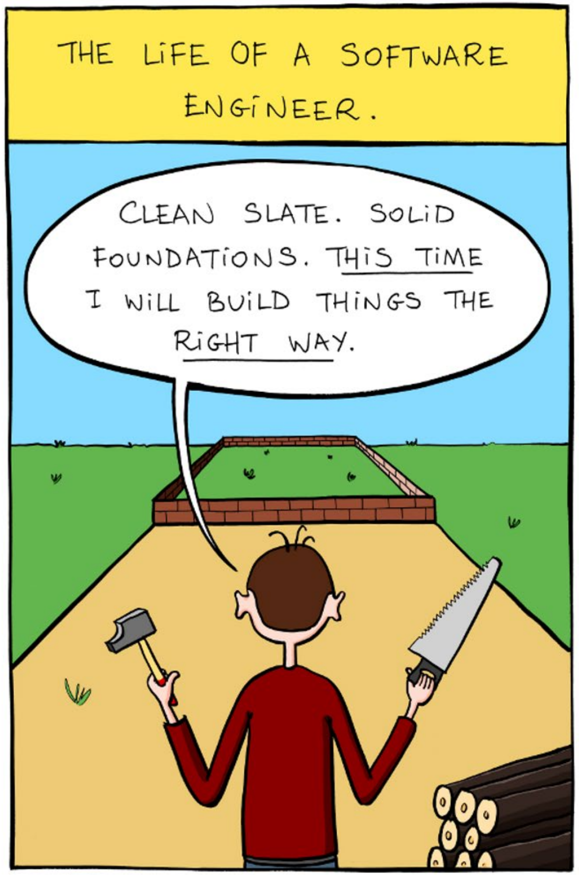
Test the untested - our journey from zero coverage to automated testing

Anton Augsburg
April 2019

Architecture

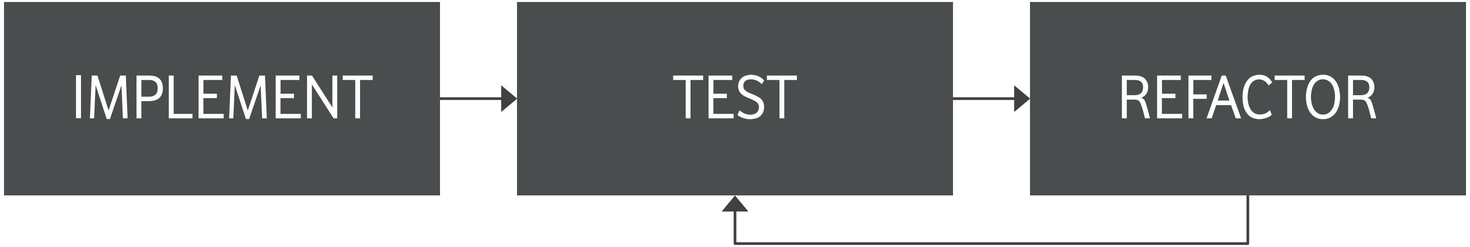

THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.

http://bonkersworld.net/building-software

Challenges

- Long lifetime of SDK
- Hard to update in case of errors
- High risk of regressions if code is edited

Perfect life



Our test approach

Before

MANUAL + CROWD

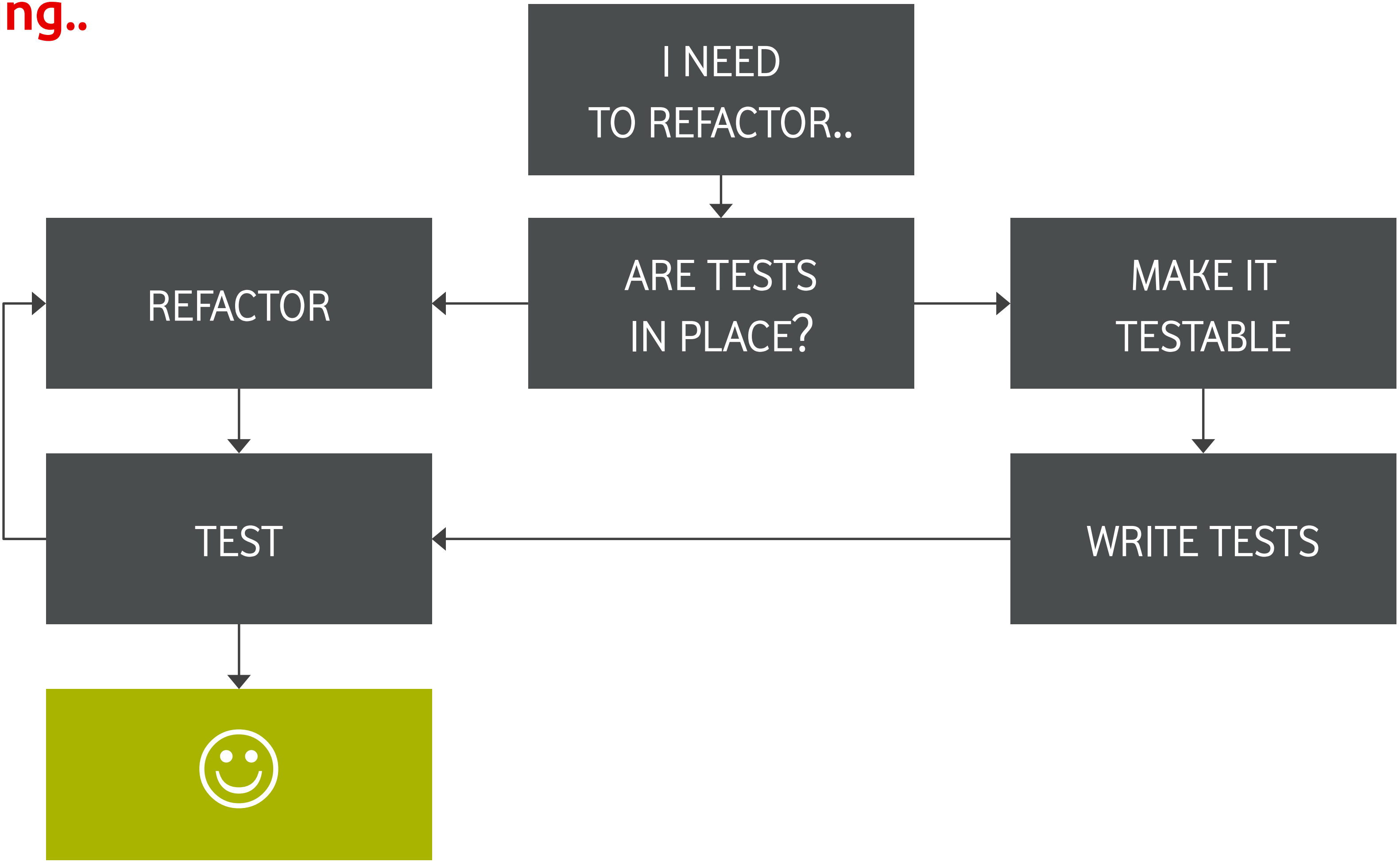
Our test approach

After

MANUAL + CROWD + AUTOMATED

Moving to testable code

Starting..



