Etude empirique de la relation entre les appels vers l'API Android et les défauts dans le code

PIERRE GÉRARD

DIRO, UNIVERSITÉ DE MONTRÉAL

Plan

Description de l’expérience réalisé.

Collection des données.

Résultats et analyse.

Questions de recherche

Quelles sont les relations entre les appels vers l'API Android et les défauts dans le code ?

Pour cela on va répondre aux sous-questions suivantes au niveau des classes.

- Est ce qu'il y a une corrélation entre les deux ?
- Est ce qu'il y a une différence entre les classes avec et sans appels en terme de défauts ?
- Est ce qu'il y a une différence entre les classes avec et sans défauts en terme de d'appels?
- Est-il possible d'estimer une valeur de défauts pour un certain nombre d'appel ?
- Est ce que la taille des classes a-t-elle aussi une influence ?

Variables

Variables indépendantes:

- Taille de la classe
- Les appels à l'API par classe

Variable dépendantes

- Défauts par classe

Schéma du modèle

Collection des données

Trois types de données:

Appels Client->Api

- Basé sur une analyse statique du code

Taille des classes

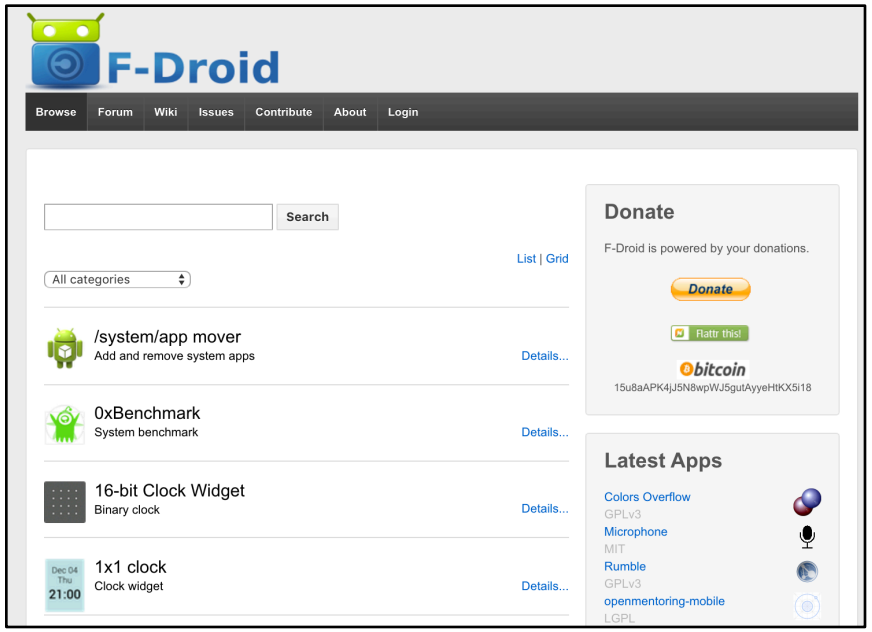
- Basé sur une analyse statique du code

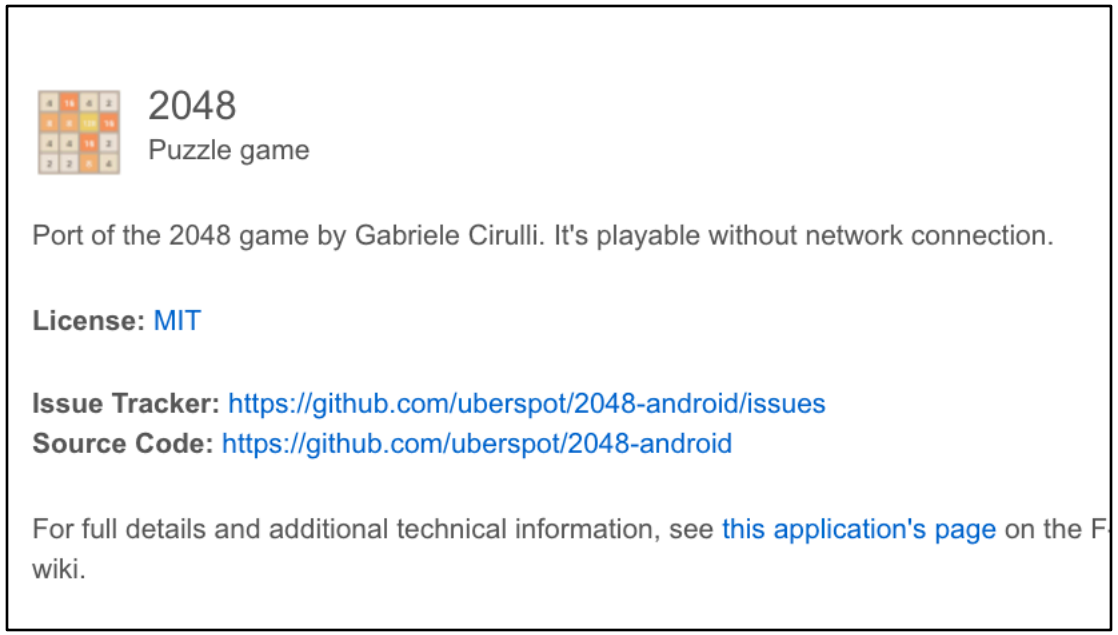
Défauts dans le code

- Basé sur le feedback des utilisateurs/développeurs

Source de donnée : F-droid

Repository de nombreuses applications open-source Android





Source de donnée : F-droid

Pas moyen d'exporter les données

- Ecriture d'un script python (lib http + parser html)
- Création d'un *.json* contenant les liens vers les codes sources, les bugs trackers , et le jar

Extrait :


```
{
  "RF Analyzer": {
    "src": "https://github.com/demantz/RFAnalyzer",
    "issue": "https://github.com/demantz/RFAnalyzer/issues",
