# Embedded Annotations Metrics

Version 0.1

This document represents feature metrics, which were partially proposed by (Andam et al., 2017; Entekhabi et al., 2019) and implemented beside others in the open-source tool FAXE[1], presented in (Schwarz et al., 2020).

Feautre metrics are a way to express the implementation of features in your software project on a more non technical level. Each project has it's own metric numbers and the metics provide an inside how features are structures. Influenced by the design of your software product, feature metric allow you to keep an eye on the raising, falling or stable complexity of it.

The examples for the different metrics are taken from the project Bitcoin-Wallet[2], annotated with embedded annotations by (Krüger et al., 2019) and adjusted to the embedded annotation specification[3] by (Schwarz et al., 2020). To show the concepts and how the different metrics are calculated, the code of Bitcoin-Wallet is extended for the specific metrics.

## Contents

[1]`https://bitbucket.org/easelab/faxe/`

[2]`https://bitbucket.org/easelab/datasetbitcoinwallet/`

[3]`https://bitbucket.org/easelab/faxe/src/master/specification/embedded_annotation_specification.pdf`

## 1 Scattering Degree

The scattering degree, or short SD, is a metric to represent how scattered a specific feature (i.e. metric is feature specific) is accross the file system. A widely scattered feature is more difficult to maintain and keep safe from side effects, than a feature being implemented in a few or even one location. (Apel et al., 2013; Liebig et al., 2010)
For this we extract the feature references (LPQ) from the embedded annotation markers and mapping files.

### 1.1 Example Text Annotations

Listings 1 till 3 show an example how the feature “DonateCoins” is scattered. The scattering degree for these three files is 4 - Three from WalletBalanceFragment.java, one from WalletActivity.java. WalletBalanceFragment.java and zero from WalletActivity.java. There are no further files mapped in the searched scope as well as no mappings files.

```java
@Override
public void onCreateOptionsMenu(final Menu menu, final MenuInflater inflater) {
    inflater.inflate(R.menu.wallet_balance_fragment_options, menu);
    super.onCreateOptionsMenu(menu, inflater);
}

@Override
public void onPrepareOptionsMenu(final Menu menu) {
    final Coin balance = viewModel.getBalance().getValue();
    final boolean hasSomeBalance = balance != null && !balance.isLessThan(Constants.SOME_BALANCE_THRESHOLD);
    //&begin[DonateCoins]
    menu.findItem(R.id.wallet_balance_options_donate)
            .setVisible(Constants.DONATION_ADDRESS != null && (!installedFromGooglePlay || hasSomeBalance));
    //&end[DonateCoins]
    super.onPrepareOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(final MenuItem item) {
    switch (item.getItemId()) {
        //&begin[DonateCoins]
    case R.id.wallet_balance_options_donate:
        handleDonate();
        return true;
        //&end[DonateCoins]
    }
    return super.onOptionsItemSelected(item);
}

//&begin[DonateCoins]
private void handleDonate() {
    //&begin[SendCoins]
    SendCoinsActivity.startDonate(activity, null, FeeCategory.ECONOMIC, 0); //&line[Fee]
    //&end[SendCoins]
}
//&end[DonateCoins]
```

Listing 1: WalletBalanceFragment.java

```java
public static void start(final Context context, final PaymentIntent paymentIntent) {
    start(context, paymentIntent, null, 0);
}

//&begin[DonateCoins]
//&begin[Fee]
public static void startDonate(final Context context, final Coin amount, final @Nullable FeeCategory feeCategory,
        final int intentFlags) {
    start(context, PaymentIntent.from(Constants.DONATION_ADDRESS,
            context.getString(R.string.wallet_donate_address_label), amount), feeCategory, intentFlags);
}
//&end[Fee]
//&end[DonateCoins]

@Override
protected void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

Listing 2: SendCoinsActivity.java

```java
    public boolean onOptionsItemSelected(final MenuItem item) {
        switch (item.getItemId()) {
            //&begin[RequestCoins]
        case R.id.wallet_options_request:
            handleRequestCoins();
            return true;
            //&end[RequestCoins]

            //&begin[SendCoins]
        case R.id.wallet_options_send:
            handleSendCoins();
            return true;
            //&begin[SendCoins]

        case R.id.wallet_options_scan:
            handleScan(null);
            return true;
```

Listing 3: WalletActivity.java

## 1.2 Example Mapping Annotations

Listing 4 till 14 present the scattering degree of text and mapping annotations. The scattering degree for this example is 4 - One from Constants.java, one from InactivityNotificationService.java (mapping _.feature-to-file) and two from all files inside /util (mapping _.feature-to-folder).