I NEED TO REFACTOR..

ARE TESTS IN PLACE?

REFACTOR

MAKE IT TESTABLE

TEST

WRITE TESTS

☺

Obstacles

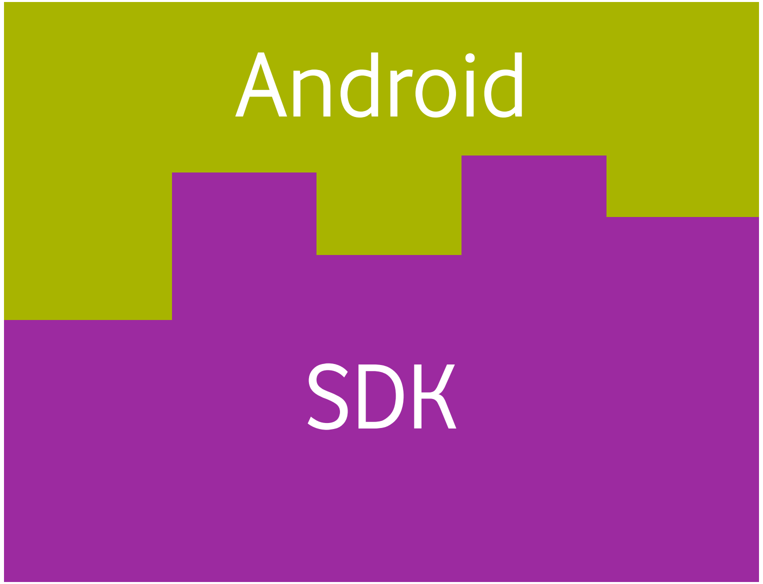
DEPENDENCIES

STATIC CALLS

CODE STRUCTURE

Resolving dependencies

Revising our architecture



```
public class MemoryLogger {

  private Database database;

  public MemoryLogger() {
    database = SDK.getDatabase();
  }

  public void logCurrentMemoryConsumption() {
     ActivityManager activityManager = (ActivityManager)
         SDK.getContext().getSystemService(Context.ACTIVITY_SERVICE);

     int pid = Process.myPid();

     MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

     database.updateMemoryConsumption(memInfos[0]);
  }
}
```

```
public class MemoryLogger {

  private Database database;

  public MemoryLogger() {
    database = SDK.getDatabase();
  }

  public void logCurrentMemoryConsumption() {
     ActivityManager activityManager = (ActivityManager)
         SDK.getContext().getSystemService(Context.ACTIVITY_SERVICE);

     int pid = Process.myPid();

     MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

     database.updateMemoryConsumption(memInfos[0]);
  }
}
```

```
public class MemoryLogger {

  private Database database;

  public MemoryLogger() {
    database = SDK.getDatabase();
  }

  public void logCurrentMemoryConsumption() {
    ActivityManager activityManager = (ActivityManager)
        SDK.getContext().getSystemService(Context.ACTIVITY_SERVICE);

    int pid = Process.myPid();

    MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

    database.updateMemoryConsumption(memInfos[0]);
  }
}
```

```java
public class MemoryLogger {

  private Database database;

  public MemoryLogger() {
    database = SDK.getDatabase();
  }

  public void logCurrentMemoryConsumption() {
     ActivityManager activityManager = (ActivityManager)
         SDK.getContext().getSystemService(Context.ACTIVITY_SERVICE);

     int pid = Process.myPid();

     MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

     database.updateMemoryConsumption(memInfos[0]);
  }
}
```

```
public class MemoryLogger {

  private Database database;

  public MemoryLogger() {
    database = SDK.getDatabase();
  }

  public void logCurrentMemoryConsumption() {
     IActivityManager activityManager = AndroidRuntime.getActivityManager();

     int pid = Process.myPid();

     MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

     database.updateMemoryConsumption(memInfos[0]);
  }
}
```

```
public class MemoryLogger {

  private Database database;

  public MemoryLogger() {
    database = SDK.getDatabase();
  }

  public void logCurrentMemoryConsumption() {
    IActivityManager activityManager = AndroidRuntime.getActivityManager();

    int pid = Process.myPid();

    MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

    database.updateMemoryConsumption(memInfos[0]);
  }
}
```

```
public class MemoryLogger {

  private Database database;

  public MemoryLogger(Database database) {
    this.database = database;
  }

  public void logCurrentMemoryConsumption() {
     IActivityManager activityManager = AndroidRuntime.getActivityManager();

     int pid = Process.myPid();

     MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

     database.updateMemoryConsumption(memInfos[0]);
  }
}
```

```
public class MemoryLogger {

  private Database database;

  public MemoryLogger(~~Database database~~) {
    ~~this.database = database;~~
  }

  public void logCurrentMemoryConsumption() {
     IActivityManager activityManager = AndroidRuntime.getActivityManager();

     int pid = Process.myPid();

     MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

     database.updateMemoryConsumption(memInfos[0]);
  }
}
```

```java
public class MemoryLogger {

  public MemoryLogger() {
  }

  public void logCurrentMemoryConsumption() {
     IActivityManager activityManager = AndroidRuntime.getActivityManager();

     int pid = Process.myPid();

     MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

     getDatabase().updateMemoryConsumption(memInfos[0]);
  }

  Database getDatabase() {
     return SDK.getDatabase();
  }
}
```