    "url": "https://f-droid.org/repository/browse/?fdid=com.mantz_it.rfanalyzer&fdpage=41"
  },
```


- Cela nous laisse avec une liste de 1647 applications

Sélection des données

Parmi l'ensemble des applications, il faudrait en choisir un nombre raisonnable.

On donne la priorité aux applications avec beaucoup de défauts .

- On écrit un autre script python qui va ajouter au fichier précédant le nombre de défauts par appli

GitHub n'autorise que 5000 requêtes authentifiées par heure.

- (et ne font pas facilement d'exception)

➢ On se limite à 14 applications et 26000 issues (6 heures de traitement)

- On écrit encore un autre script python qui va cloner les repos et récupérer les apk.

Analyse statique du code

Pour l'analyse statique on utilise : **java-callgraph** (Georgios Gousios, Institute for Computing and Information Sciences, Radboud Universiteit Nijmegen)

Dalvik VM format -> Java VM format -> Graphe d'appels

Exemple

```
M:de.danoeh.antennapod.activity.AudioplayerActivity$8:<init> (O)java.lang.Object:<init>
M:de.danoeh.antennapod.activity.AudioplayerActivity$8:onLongClick (S)com.aocate.media.MediaPlayer$1:showDialog
C:de.danoeh.antennapod.core.asynctask.FeedRemover de.danoeh.antennapod.core.asynctask.FeedRemover
C:de.danoeh.antennapod.core.asynctask.FeedRemover android.os.AsyncTask
C:de.danoeh.antennapod.core.asynctask.FeedRemover de.danoeh.antennapod.core.asynctask.FeedRemover$1
C:de.danoeh.antennapod.core.asynctask.FeedRemover java.lang.InterruptedException
```

Appels Client->Api

On écrit un script python qui va :

1. Appeler un sub-process *java-callgraph* et récupérer le graphe d'appels,
2. Pour chaque appel, identifier la classe appelante et si c'est un appel vers une classe de l'API Android,
3. Additionner le nombre d'appels vers des classes différentes de l'API.