```
    /** User-agent to use for network access
    . */
    public static final String USER_AGENT =
    "Bitcoin Wallet";

    //&begin[SetDefault]
    /** Default currency to use if all
    default mechanisms fail. */
    public static final String
    DEFAULT_EXCHANGE_CURRENCY = "USD";
    //&end[SetDefault]

    //&begin[DonateCoins]
    /** Donation address for tip/donate
    action. */
    public static final String
    DONATION_ADDRESS = NETWORK_PARAMETERS.
    getId().equals(NetworkParameters.
    ID_MAINNET)
            ? "182
    Di1dqanjhNiphpNfrBRtKtdiUQtpgfb" : null;
    //&end[DonateCoins]

    //&begin[IssueReporter]
    /** Recipient e-mail address for reports
    . */
    public static final String REPORT_EMAIL
    = "bitcoin.wallet.developers@gmail.com";

    /** Subject line for manually reported
    issues. */
    public static final String
    REPORT_SUBJECT_ISSUE = "Reported issue";

    /** Subject line for crash reports. */
    public static final String
    REPORT_SUBJECT_CRASH = "Crash report";
    //&end[IssueReporter]

    public static final char CHAR_HAIR_SPACE
     = '\u200a';
```

Listing 4: Constants.java

```
InactivityNotificationService.java
DonateCoins
```

Listing 5: _.feature-to-file

```
DonateCoins
```

Listing 6: _.feature-to-folder

```
bitcoin-wallet
|-- Constant.java
|-- service
    |-- _.feature-to-file
    |-- InactivityNotificationService.java
|-- util
    |-- _.feature-to-folder
    |-- Base43.java
    |-- Bluetooth.java
```

Listing 7: Folder structure

## 2 Number of File Annotations

The metric Number of File Annotations represents the total number of file annotations, refferring to a specific feature. This means, it counts all appearances of the specific feature in the feature-to-file mapping files. Mapping whole files to features has the benefit, that with little effort and wihtout changing the source file, a good mapping of features to source code is possible. This works especially well for object oriented software projects where a file represents a class.

### 2.1 Example Mapping Annotations

```
InactivityNotificationService.java
DonateCoins
```

Listing 8: service_.feature-to-file

```
Base43.java, Bluetooth.java
Bluetooth
```

Listing 9: util_.feature-to-file

```
file1.java, file2.java
Feature1, Feature2
```

Listing 10: Special Case _.feature-to-file

```
bitcoin-wallet
|-- Constant.java
|-- service
    |-- _.feature-to-file
    |-- InactivityNotificationService.java
|-- util
    |-- _.feature-to-folder
    |-- _.feature-to-file
    |-- Base43.java
    |-- Bluetooth.java
```

Listing 11: Folder structure

Listing 8 shows the mapping of feature "DonateCoins" to the file "InactivityNotificationService.java". Considering that there are no further file mappings for "DonateCoins", the metric Number of File Annotations is one.

Listing 9 shows the mapping of feature "Bluetooth" to the file "Base43.java" and "Bluetooth.java". Considering that there are no further file mappings for "Bluetooth", the metric Number of File Annotations is two.

Listinig 10 shows the special case of a many-to-many relationship in the feature-to-file mapping. I.e. that both files are mapped with both features. Therefore, the metric Number of File Annotations is two per feature.

In case the specific feature is not mapped to any file, a zero will be returned.

## 3 Number of Folder Annotations

The metric Number of Folder Annotations represents the total number of folder annotations directly referencing the feature. This means, the number of appearances of a specific feature in all feature-to-folder mapping files. Mapping whole folders to features has the benefit, that with little effort and wihtout changing the source file, a good mapping of features to source code is possible. This works especially well for higher level features and the folder structure reflects the idea of features.

### 3.1 Example Mapping Annotations