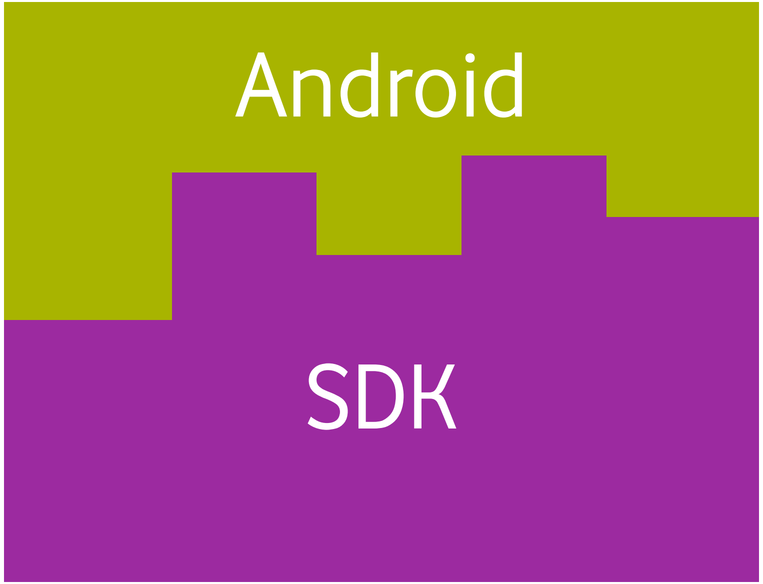
Lazy-init in getter
→ override in special test class

```
public class TestMemoryLogger {

  @Override
  Database getDatabase() {
     return new FakeDatabase();
  }

}
```

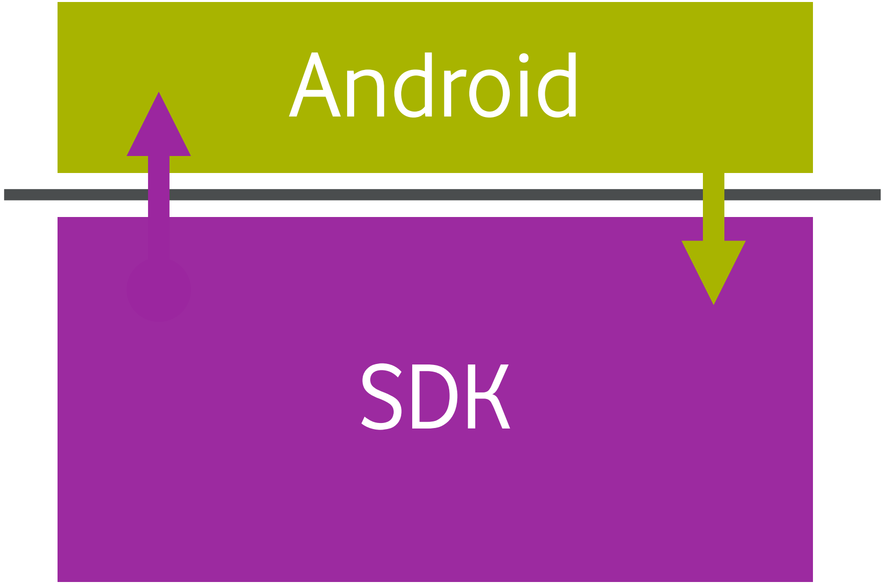
```
public class MemoryLogger {

  private Database database;

  public MemoryLogger() {
  }

  public void logCurrentMemoryConsumption() {
     IActivityManager activityManager = AndroidRuntime.getActivityManager();

     int pid = Process.myPid();

     MemoryInfo[] memInfos = activityManager.getProcessMemoryInfo(new int[] {pid});

     dataBase.updateMemoryConsumption(memInfos[0]);
  }

  static void setDatabase(Database database) {
     this.database = database;
  }
}
```

Static setter
→ call and replace in test class

Revising our architecture



Revising our architecture



jUnit can help us

```
public class MemoryLoggerTest {

  @ClassRule
  public static EnvironmentRule environmentRule = new EnvironmentRule();

  @Test
  public void myFirstTest() {
      ...
  }

}
```

jUnit can help us

```
public class MemoryLoggerTest {

   @ClassRule
   public static EnvironmentRule environmentRule = new EnvironmentRule();

   @Test
   public void myFirstTest() {
       ..
   }

}
```

```
public interface TestRule {

   public Statement apply(Statement base, Description description)

}
```

```java
public class EnvironmentRule implements TestRule {

  public Statement apply(Statement base, Description description) {
      return new Statement() {
          @Override
          public void evaluate() throws Throwable {
              before();
              try {
                  base.evaluate();
              } finally {
                  after();
              }
          }
      };
  }




}
```

```java
public class EnvironmentRule implements TestRule {

  public Statement apply(Statement base, Description description) {
      return new Statement() {
          @Override
          public void evaluate() throws Throwable {
              before();
              try {
                  base.evaluate();
              } finally {
                  after();
              }
          }
      };
  }



}
```

```
public class EnvironmentRule implements TestRule {

  public Statement apply(Statement base, Description description) {
      return new Statement() {
          @Override
          public void evaluate() throws Throwable {
              before();
              try {
                  base.evaluate();
              } finally {
                  after();
              }
          }
      };
  }

  protected void before() throws Throwable {
      // setup the environment here
  }

  protected void after() {
      // tear down so we do not leak state into other tests
  }
}
```

Revising our architecture

- Use constructor injection
- Introduce abstractions
- Always depend on abstraction and not on implementation

