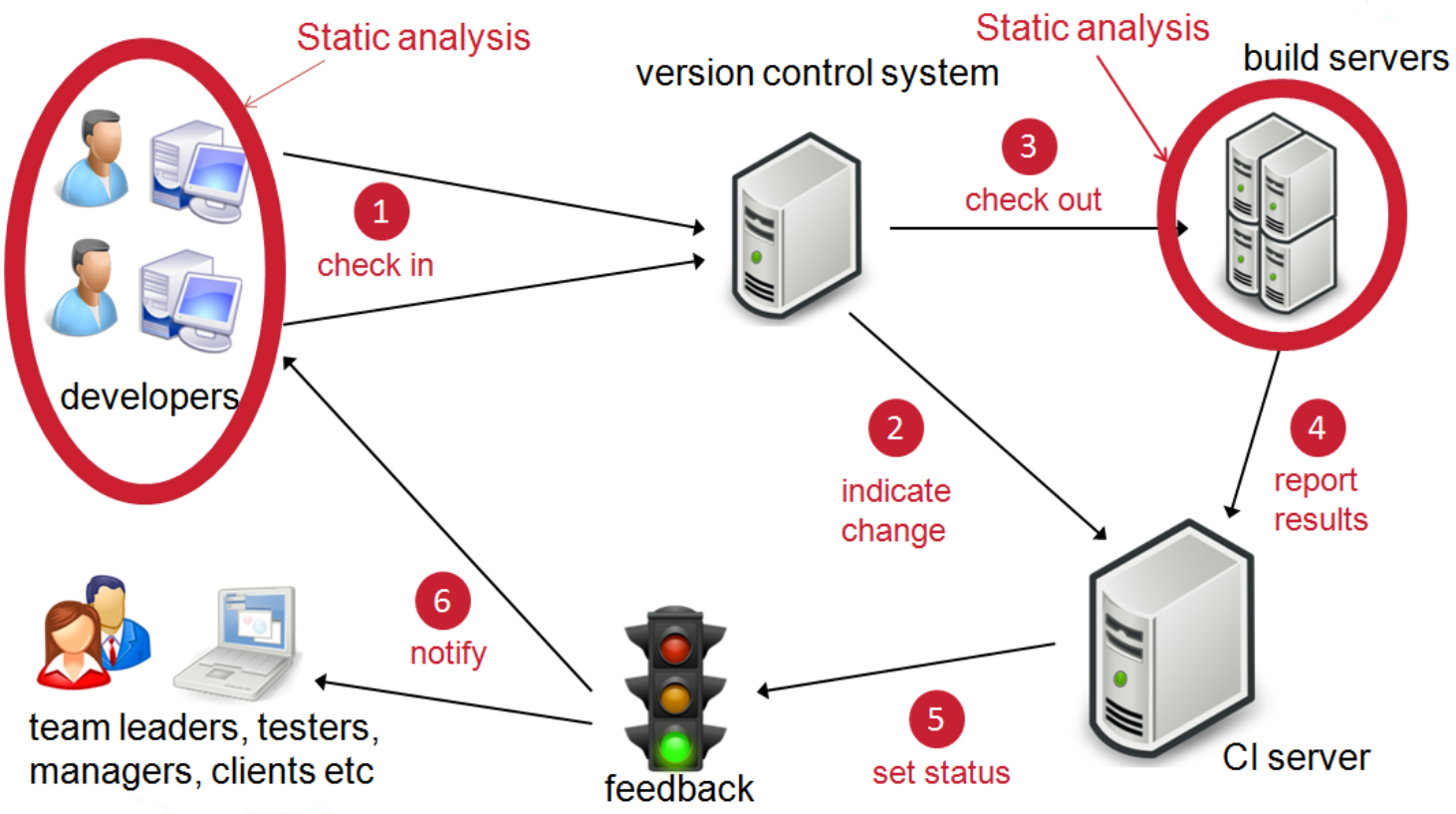


Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
hu.bme.aut.android.spotifydemo.model		47%		n/a	31	54	45	81	31	54	1	6
hu.bme.aut.android.spotifydemo.ui		54%		25%	22	40	17	52	6	24	1	6
hu.bme.aut.android.spotifydemo.ui.artists		86%		44%	28	70	15	150	6	46	0	13
hu.bme.aut.android.spotifydemo		76%		33%	12	24	12	52	3	15	0	3
hu.bme.aut.android.spotifydemo.network		72%		35%	11	22	3	29	1	12	1	4
hu.bme.aut.android.spotifydemo.ui.main		88%		33%	6	21	1	40	0	15	0	4
hu.bme.aut.android.spotifydemo.interactor.artists.event		58%		n/a	2	8	6	16	2	8	0	1
hu.bme.aut.android.spotifydemo.interactor.artists		87%		50%	4	11	1	26	0	6	0	2
hu.bme.aut.android.spotifydemo.interactor		74%		38%	4	10	1	12	0	6	0	2
hu.bme.aut.android.spotifydemo.ui.details		100%		n/a	0	2	0	4	0	2	0	1
Total	416 of 1,669	75%	94 of 148	36%	120	262	101	462	49	188	3	42