```
DonateCoins, FeatureA
```

Listing 12: service_.feature-to-folder

```
Base43, Bluetooth, FeatureA
```

Listing 13: util_.feature-to-folder

```
bitcoin-wallet
|-- Constant.java
|-- service
    |-- _.feature-to-folder
    |-- InactivityNotificationService.java
|-- util
    |-- _.feature-to-folder
    |-- Base43.java
    |-- Bluetooth.java
```

Listing 14: Folder structure

Listing 12 shows the mapping of features “DonateCoins” and “FeatureA” to its folder “service”. Listing 13 mapps features “Base43”, “Bluetooth”, and “FeatureA” to its folder “util”.
By asking for the metric Number of Folder Annotations for feature “DonateCoins” we will receive one. Same metric value of one for features “Base43” and ‘’Bluetooth”. As feature “FeatureA” is annotating two files, the returned value is two.

## 4 Tangling Degree

The Tangling Degree, or short TD, is a metric to represent how tangled a specific feature (i.e. metric is feature specific) is with other features. A lower Tangling Degree is good, as it indicates less potential interactions with other features. (Apel et al., 2013; Liebig et al., 2010)
For this we extract the feature references (LPQ) from the embedded annotation markers and check which features the searched one is tangled with. The tangling with a specific feature is counted uniquely. I.e. independent if the searched feature is tangled once or multiple times by a specific other feature, the Tangling Degree is increased by one.
**Text Annotations:** The smallest unit to count is the file level. I.e. when the searched feature is present the tangling degree is increased per other individual feature.
The tangling degree gets as well increased by the overarching feature-to-file and feature-to-folder mappings and the there additional appearing features.
**File Annotations:** In case the search is conducted on a folder, all files where the searched feature appears are counted.
**Folder Annotations:** In case of a folder annotation, all features inside this folder are tangled with the searched feature.

Tangling Degree = 0 : Feature not present. Therefore no tangling.
Tangling Degree = 0 : Feature present, but no tangling with any other feature.
Tangling Degree = 1 : Feature tangled with one other feature.

### 4.1 Example Text Annotations

```
Line 123: //&begin[Bluetooth]
Line 127: //&end[Bluetooth]
Line 192: //&begin[RestoreWallet]
Line 205: //&end[RestoreWallet]
Line 219: WalletUtils.autoBackupWallet(WalletApplication.this, wallet); //&line[BackupWallet]
Line 223: config.armBackupReminder(); //&line[BackupReminder]
Line 264: BlockchainService.resetBlockchain(this);  //&line[ResetBlockChain]
Line 273: WalletUtils.autoBackupWallet(this, newWallet);  //&line[BackupWallet]
Line 281: //&begin[BackupWallet]
Line 282:     //&begin[Codecs]
Line 283:     if (filename.startsWith(Constants.Files.WALLET_KEY_BACKUP_BASE58) //&line[base58]
Line 284:     //&end[Codecs]
Line 291: //&end[BackupWallet]
Line 300: //&begin[NotifyReceived]
Line 308: //&end[NotifyReceived]
Line 326: BlockchainService.broadcastTransaction(this, tx); //&line[BlockchainSync]
```

Listing 15: WalletApplication.java

The Tangling Degree of Listing 15 is 8 . Available features: [Bluetooth, Codecs, BackupWallet, base58, BackupReminder, NotifyReceived, RestoreWallet, ResetBlockChain]. Constrain is, that there is no feature mapping above in the feature-to-file and feature-to-folder mappings.

## 4.2 Example Mapping Annotations

```java
import java.io.File;

//&begin[AppLog]
/**
 * @author Andreas Schildbach
 */
public class Logging {
    ...
}
//&end[AppLog]
```

Listing 16: Logging.java

```
Logging.java WalletBalanceWidgetProvider.java
AppLog BitcoinBalance
```

Listing 17: _.feature-to-file

```
Bluetooth::Codecs
```

Listing 18: _.feature-to-folder

The file Logging.java contains itself only the feature “AppLog” and has therefore no tangling on the text level. In this example, the file Logging.java itself is mapped by feature-to-file mapping to the features “ApplLog” and “BitcoinBalance”. At the same time the file “WalletBalanceWidgetProvider.java” is mapped to this feature. I.e. all features in this file are tangled with it. With this the tangling degree for “AppLog” is 2 : [AppLog, BitcoinBalance] plus all unique features from “WalletBalanceWidgetProvider.java”. Constrain is, that there is no other feature-to-file mapping and feature-to-folder mappings.

By searching for the feature “Bluetooth::Codecs” all features mapped to the folder and inside the folder get counted.

# 5 Lines of Feature Code

The metric Lines of Feature Code represents the count of code lines belonging directly, or indirectly annotated to a specific feature. With this metric tells us, it can be determined if a small or large portion of the source code is annotated to this feature.
The metric considers all annotated elements, including source code, feature-to-file mapping and feature-to-folder mapping. For the calculation of text fragments, all lines from the &begin till &end line are considered:
Example - 5 lines of fragment code

```
//&begin[BlockchainSync]
public final Date bestChainDate;
public final int bestChainHeight;
public final boolean replaying;
//&end[BlockchainSync]
```

This is for stability reasons to cover all ways of how text fragment annotations could be written.