Revise static calls

```
public class MemoryLogger {

  public void logCurrentMemoryConsumption() {
      ...
     FileUtils.writeToFile(memInfos[0].toString(), "memory.log");
  }
}

public class FileUtils {

  public static void writeToFile(String filename, String fileContents) {
    FileOutputStream outputStream;
    try {
        outputStream = someContext().openFileOutput(filename, Context.MODE_PRIVATE);
        outputStream.write(fileContents.getBytes());
        outputStream.close();
    } catch (Exception e) {
        ...
    }
  }
}
```

```java
public class FileUtils {

  public static void writeToFile(String filename, String fileContents) {
    FileOutputStream outputStream;
    try {
        outputStream = someContext().openFileOutput(filename, Context.MODE_PRIVATE);
        outputStream.write(fileContents.getBytes());
        outputStream.close();
    } catch (Exception e) {
        ...
    }
  }
}
```

```
public class FileUtils {

  public static void writeToFile(String filename, String fileContents) {
    FileOutputStream outputStream;
    try {
        outputStream = someContext().openFileOutput(filename, Context.MODE_PRIVATE);
        outputStream.write(fileContents.getBytes());
        outputStream.close();
    } catch (Exception e) {
        ...
    }
  }
}

class FileUtilsImpl {

  public void writeToFile(String filename, String fileContents) {

  }
}
```

```
public class FileUtils {

  public static void writeToFile(String filename, String fileContents) {

  }
}

class FileUtilsImpl {

  public void writeToFile(String filename, String fileContents) {
    FileOutputStream outputStream;
    try {
        outputStream = someContext().openFileOutput(filename, Context.MODE_PRIVATE);
        outputStream.write(fileContents.getBytes());
        outputStream.close();
    } catch (Exception e) {
        ...
    }
  }
}
```

```
public class FileUtils {

  static FileUtilsImpl IMPL = new FileUtilsImpl()

  public static void writeToFile(String filename, String fileContents) {
    IMPL.writeToFile(filename, fileContents);
  }
}

public class FileUtilsImpl {

  public void writeToFile(String filename, String fileContents) {
      ...
  }
}
```

Replace in test with InMemoryFileUtilsImpl() e.g. via static setter of field access

Restructuring code

```
public class Activity {

    public void setRequestedOrientation(@ActivityInfo.ScreenOrientation int requestedOrientation) {
        if (mParent == null) {
            try {
                ActivityManagerNative.getDefault().setRequestedOrientation(mToken, requestedOrientation);
            } catch (RemoteException e) {
                // Empty
            }
        } else {
            mParent.setRequestedOrientation(requestedOrientation);
        }
    }

    @ActivityInfo.ScreenOrientation
    public int getRequestedOrientation() {
        if (mParent == null) {
            try {
                return ActivityManagerNative.getDefault().getRequestedOrientation(mToken);
            } catch (RemoteException e) {
                // Empty
            }
        } else {
            return mParent.getRequestedOrientation();
        }
        return ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED;
    }

    public int getTaskId() {
        try {
            return ActivityManagerNative.getDefault().getTaskForActivity(mToken, false);
        } catch (RemoteException e) {
            return -1;
        }
    }

    public boolean isTaskRoot() {
        try {
            return ActivityManagerNative.getDefault().getTaskForActivity(mToken, true) >= 0;
        } catch (RemoteException e) {
            return false;
        }
    }

    public boolean moveTaskToBack(boolean nonRoot) {
        try {
            return ActivityManagerNative.getDefault().moveActivityTaskToBack(mToken, nonRoot);
        } catch (RemoteException e) {
            // Empty
        }
        return false;
    }

    @NonNull
    public String getLocalClassName() {
        final String pkg = getPackageName();
        final String cls = mComponent.getClassName();
        int packageLen = pkg.length();
        if (!cls.startsWith(pkg) || cls.length() <= packageLen || cls.charAt(packageLen) != '.') {
            return cls;
        }
        return cls.substring(packageLen + 1);
    }
    public ComponentName getComponentName() {
        return mComponent;
    }

    public SharedPreferences getPreferences(int mode) {
        return getSharedPreferences(getLocalClassName(), mode);
    }

    private void ensureSearchManager() {
        if (mSearchManager != null) {
            return;
        }

        mSearchManager = new SearchManager(this, null);
    }

    @Override
    public Object getSystemService(@ServiceName @NonNull String name) {
        if (getBaseContext() == null) {
            throw new IllegalStateException("System services not available to Activities before onCreate()");
        }

        if (WINDOW_SERVICE.equals(name)) {
            return mWindowManager;
        } else if (SEARCH_SERVICE.equals(name)) {
            ensureSearchManager();
            return mSearchManager;
        }
        return super.getSystemService(name);
    }

    public void setTitle(CharSequence title) {
        mTitle = title;
        onTitleChanged(title, mTitleColor);

        if (mParent != null) {
            mParent.onChildTitleChanged(this, title);
        }
    }

    public void setTitle(int titleId) {
        setTitle(getText(titleId));
    }

    @Deprecated
    public void setTitleColor(int textColor) {
        mTitleColor = textColor;
        onTitleChanged(mTitle, textColor);
    }

    public final CharSequence getTitle() {
        return mTitle;
    }

    public final int getTitleColor() {
        return mTitleColor;
    }

    protected void onTitleChanged(CharSequence title, int color) {
        if (mTitleReady) {
            final Window win = getWindow();
            if (win != null) {
                win.setTitle(title);
                if (color != 0) {
                    win.setTitleColor(color);
```

```
protected void onTitleChanged(CharSequence title, int color) {
    if (mTitleReady) {
        final Window win = getWindow();
        if (win != null) {
            win.setTitle(title);
            if (color != 0) {
                win.setTitleColor(color);
            }
        }
        if (mActionBar != null) {
            mActionBar.setWindowTitle(title);
        }
    }
}

public boolean dispatchPopulateAccessibilityEvent(AccessibilityEvent event) {
    event.setClassName(getClass().getName());
    event.setPackageName(getPackageName());

    LayoutParams params = getWindow().getAttributes();
    boolean isFullScreen = (params.width == LayoutParams.MATCH_PARENT) &&
        (params.height == LayoutParams.MATCH_PARENT);
    event.setFullScreen(isFullScreen);

    CharSequence title = getTitle();
    if (!TextUtils.isEmpty(title)) {
       event.getText().add(title);
    }

    return true;
}

@Override
protected void onApplyThemeResource(Resources.Theme theme, int resid,
        boolean first) {
    if (mParent == null) {
        super.onApplyThemeResource(theme, resid, first);
    } else {
        try {
            theme.setTo(mParent.getTheme());
        } catch (Exception e) {
            // Empty
        }
        theme.applyStyle(resid, false);
    }

    // Get the primary color and update the TaskDescription for this activity
    if (theme != null) {
        TypedArray a = theme.obtainStyledAttributes(com.android.internal.R.styleable.Theme);
        int colorPrimary = a.getColor(com.android.internal.R.styleable.Theme_colorPrimary, 0);
        a.recycle();
        if (colorPrimary != 0) {
            ActivityManager.TaskDescription v = new ActivityManager.TaskDescription(null, null,
                    colorPrimary);
            setTaskDescription(v);
        }
    }
}
```