Exemple:

```
- de.danoeh.antennapod.activity.MainActivity ->  [android.support.v7.app.ActionBarActivity android.os.AsyncTask android.content.SharedPreferences android.
content.Intent android.support.v4.app.FragmentManager android.support.v4.app.FragmentTransaction android.content.SharedPreferences$Editor android.support.
v4.app.Fragment android.support.v7.app.ActionBar android.support.v7.app.ActionBarDrawerToggle android.view.MenuItem android.widget.AdapterView$AdapterCont
extMenuInfo android.view.View android.widget.ListView android.app.AlertDialog android.support.v7.widget.Toolbar android.support.v4.widget.DrawerLayout and
roid.os.Bundle android.os.Handler android.view.MenuInflater android.view.ContextMenu android.app.ProgressDialog]
```

- On obtient : AntennaPod[MainActivity] -> 17

Taille des classes

Assez similaire aux appels, on écrit un script python qui va :

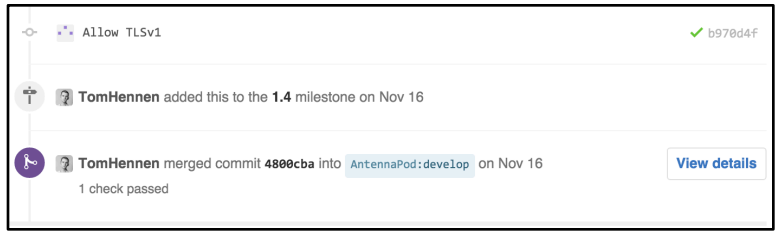
1. Appeler un sub-process *java-callgraph* et récupérer le graphe d'appels,
2. Pour chaque classe, on identifie ses méthodes apparaissant dans le graphe d'appel.
3. On les additionne et on stocke les résultats.

Défauts dans le code (1/2)

Ecrire un script python qui va :

1. Pour chaque applications, récupérer l'ensemble des issues ouvert sur GitHub via l'API Gihub,
2. Pour chaque issue, récupérer l'ensemble des *events,*
3. Pour chaque *event*, récupérer, si il existe, le commit associé ou la *pull request* associé.

Issue avec une réference vers une pull request



Pull Request avec référence vers le commit

Défauts dans le code (2/2)

4. Dans le code source, pour chaque commit, fair un sub-process *git show et* identifier les classes concernés
5. Ajouter le score $\frac{1}{nombre\ de\ classes\ concernés}$ aux classes concernés

Exemple :

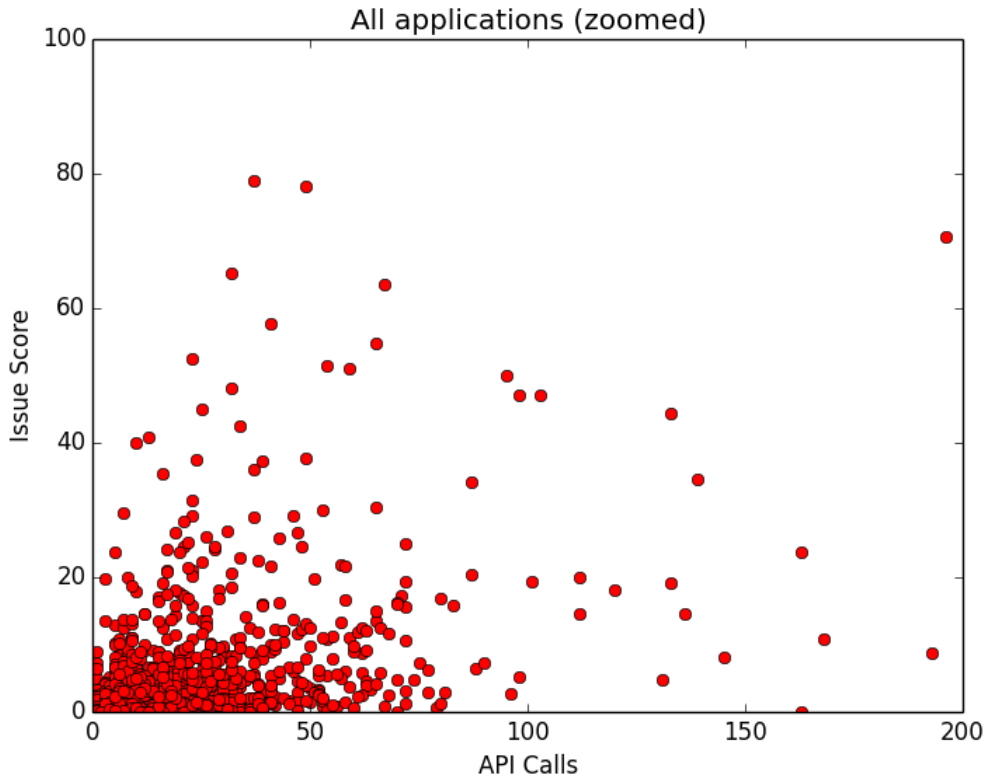
```
commit 08a3106f1172208daa93078de3e5a35bf59a69fc
Author: Martin Fietz <Martin.Fietz@gmail.com>
Date:   Sun Nov 1 23:24:55 2015 +0100

    Check for null

diff --git a/core/src/main/java/de/danoeh/antennapod/core/service/playback/PlaybackServiceTaskManager.java b/core/src/main/java/de/d
index 19ef6ef..9c3ff44 100644
--- a/core/src/main/java/de/danoeh/antennapod/core/service/playback/PlaybackServiceTaskManager.java
+++ b/core/src/main/java/de/danoeh/antennapod/core/service/playback/PlaybackServiceTaskManager.java
@@ -366,10 +366,14 @@ public class PlaybackServiceTaskManager {
                     }
                     if (waitingTime <= 0) {
                         Log.d(TAG, "Sleep timer expired");
-                        shakeListener.pause();
-                        shakeListener = null;
+                        if(shakeListener != null) {
+                            shakeListener.pause();
+                            shakeListener = null;
+                        }
                         if (!Thread.currentThread().isInterrupted()) {
                             callback.onSleepTimerExpired();
+                        } else {
+                            Log.d(TAG, "Sleep timer interrupted");
                         }
                     }
                 } catch (InterruptedException e) {
```