```
/*&begin[BlockchainSync] * /public final bestValue;
public final Date bestChainDate;
public final int bestChainHeight;
public final boolean replaying;
public final boolean play //&end[BlockchainSync]
```

## 5.1 Example Text Annotations

For the following examples only the three showns files will be considered. I.e. there are no other text annotations or mapping files to these features.
The feature “DonateCoins” has a 20 Lines of code: 16 from Listing 19, lines 194 till 197, 204 till 208, 213 till 219. Plus 8 lines from Listing 20, lines 60 till 67.

The feature “Fee” has a 7 Lines of code: 1 from Listing 19, line 216. Plus 6 lines from Listing 21, lines 61 till 66.

```java
@Override
public void onCreateOptionsMenu(final Menu menu, final MenuInflater inflater) {
    inflater.inflate(R.menu.wallet_balance_fragment_options, menu);
    super.onCreateOptionsMenu(menu, inflater);
}

@Override
public void onPrepareOptionsMenu(final Menu menu) {
    final Coin balance = viewModel.getBalance().getValue();
    final boolean hasSomeBalance = balance != null && !balance.isLessThan(Constants.SOME_BALANCE_THRESHOLD);
    //&begin[DonateCoins]
    menu.findItem(R.id.wallet_balance_options_donate)
            .setVisible(Constants.DONATION_ADDRESS != null && (!installedFromGooglePlay || hasSomeBalance));
    //&end[DonateCoins]
    super.onPrepareOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(final MenuItem item) {
    switch (item.getItemId()) {
        //&begin[DonateCoins]
    case R.id.wallet_balance_options_donate:
        handleDonate();
        return true;
        //&end[DonateCoins]
    }
    return super.onOptionsItemSelected(item);
}

//&begin[DonateCoins]
private void handleDonate() {
    //&begin[SendCoins]
    SendCoinsActivity.startDonate(activity, null, FeeCategory.ECONOMIC, 0); //&line[Fee]
    //&end[SendCoins]
}
//&end[DonateCoins]
```

Listing 19: WalletBalanceFragment.java

```java
public static void start(final Context context, final PaymentIntent paymentIntent) {
    start(context, paymentIntent, null, 0);
}

//&begin[DonateCoins]
//&begin[Fee]
public static void startDonate(final Context context, final Coin amount, final @Nullable FeeCategory feeCategory,
        final int intentFlags) {
    start(context, PaymentIntent.from(Constants.DONATION_ADDRESS,
            context.getString(R.string.wallet_donate_address_label), amount), feeCategory, intentFlags);
}
//&end[Fee]
//&end[DonateCoins]

@Override
protected void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

Listing 20: SendCoinsActivity.java

```java
    public boolean onOptionsItemSelected(final MenuItem item) {
        switch (item.getItemId()) {
            //&begin[RequestCoins]
        case R.id.wallet_options_request:
            handleRequestCoins();
            return true;
            //&end[RequestCoins]

            //&begin[SendCoins]
        case R.id.wallet_options_send:
            handleSendCoins();
            return true;
            //&begin[SendCoins]

        case R.id.wallet_options_scan:
            handleScan(null);
            return true;
```

Listing 21: WalletActivity.java

## 5.2 Example Mapping Annotations

For the following examples, only the three shown sources will be considered. I.e. there are no other text annotations or mapping files to these features.
The feature "DonateCoins" has a Lines of code from:

- 5 from Listing 22, lines 153 till 157
- From Listing 23 all lines of file "InactivityNotificationService.java"
- From Listing 24, all files shown in Listing 25 below the folder "util". I.e. in this case all

lines from the files “Base43.java” and “Bluetooth.java”.

```
    /** User−agent to use for network access
    . */
    public static final String USER_AGENT =
    "Bitcoin Wallet";

    //&begin[SetDefault]
    /** Default currency to use if all
    default mechanisms fail. */
    public static final String
    DEFAULT_EXCHANGE_CURRENCY = "USD";
    //&end[SetDefault]

    //&begin[DonateCoins]
    /** Donation address for tip/donate
    action. */
    public static final String
    DONATION_ADDRESS = NETWORK_PARAMETERS.
    getId().equals(NetworkParameters.
    ID_MAINNET)
            ? "182
    Di1dqanjhNiphpNfrBRtKtdiUQtpgfb" : null;
    //&end[DonateCoins]

    //&begin[IssueReporter]
    /** Recipient e−mail address for reports
    . */
    public static final String REPORT_EMAIL
    = "bitcoin.wallet.developers@gmail.com";

    /** Subject line for manually reported
    issues. */
    public static final String
    REPORT_SUBJECT_ISSUE = "Reported issue";

    /** Subject line for crash reports. */
    public static final String
    REPORT_SUBJECT_CRASH = "Crash report";
    //&end[IssueReporter]

    public static final char CHAR_HAIR_SPACE
     = '\u200a';
```

Listing 22: Constants.java

```
InactivityNotificationService.java
DonateCoins
```

Listing 23: _.feature-to-file

```
DonateCoins
```

Listing 24: _.feature-to-folder

```
bitcoin−wallet
|−− Constant.java
|−− service
    |−− _.feature−to−file
    |−− InactivityNotificationService.java
|−− util
    |−− _.feature−to−folder
    |−− Base43.java
    |−− Bluetooth.java
```

Listing 25: Folder structure

# 6 Nesting Depths Average / Maximum / Minimum

The metric Nesting Depths of annotations has three dimensions: Maximum (MaxND), Minimum (MinND), and Average (AvgND) nesting depth.
Nesting depth expresses the fact, how deep a specific feature annotation is nested - completely or partially - with another feature annotation. The depth of nesting is 1, when the annotations is neither inside another textual annotation (e.g. &begin / &end) nor the containing file or any (parent-)folder contains a feature annotation. Each textual, file and folder annotation increases the nesting depth by 1.

## 6.1 Example Text Annotations

For the following examples only the three showns files will be considered. I.e. there are no other text annotations or mapping files to these features.
Feature “DonateCoins” appears in Listing 26 at

- Lines 194 till 197 with a nesting degree of one.
- Lines 204 till 208 with a nesting degree of one.
- Lines 213 till 219 with a nesting degree of one.

And appears in Listing 27 at

- Lines 60 till 67 with a nesting degree of one.

With that the Average Nesting Degree is one, while the Maximum Nesting Degree is one, and the Minimun Nesting Degree is one, too.

Feature “Fee” appears in Listing 26 at

- Line 216 with a nesting degree of three.

And appears in Listing 27 at

- Lines 61 till 66 with a nesting degree of two.

With that the Average Nesting Degree is two dot five, while the Maximum Nesting Degree is three, and the Minimun Nesting Degree is two.

```java
@Override
public void onCreateOptionsMenu(final Menu menu, final MenuInflater inflater) {
    inflater.inflate(R.menu.wallet_balance_fragment_options, menu);
    super.onCreateOptionsMenu(menu, inflater);
}

@Override
public void onPrepareOptionsMenu(final Menu menu) {
    final Coin balance = viewModel.getBalance().getValue();
    final boolean hasSomeBalance = balance != null && !balance.isLessThan(Constants.SOME_BALANCE_THRESHOLD);
    //&begin[DonateCoins]
    menu.findItem(R.id.wallet_balance_options_donate)
            .setVisible(Constants.DONATION_ADDRESS != null && (!installedFromGooglePlay || hasSomeBalance));
    //&end[DonateCoins]
    super.onPrepareOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(final MenuItem item) {
    switch (item.getItemId()) {
        //&begin[DonateCoins]
    case R.id.wallet_balance_options_donate:
        handleDonate();
        return true;
        //&end[DonateCoins]
    }
    return super.onOptionsItemSelected(item);
}

//&begin[DonateCoins]
private void handleDonate() {
    //&begin[SendCoins]
    SendCoinsActivity.startDonate(activity, null, FeeCategory.ECONOMIC, 0); //&line[Fee]
    //&end[SendCoins]
}
//&end[DonateCoins]
```