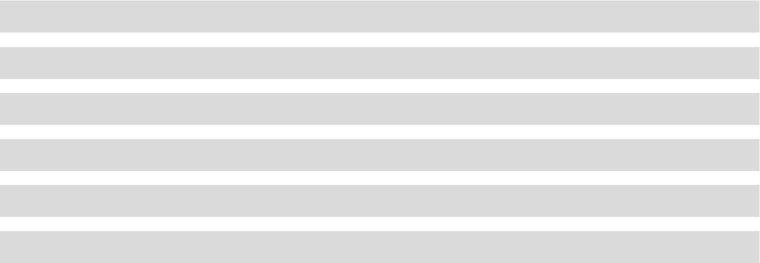
```
protected void onTitleChanged(CharSequence title, int color) {
    if (mTitleReady) {
        final Window win = getWindow();
        if (win != null) {
            win.setTitle(title);
            if (color != 0) {
                win.setTitleColor(color);
            }
        }
        if (mActionBar != null) {
            mActionBar.setWindowTitle(title);
        }
    }
}

public boolean dispatchPopulateAccessibilityEvent(AccessibilityEvent event) {
    event.setClassName(getClass().getName());
    event.setPackageName(getPackageName());

    LayoutParams params = getWindow().getAttributes();
    boolean isFullScreen = (params.width == LayoutParams.MATCH_PARENT) &&
        (params.height == LayoutParams.MATCH_PARENT);
    event.setFullScreen(isFullScreen);

    CharSequence title = getTitle();
    if (!TextUtils.isEmpty(title)) {
       event.getText().add(title);
    }

    return true;
}

@Override
protected void onApplyThemeResource(Resources.Theme theme, int resid,
        boolean first) {
    if (mParent == null) {
        super.onApplyThemeResource(theme, resid, first);
    } else {
        try {
            theme.setTo(mParent.getTheme());
        } catch (Exception e) {
            // Empty
        }
        theme.applyStyle(resid, false);
    }

    // Get the primary color and update the TaskDescription for this activity
    if (theme != null) {
        TypedArray a = theme.obtainStyledAttributes(com.android.internal.R.styleable.Theme);
        int colorPrimary = a.getColor(com.android.internal.R.styleable.Theme_colorPrimary, 0);
        a.recycle();
        if (colorPrimary != 0) {
            ActivityManager.TaskDescription v = new ActivityManager.TaskDescription(null, null,
                    colorPrimary);
            setTaskDescription(v);
        }
    }
}
```

```
public boolean dispatchPopulateAccessibilityEvent(AccessibilityEvent event) {
    event.setClassName(getClass().getName());
    event.setPackageName(getPackageName());

    LayoutParams params = getWindow().getAttributes();
    boolean isFullScreen = (params.width == LayoutParams.MATCH_PARENT) &&
        (params.height == LayoutParams.MATCH_PARENT);
    event.setFullScreen(isFullScreen);

    CharSequence title = getTitle();
    if (!TextUtils.isEmpty(title)) {
       event.getText().add(title);
    }

    return true;
}

@Override
protected void onApplyThemeResource(Resources.Theme theme, int resid,
        boolean first) {
    if (mParent == null) {
        super.onApplyThemeResource(theme, resid, first);
    } else {
        try {
            theme.setTo(mParent.getTheme());
        } catch (Exception e) {
            // Empty
        }
        theme.applyStyle(resid, false);
    }

    // Get the primary color and update the TaskDescription for this activity
    if (theme != null) {
        TypedArray a = theme.obtainStyledAttributes(com.android.internal.R.styleable.Theme);
        int colorPrimary = a.getColor(com.android.internal.R.styleable.Theme_colorPrimary, 0);
        a.recycle();
        if (colorPrimary != 0) {
            ActivityManager.TaskDescription v = new ActivityManager.TaskDescription(null, null,
                    colorPrimary);
            setTaskDescription(v);
        }
    }
}
```

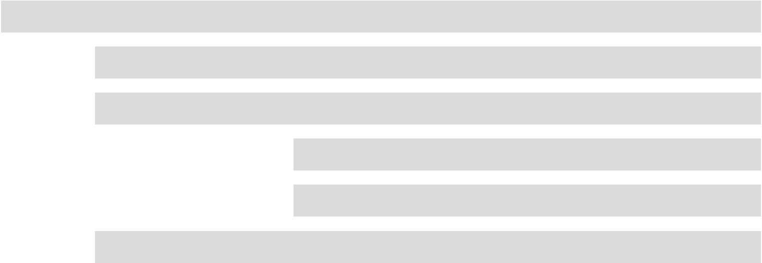
Bullet method

```
@Override
protected void onApplyThemeResource(Resources.Theme theme, int resid,
        boolean first) {
    if (mParent == null) {
        super.onApplyThemeResource(theme, resid, first);
    } else {
        try {
            theme.setTo(mParent.getTheme());
        } catch (Exception e) {
            // Empty
        }
        theme.applyStyle(resid, false);
    }

    // Get the primary color and update the TaskDescription for this activity
    if (theme != null) {
        TypedArray a = theme.obtainStyledAttributes(com.android.internal.R.styleable.Theme);
        int colorPrimary = a.getColor(com.android.internal.R.styleable.Theme_colorPrimary, 0);
        a.recycle();
        if (colorPrimary != 0) {
            ActivityManager.TaskDescription v = new ActivityManager.TaskDescription(null, null,
                    colorPrimary);
            setTaskDescription(v);
        }
    }
}
```