Résultats : csv

	A	B	C	D	E
1	Application	class	Number_Api_Calls	Class_Size	Issue_Score
491	Conversations	XmppConnectionService	196	164	70.66997378
492	Conversations	TagWriter	11	9	0.048017533
493	Conversations	IqPacket	2	7	2.763807114
494	Conversations	AboutPreference	4	3	0
495	Conversations	UIHelper	23	15	52.40479752
496	AntennaPod	DirectoryChooserActivity	50	19	3.556309097
497	AntennaPod	FeedPreferences	4	4	1.175008991
498	AntennaPod	ApplicationCallbacks	2	3	0
499	AntennaPod	VariableSpeedDialog	11	2	0.46714607
500	AntennaPod	GpodnetEpisodeAction	15	13	2.259987153
501	AntennaPod	ItemDescriptionFragment	43	26	5.1040353
502	AntennaPod	HandlerState	6	4	0

Résultats : graphique

Pour l'ensemble des applications : 1452 points



Analyse : normalité

H_0 : Les distributions des variable sont normales

H_1 : Les distribution ne suivent pas une loi normal

Test de Kolmogorov–Smirnov:

- Appels à l'API : p-value = 0
- Taille de la classe : p-value = 0
- Défauts: p-value = 0

On rejette l'hypothèse nulle.

➢ Les distributions ne sont pas normal.

Analyse : corrélation

H_0 : Il n'y a pas de corrélation entre les deux variables.

H_1 : Il y a une corrélation entre les deux variables.

Test de rho de Spearman et tau de Kendall (corrélation de rang)

```
--- Correlation : API Call <> Issue
        Spearman rho correlation coefficient = 0.527401842979
        Spearman p-value = 4.86887566494e-105
        Kendall Tau = 0.370574255364
        Kendall p-value = 1.31735681205e-99
--- Correlation : Class Size <> Issue
        Spearman rho correlation coefficient = 0.552109544494
        Spearman p-value = 3.67737549189e-119
        Kendall Tau = 0.395892647999
        Kendall p-value = 1.48347036067e-115
```

- On rejette l'hypothèse nulle: Il y a une corrélation.
- Force moyenne quand l'un augmente, l'autre à tendance à augmenter.

Analyse : différence

H_0 : Echantillon de la même population avec la même moyenne.

Comparaison des défauts pour les classes avec appels et sans appels.

On a deux ensemble non conçu par paire.

```
Kruskal-wallis F-value = 708.49570002
Kruskal-wallis p-value = 4.24887675266e-156
```

Comparaison des appels pour les classes avec défauts et sans défauts.

```
Kruskal-wallis F-value = 713.701078369
Kruskal-wallis p-value = 3.13585129284e-157
```

- On rejette l'hypothèse nulle. Les distributions ont une moyenne différente. Une F-value élevé ajoute comme information qu'il y a des valeurs élevés dans un groupe et pas dans l'autre.