Listing 26: WalletBalanceFragment.java

```java
public static void start(final Context context, final PaymentIntent paymentIntent) {
    start(context, paymentIntent, null, 0);
}

//&begin[DonateCoins]
//&begin[Fee]
public static void startDonate(final Context context, final Coin amount, final @Nullable FeeCategory feeCategory,
        final int intentFlags) {
    start(context, PaymentIntent.from(Constants.DONATION_ADDRESS,
            context.getString(R.string.wallet_donate_address_label), amount), feeCategory, intentFlags);
}
//&end[Fee]
//&end[DonateCoins]

@Override
protected void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

Listing 27: SendCoinsActivity.java

```java
    public boolean onOptionsItemSelected(final MenuItem item) {
        switch (item.getItemId()) {
            //&begin[RequestCoins]
        case R.id.wallet_options_request:
            handleRequestCoins();
            return true;
            //&end[RequestCoins]

            //&begin[SendCoins]
        case R.id.wallet_options_send:
            handleSendCoins();
            return true;
            //&begin[SendCoins]

        case R.id.wallet_options_scan:
            handleScan(null);
            return true;
```

Listing 28: WalletActivity.java

## 6.2 Example Mapping Annotations

For the following examples, only the three shown sources will be considered. I.e. there are no other text annotations or mapping files to these features.

Feature “DonateCoins” appears in Listing 29 at

- Lines 153 till 157 with a nesting degree of one.

And appears in Listing 31 at

- The mapped file of Listing 31 has a nesting degree of one by itself.
- Lines 97 till 100 with a nesting degree of two.
- Lines 113 till 115 with a nesting degree of three.

In Listing 31, all nestings get increased by one due to the feature-to-file mapping of Listing 30 According to the feature-to-folder mapping in Listing 32, the files “Base43.java” and “Bluetooth.java” have a nesting degree of one each.

With that the Average Nesting Degree is one dot five (1+1+2+3+1+1), while the Maximum Nesting Degree is three, and the Minimum Nesting Degree is one.

Definition fitting? Do mapped files/folders be considered different?

```java
    /** User-agent to use for network access. */
    public static final String USER_AGENT = "Bitcoin Wallet";

    //&begin[SetDefault]
    /** Default currency to use if all default mechanisms fail. */
    public static final String DEFAULT_EXCHANGE_CURRENCY = "USD";
    //&end[SetDefault]

    //&begin[DonateCoins]
    /** Donation address for tip/donate action. */
    public static final String DONATION_ADDRESS = NETWORK_PARAMETERS.getId().equals(NetworkParameters.ID_MAINNET)
            ? "182Di1dqanjhNiphpNfrBRtKtdiUQtpgfb" : null;
    //&end[DonateCoins]

    //&begin[IssueReporter]
    /** Recipient e-mail address for reports. */
    public static final String REPORT_EMAIL = "bitcoin.wallet.developers@gmail.com";

    /** Subject line for manually reported issues. */
    public static final String REPORT_SUBJECT_ISSUE = "Reported issue";

    /** Subject line for crash reports. */
    public static final String REPORT_SUBJECT_CRASH = "Crash report";
    //&end[IssueReporter]

    public static final char CHAR_HAIR_SPACE = '\u200a';
```

Listing 29: Constants.java

```
InactivityNotificationService.java
DonateCoins
```

Listing 30: _.feature-to-file

```java
    else if (ACTION_DISMISS_FOREVER.equals(intent.getAction()))
        handleDismissForever();
    //&begin[DonateCoins]
    else if (ACTION_DONATE.equals(intent.getAction()))
        handleDonate(wallet);
    //&end[DonateCoins]
    else
        handleMaybeShowNotification(wallet);
}

//&begin[BitcoinBalance]
private void handleMaybeShowNotification(final Wallet wallet) {
    final Coin estimatedBalance = wallet.getBalance(BalanceType.ESTIMATED_SPENDABLE);

    if (estimatedBalance.isPositive()) {
        log.info("detected balance, showing inactivity notification");

        final Coin availableBalance = wallet.getBalance(BalanceType.AVAILABLE_SPENDABLE);
        //&begin[DonateCoins]
        final boolean canDonate = Constants.DONATION_ADDRESS != null && !availableBalance.isLessThan(Constants.SOME_BALANCE_THRESHOLD);
        //&end[DonateCoins]
        ...
//&end[BitcoinBalance]
```

Listing 31: InactivityNotificationService.java