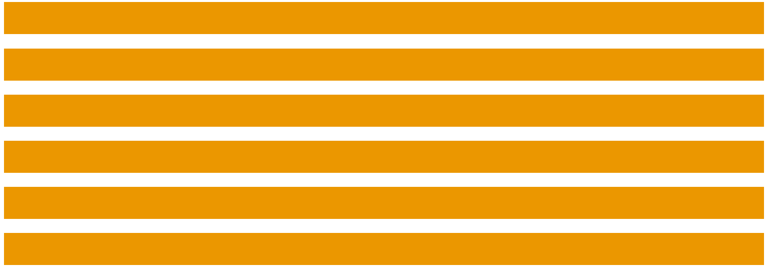
Bullet method



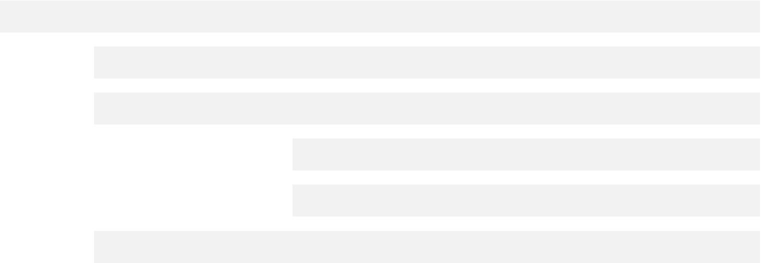
Nested method



Bullet method



Nested method



```
public String buildHtmlPage(int id) {
  PageContent content = database.getContent(id);
  StringBuilder sb = new StringBuilder();
  sb.append("<!DOCTYPE>");
  sb.append("<html>");
  sb.append("<head>")
   .append("<meta content=\"text/html; chartset=UTF-8\" />")
   .append("<title>").append(content.getTitle()). append("</title>")
   .append("</head>");
  sb.append("<body>")
   .append("<h1>")append(content.getHeading()).append("</h1>")
   .append("<section>").append(content.getText()).append("</section>")
  sb.append("<footer>").append(content.getCopyright()).append("</footer>");
  sb.append("</body>");
  sb.append("</html>");
  return sb.toString();
}
```

```
public String buildHtmlPage(int id) {
    PageContent content = database.getContent(id);
    StringBuilder sb = new StringBuilder();
    sb.append("<!DOCTYPE>");
    sb.append("<html>");
    sb.append("<head>")
     .append("<meta content=\"text/html; chartset=UTF-8\" />")
     .append("<title>").append(content.getTitle()). append("</title>")
     .append("</head>");
    sb.append("<body>")
     .append("<h1>")append(content.getHeading()).append("</h1>")
     .append("<section>").append(content.getText()).append("</section>")
    sb.append("<footer>").append(content.getCopyright()).append("</footer>");
    sb.append("</body>");
    sb.append("</html>");
    return sb.toString();
}
```

```
public String buildHtmlPage(PageContent content) {

   StringBuilder sb = new StringBuilder();
   sb.append("<!DOCTYPE>");
   sb.append("<html>");
   sb.append("<head>")
    .append("<meta content=\"text/html; chartset=UTF-8\" />")
    .append("<title>").append(content.getTitle()). append("</title>")
    .append("</head>");
   sb.append("<body>")
    .append("<h1>")append(content.getHeading()).append("</h1>")
    .append("<section>").append(content.getText()).append("</section>")
   sb.append("<footer>").append(content.getCopyright()).append("</footer>");
   sb.append("</body>");
   sb.append("</html>");
   return sb.toString();
}
```

```
public String buildHtmlPage(PageContent content) {

    StringBuilder sb = new StringBuilder();
    sb.append("<!DOCTYPE>");
    sb.append("<html>");
    sb.append("<head>")
     .append("<meta content=\"text/html; chartset=UTF-8\" />")
     .append("<title>").append(content.getTitle()). append("</title")
     .append("</head>");
    sb.append("<body>")
     .append("<h1>")append(content.getHeading()).append("</h1>")
     .append("<section>").append(content.getText()).append("</section>")
    sb.append("<footer>").append(content.getCopyright()).append("</footer>");
    sb.append("</body>");
    sb.append("</html>");
    return sb.toString();
}
```

```
public String buildHtmlPage(PageContent content) {

   StringBuilder sb = new StringBuilder();
   appendDoctype(sb);
   sb.append("<html>");
   sb.append("<head>")
    .append("<meta content=\"text/html; chartset=UTF-8\" />")
    .append("<title>").append(content.getTitle()). append("</title")
    .append("</head>");
   sb.append("<body>")
    .append("<h1>")append(content.getHeading()).append("</h1>")
    .append("<section>").append(content.getText()).append("</section>")
   sb.append("<footer>").append(content.getCopyright()).append("</footer>");
   sb.append("</body>");
   sb.append("</html>");
   return sb.toString();
}
```

```
public String buildHtmlPage(PageContent content) {

   StringBuilder sb = new StringBuilder();
   appendDoctype(sb);
   startHtml(sb);
   sb.append("<head>")
    .append("<meta content=\"text/html; chartset=UTF-8\" />")
    .append("<title>").append(content.getTitle()). append("</title")
    .append("</head>");
   sb.append("<body>")
    .append("<h1>")append(content.getHeading()).append("</h1>")
    .append("<section>").append(content.getText()).append("</section>")
   sb.append("<footer>").append(content.getCopyright()).append("</footer>");
   sb.append("</body>");
   sb.append("</html>");
   return sb.toString();
}
```

```
public String buildHtmlPage(PageContent content) {

  StringBuilder sb = new StringBuilder();
  appendDoctype(sb);
  startHtml(sb);
  appendHead(sb, content);

  sb.append("<body>")
   .append("<h1>")append(content.getHeading()).append("</h1>")
   .append("<section>").append(content.getText()).append("</section>")
  sb.append("<footer>").append(content.getCopyright()).append("</footer>");
  sb.append("</body>");
  sb.append("</html>");
  return sb.toString();
}
```

```
public String buildHtmlPage(PageContent content) {

   StringBuilder sb = new StringBuilder();
   appendDoctype(sb);
   startHtml(sb);
   appendHead(sb, content);
   appendBody(sb, content);

   sb.append("</html>");
   return sb.toString();
}
```

```
public String buildHtmlPage(PageContent content) {

    StringBuilder sb = new StringBuilder();
    appendDoctype(sb);
    startHtml(sb);
    appendHead(sb, content);
    appendBody(sb, content);
    endHtml(sb);

    return sb.toString();
}
```

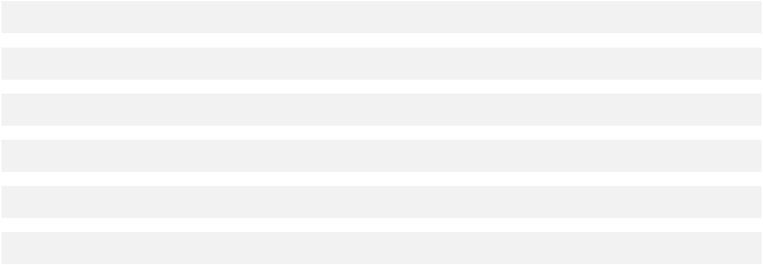
TESTABLE

```
public String buildHtmlPage(PageContent content) {

  StringBuilder sb = new StringBuilder();
  appendDoctype(sb);
  startHtml(sb);
  appendHead(sb, content);
  appendBody(sb, content);
  endHtml(sb);

  return sb.toString();
}
```