Analyse : modèle prédictif

Idée : regarder si il possible de créer un modèle suivant :

- Si je connais le nombre d'appel et la taille de la classe, est ce qu'il serait possible d'estimer le nombre de défauts ?

Dans ce cas ci, on va réaliser une régression linéaire par la méthode des moindres carrés.

Analyse : Régression linéaire (1/3)

OLS Regression Results

Dep. Variable:	issues	R-squared:	0.195
Model:	OLS	Adj. R-squared:	0.194
Method:	Least Squares	F-statistic:	351.0
Date:	Sat, 19 Dec 2015	Prob (F-statistic):	2.62e-70
Time:	15:22:12	Log-Likelihood:	-5569.1
No. Observations:	1452	AIC:	1.114e+04
Df Residuals:	1450	BIC:	1.115e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
APIcalls	0.2176	0.012	18.735	0.000	0.195	0.240
const	0.6381	0.371	1.721	0.085	-0.089	1.365

Omnibus:	2032.751	Durbin-Watson:	1.942
Prob(Omnibus):	0.000	Jarque-Bera (JB):	832433.673
Skew:	7.672	Prob(JB):	0.00
Kurtosis:	119.292	Cond. No.	40.2

Analyse : Régression linéaire (2/3)

OLS Regression Results

Dep. Variable:	issues	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	2.644e+32
Date:	Sat, 19 Dec 2015	Prob (F-statistic):	0.00
Time:	15:22:12	Log-Likelihood:	43188.
No. Observations:	1452	AIC:	-8.637e+04
Df Residuals:	1450	BIC:	-8.636e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
ClassSize	1.0000	6.15e-17	1.63e+16	0.000	1.000	1.000
const	2.887e-15	8.24e-16	3.502	0.000	1.27e-15	4.5e-15

Omnibus:	2245.376	Durbin-Watson:	1.827
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1099717.849
Skew:	9.345	Prob(JB):	0.00
Kurtosis:	136.521	Cond. No.	14.4

Analyse : Régression linéaire (3/3)

OLS Regression Results

Dep. Variable:	issues	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	4.790e+32
Date:	Sat, 19 Dec 2015	Prob (F-statistic):	0.00
Time:	15:22:12	Log-Likelihood:	44123.
No. Observations:	1452	AIC:	-8.824e+04
Df Residuals:	1449	BIC:	-8.822e+04
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
APIcalls	-2.142e-16	1.78e-17	-12.070	0.000	-2.49e-16	-1.79e-16
ClassSize	1.0000	3.6e-17	2.78e+16	0.000	1.000	1.000
const	-2.22e-16	5.09e-16	-0.436	0.663	-1.22e-15	7.76e-16

Omnibus:	2190.644	Durbin-Watson:	1.912
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1032412.698
Skew:	-8.888	Prob(JB):	0.00
Kurtosis:	132.417	Cond. No.	41.5

Limites

La fiabilité du système de comptage des défauts.

La taille de la classe est calculé en fonction du nombre de méthode.

Les renommage de classes dans l'historique ne sont pas pris en compte.

L'héritage n'est pas considéré.

Il semblerait avoir des différence de forces statistiques entre les résultats des outils montrés ci-dessus (*numpy* + *scipy*+ *statsmodels*) et les résultats que donne *IBM SSPS Statistics.*

- *Exemple :* Coefficient de rho de spearman pour Class size <> Issues
- *Scipy :* = 0.552109544494
- *IBM SSPS Statistics:* = 0.544

…

Conclusion

Les données montre une corrélation entre les appels à l'API Android et les défauts dans le code. Elles montrent aussi qu'il des différences en terme de défauts entre les bouts de codes avec appels et sans appels.

Cependant, la régression linéaire montre qu'il n'est pas possible d'estimer correctement le nombre de défauts à partir de uniquement le nombre d'appels, avec les données récoltés et la méthode présenté ci-dessus. Si on y ajoute la taille, l'estimation semble correct mais le nombre d'appel ne rentre presque pas en compte.