```
DonateCoins
```

Listing 32: _.feature-to-folder

```
bitcoin-wallet
|-- Constant.java
|-- service
    |-- _.feature-to-file
    |-- InactivityNotificationService.java
|-- util
    |-- _.feature-to-folder
    |-- Base43.java
    |-- Bluetooth.java
```

Listing 33: Folder structure

# 7 Number of Annotated Files

The metric Number of File Annotations provides for a specific feature the total number of file and folder annotations referring to it.

## 7.1 Example Mapping Annotations

```
import java.io.File;

//&begin[AppLog]
/**
 * @author Andreas Schildbach
 */
public class Logging {
   ...
}
//&end[AppLog]
```

Listing 34: Logging.java

```
Logging.java WalletBalanceWidgetProvider.java
AppLog BitcoinBalance
```

Listing 35: _.feature-to-file

```
Bluetooth::Codecs
```

Listing 36: _.feature-to-folder

For feature “AppLog” the metric of Number of Annotated Files is two. One time with “Logging.java” and one time with “WalletBalanceWidgetProvider.java”.
Feature “BitcoinBalance” is identical to feature “AppLog”.
Feature “Bluetooth::Codecs” has a metric of Number of Annotated Files equal to the number of files and folders below the mapped folder.

# 8 Number of Features

The metric Number of Features provides the total number of different features, used text, file, and folder annotations.

## 8.1 Example Text Annotations

For the following examples only the three showns files will be considered. I.e. there are no other text annotations or mapping files to these features.
The metric Number of Features for the Listings 37 till 39 is four. In total four features have beend used: DonateCoins, Fee, RequestCoins, and SendCoins .

```java
@Override
public void onCreateOptionsMenu(final Menu menu, final MenuInflater inflater) {
    inflater.inflate(R.menu.wallet_balance_fragment_options, menu);
    super.onCreateOptionsMenu(menu, inflater);
}

@Override
public void onPrepareOptionsMenu(final Menu menu) {
    final Coin balance = viewModel.getBalance().getValue();
    final boolean hasSomeBalance = balance != null && !balance.isLessThan(Constants.SOME_BALANCE_THRESHOLD);
    //&begin[DonateCoins]
    menu.findItem(R.id.wallet_balance_options_donate)
            .setVisible(Constants.DONATION_ADDRESS != null && (!installedFromGooglePlay || hasSomeBalance));
    //&end[DonateCoins]
    super.onPrepareOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(final MenuItem item) {
    switch (item.getItemId()) {
        //&begin[DonateCoins]
    case R.id.wallet_balance_options_donate:
        handleDonate();
        return true;
        //&end[DonateCoins]
    }
    return super.onOptionsItemSelected(item);
}

//&begin[DonateCoins]
private void handleDonate() {
    //&begin[SendCoins]
    SendCoinsActivity.startDonate(activity, null, FeeCategory.ECONOMIC, 0); //&line[Fee]
    //&end[SendCoins]
}
//&end[DonateCoins]
```

Listing 37: WalletBalanceFragment.java

```java
public static void start(final Context context, final PaymentIntent paymentIntent) {
    start(context, paymentIntent, null, 0);
}

//&begin[DonateCoins]
//&begin[Fee]
public static void startDonate(final Context context, final Coin amount, @Nullable FeeCategory feeCategory,
        final int intentFlags) {
    start(context, PaymentIntent.from(Constants.DONATION_ADDRESS,
            context.getString(R.string.wallet_donate_address_label), amount), feeCategory, intentFlags);
}
//&end[Fee]
//&end[DonateCoins]

@Override
protected void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

Listing 38: SendCoinsActivity.java

```java
    public boolean onOptionsItemSelected(final MenuItem item) {
        switch (item.getItemId()) {
            //&begin[RequestCoins]
        case R.id.wallet_options_request:
            handleRequestCoins();
            return true;
            //&end[RequestCoins]

            //&begin[SendCoins]
        case R.id.wallet_options_send:
            handleSendCoins();
            return true;
            //&begin[SendCoins]

        case R.id.wallet_options_scan:
            handleScan(null);
            return true;
```