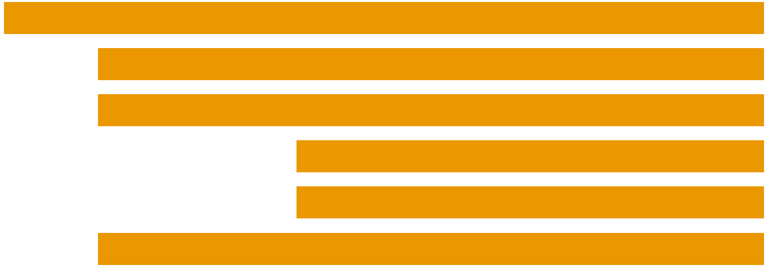
TESTABLE

```
public String buildHtmlPage(PageContent content) {

   StringBuilder sb = new StringBuilder();
   appendDoctype(sb);
   startHtml(sb);
   appendHead(sb, content);
   appendBody(sb, content);
   endHtml(sb);

   return sb.toString();
}
```

Bullet method



Nested method



```
public void package(Parcel parcel) {
 if (parcel.shouldPackage() && parcel.getState() != PACKAGED) {
   int w = parcel.getWidth()
   int h = parcel.getHeight();
   int surface = w*h;
   Color color = parcel.getPreferredPackagingColor();
   Package package = new Package(w, h, color);
   package.wrap(parcel);

   Taper taper = Taper.getInstance();
   if (parcel.mustTape() && taper.canTape(package)) {
     taper.tape(package);
   } else {
     throw new InsufficientTapeException();
   }
   dispatcher.dispatch(package);
 }
}
```

```
public void package(Parcel parcel) {
 if (parcel.shouldPackage() && parcel.getState() != PACKAGED) {
   int w = parcel.getWidth()
   int h = parcel.getHeight();
   int surface = w*h;
   Color color = parcel.getPreferredPackagingColor();
   Package package = new Package(w, h, color);
   package.wrap(parcel);

   Taper taper = Taper.getInstance();
   if (parcel.mustTape() && taper.canTape(package)) {
     taper.tape(package);
   } else {
     throw new InsufficientTapeException();
   }
   dispatcher.dispatch(package);
 }
}
```

```
public void package(Parcel parcel) {
 if (shouldPackage(parcel)) {
   int w = parcel.getWidth()
   int h = parcel.getHeight();
   int surface = w*h;
   Color color = parcel.getPreferredPackagingColor();
   Package package = new Package(w, h, color);
   package.wrap(parcel);

   Taper taper = Taper.getInstance();
   if (parcel.mustTape() && taper.canTape(package)) {
     taper.tape(package);
   } else {
     throw new InsufficientTapeException();
   }
   dispatcher.dispatch(package);
 }
}
```

```
public void package(Parcel parcel) {
  if (shouldPackage(parcel)) {
      int w = parcel.getWidth()
      int h = parcel.getHeight();
      int surface = w*h;
      Color color = parcel.getPreferredPackagingColor();
      Package package = new Package(w, h, color);
      package.wrap(parcel);

      Taper taper = Taper.getInstance();
      if (parcel.mustTape() && taper.canTape(package)) {
          taper.tape(package);
      } else {
          throw new InsufficientTapeException();
      }
      dispatcher.dispatch(package);
  }
}
```

```
public void package(Parcel parcel) {
  if (shouldPackage(parcel)) {
      Package package = packageParcel(parcel);

      Taper taper = Taper.getInstance();
      if (parcel.mustTape() && taper.canTape(package)) {
          taper.tape(package);
      } else {
          throw new InsufficientTapeException();
      }
      dispatcher.dispatch(package);
  }
}
```

```
public void package(Parcel parcel) {
 if (shouldPackage(parcel)) {
   Package package = packageParcel(parcel);

   Taper taper = Taper.getInstance();
   if (parcel.mustTape() && taper.canTape(package)) {
     taper.tape(package);
   } else {
     throw new InsufficientTapeException();
   }
   dispatcher.dispatch(package);
 }
}
```

```
public void package(Parcel parcel) {
  if (shouldPackage(parcel)) {
      Package package = packageParcel(parcel);

      if (shouldTape(package)) {
          tapePackage(package);
      } else {
          throw new InsufficientTapeException();
      }
      dispatcher.dispatch(package);
  }
}
```

```java
public void package(Parcel parcel) {
  if (shouldPackage(parcel)) {
      Package package = packageParcel(parcel);

      if (shouldTape(package)) {
          tapePackage(package);
      } else {
          throw new InsufficientTapeException();
      }
      dispatcher.dispatch(package);
  }
}
```

TESTABLE

TESTABLE

Moving to testable code

- Focused interfaces for test calls
- Focus on main purpose of class

FOCUS!