Continous Integration and Delivery

Continous Integration

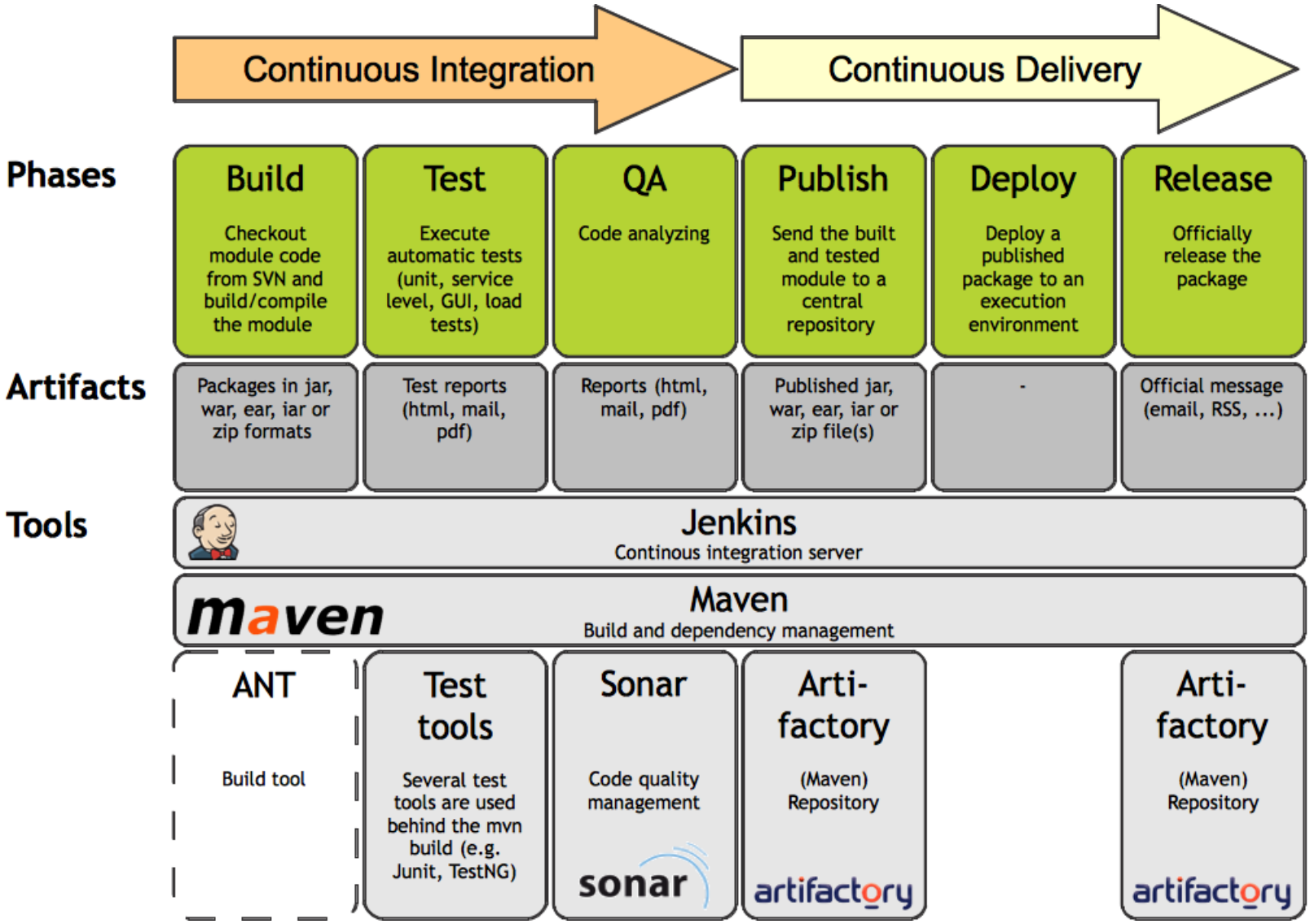
- Jenkins, Teamcity (local setup)
- Travis CI, Circle CI, Codeship (cloud based)



Continous Integration

- Own server + Software or SaaS solution
- Features
  - Integration with version control (Git)
  - Generate test reports
  - Push into artifact repositories
  - Publish into real and test environments
  - Notify the developers
  - Ticketing system, monitoring
  - Integration with tools
    - Build systems, code analysis etc.

Continous Integration and Delivery



Thank you for your attention!

Based on Mobil Software System Development course slides by
Tamás Balogh, Péter Ekler, Péter Gerencsér, Tibor Kántor and Dániel Tóth