Listing 39: WalletActivity.java

## 8.2 Example Mapping Annotations

For the following examples, only the three shown sources will be considered. I.e. there are no other text annotations or mapping files to these features.
The metric Number of Features for the Listings 40 till 42 is three. In total three features have beend used: DonateCoins, IssueReporter, and SetDefault

```
    /** User-agent to use for network access
    . */
    public static final String USER_AGENT =
    "Bitcoin Wallet";

    //&begin[SetDefault]
    /** Default currency to use if all
    default mechanisms fail. */
    public static final String
    DEFAULT_EXCHANGE_CURRENCY = "USD";
    //&end[SetDefault]

    //&begin[DonateCoins]
    /** Donation address for tip/donate
    action. */
    public static final String
    DONATION_ADDRESS = NETWORK_PARAMETERS.
    getId().equals(NetworkParameters.
    ID_MAINNET)
            ? "182
    Di1dqanjhNiphpNfrBRtKtdiUQtpgfb" : null;
    //&end[DonateCoins]

    //&begin[IssueReporter]
    /** Recipient e-mail address for reports
    . */
    public static final String REPORT_EMAIL
    = "bitcoin.wallet.developers@gmail.com";

    /** Subject line for manually reported
    issues. */
    public static final String
    REPORT_SUBJECT_ISSUE = "Reported issue";

    /** Subject line for crash reports. */
    public static final String
    REPORT_SUBJECT_CRASH = "Crash report";
    //&end[IssueReporter]

    public static final char CHAR_HAIR_SPACE
     = '\u200a';
```

Listing 40: Constants.java

```
InactivityNotificationService.java
DonateCoins
```

Listing 41: _.feature-to-file

```
DonateCoins
```

Listing 42: _.feature-to-folder

```
bitcoin-wallet
|-- Constant.java
|-- service
    |-- _.feature-to-file
    |-- InactivityNotificationService.java
|-- util
    |-- _.feature-to-folder
    |-- Base43.java
    |-- Bluetooth.java
```

Listing 43: Folder structure

# 9 Average Feature Lines of Feature Code

The metric Average Feature Lines of Feature Code calculates the lines of feautre code per existing feature, as shown in Chapter 5 “Lines of Feature Code” and divides this result by the number of existing features.

## 10 Average Feature Nesting Depth

The metric Average Feature Nesting Depth calculates the metric Nesting Depths Average of all feautres, as shown in Chapter 6 " Nesting Depths Average / Maximum / Minimum" and divides this result by the number of existing features.

## 11 Average Feature Scattering Degree

The metric Average Feature Scattering Degree calculates the metric Scattering Degree of all feautres, as shown in Chapter 1 “Scattering Degree” and divides this result by the number of existing features.

## References

Andam, Berima et al. (2017). “FLOrIDA: Feature LOcatIon DAshboard for Extracting and Visualizing Feature Traces”. In: *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-Intensive Systems*. VAMOS ’17. Eindhoven, Netherlands: Association for Computing Machinery, pp. 100–107. ISBN: 9781450348119. DOI: `10.1145/3023956.3023967`. URL: `https://doi.org/10.1145/3023956.3023967`.

Apel, Sven et al. (2013). *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated. ISBN: 3642375200.

Entekhabi, Sina et al. (Sept. 2019). “Visualization of Feature Locations with the Tool FeatureDashboard”. In: pp. 1–4. ISBN: 978-1-4503-6668-7. DOI: `10.1145/3307630.3342392`.

Krüger, Jacob et al. (2019). “Where is my feature and what is it about? A case study on recovering feature facets”. In: *Journal of Systems & Software* 152, pp. 239–253. ISSN: 01641212. DOI: `10.1016/j.jss.2019.01.057`. URL: `http://search.ebscohost.com/login.aspx?direct=true&AuthType=sso&db=buh&AN=135661128&site=eds-live&scope=site&custid=s3911979&authtype=sso&group=main&profile=eds`.

Liebig, J. et al. (2010). “An analysis of the variability in forty preprocessor-based software product lines”. In: *2010 ACM/IEEE 32nd International Conference on Software Engineering*. Vol. 1, pp. 105–114.

Schwarz, Tobias, Wardah Mahmood, and Thorsten Berger (2020). “A Common Notation and Tool Support for Embedded Feature Annotations”. In: *Proceedings of the 24th ACM International Systems and Software Product Line Conference*. SPLC ’20. Montreal, Canada.