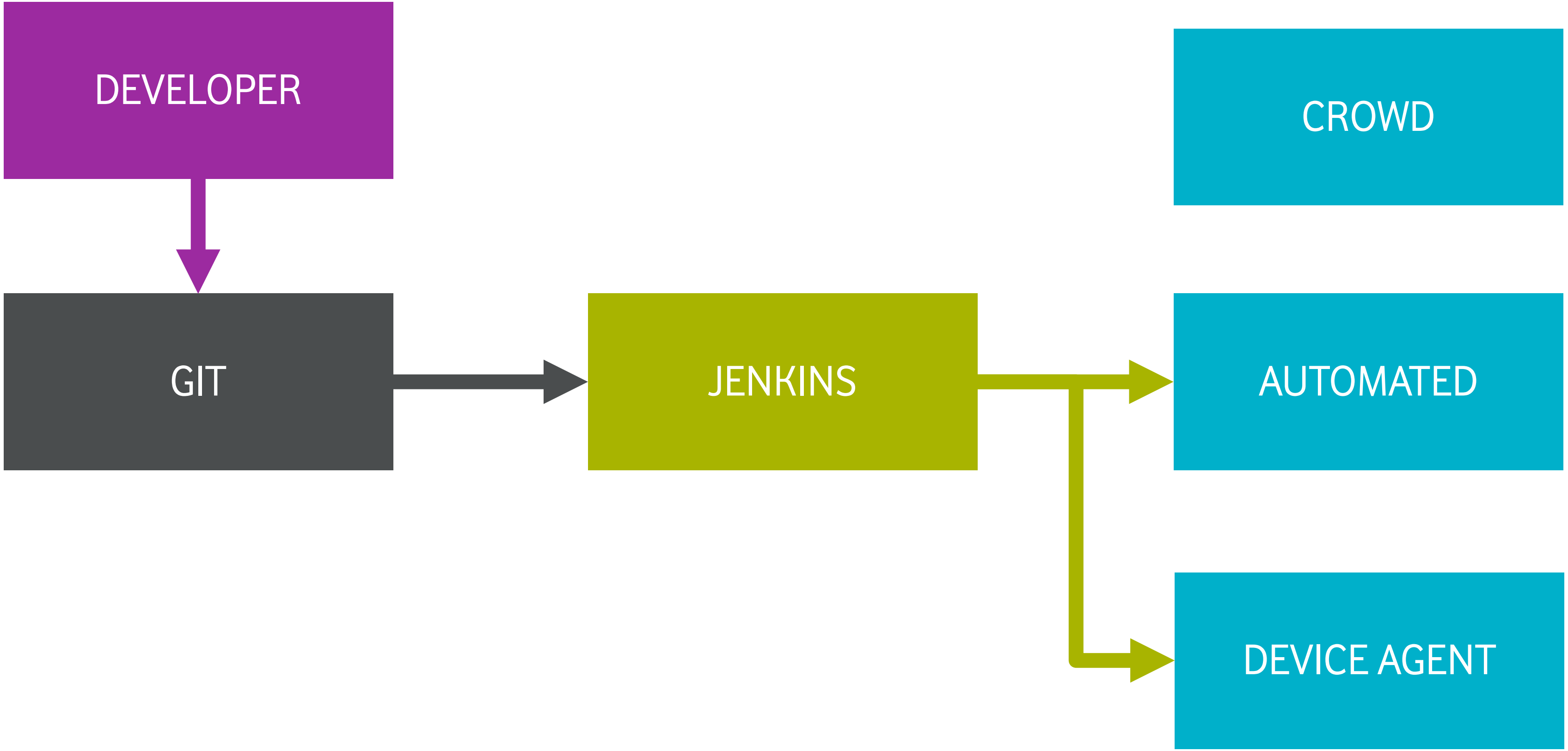
"Programming is the art of doing one thing at a time."

Michael C. Feathers - Working effectively with legacy code

Auxiliaries

- Get to know the code before doing changes
- Use Test-driven development
- Use Pull-Requests, review them yourself, eventually reject and simplify

Test automation



DEVELOPER

GIT

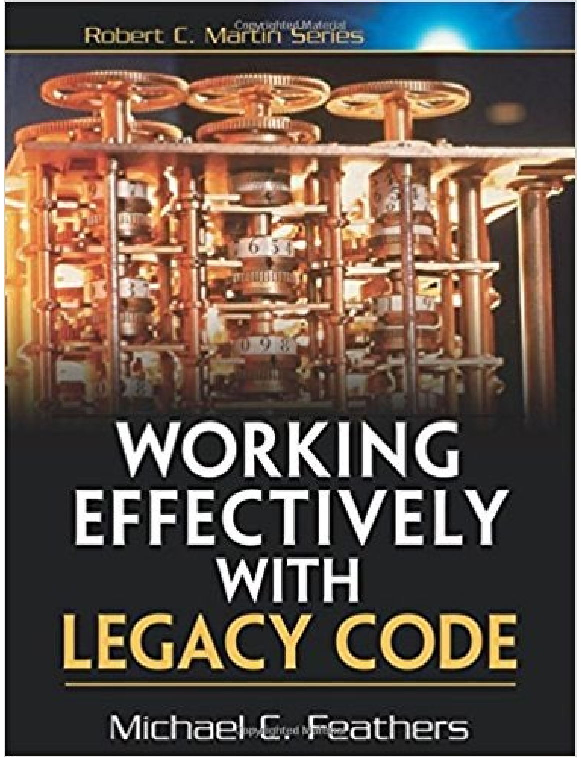
JENKINS

CROWD

AUTOMATED

DEVICE AGENT

Summary

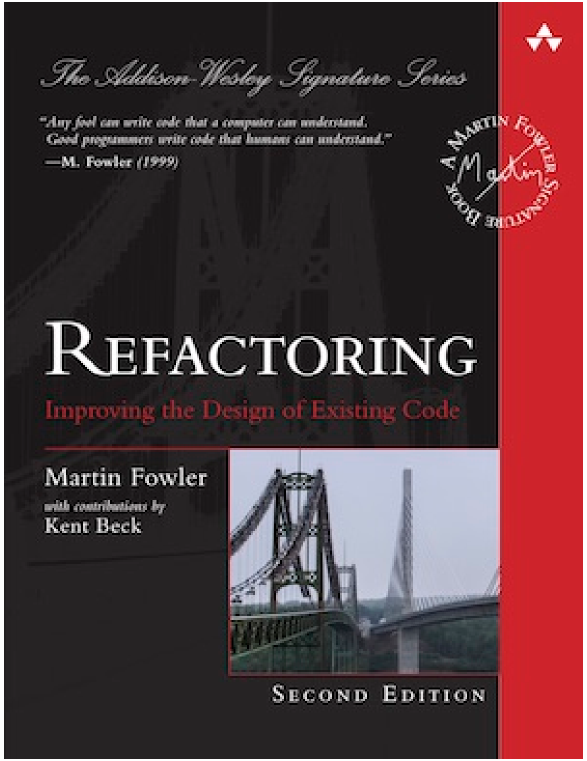


Robert C. Martin Series

Copyrighted Material

WORKING EFFECTIVELY WITH LEGACY CODE

Michael C. Feathers



The Addison-Wesley Signature Series

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
—M. Fowler (1999)

A Martin Fowler Signature Book

REFACTORING

Improving the Design of Existing Code

Martin Fowler
with contributions by
Kent Beck

SECOND EDITION

Thank you!

Anton Augsburg

@ungesehn