Méthode d'évaluation de la complexité de logiciels Open Source (FLOSC)

Version 1.1 - 18/01/2013

Table des matières

1 Note de licence

2 Historique des modifications

Version	Date	Auteurs	Commentaires
1.0	17/09/2012	Raphaël Semeteys	Conception et rédaction initiales.
1.1	18/01/2013	Raphaël Semeteys	Passage au format Markdown, corrections.

3 Introduction

L'évaluation de la complexité d'un logiciel est une problématique connue de l'ingénierie du développement. Plusieurs méthodes ont été proposées en allant du simple comptage du nombre de lignes de code à des démarche plus élaborées comme la méthode des points de fonction.

Nous proposons une méthode intermédiaire entre ces deux extrêmes pour pouvoir disposer d'une démarche simple, adaptée à la nature des logiciels libres ou open source et pouvant être outillée.

La méthode d'analyse en Points de Fonction (PF) mesure la taille d'une application en quantifiant les fonctionnalités offertes aux utilisateurs ou aux autres applications ainsi que les données manipulées. Cette mesure est réalisée en se basant sur les spécifications fonctionnelles et la modélisation logique des données.

Les concepts de cette méthode sont basés sur le point de fonction qui est une unité de mesure, permettant d'exprimer la taille fonctionnelle d'un système d'information du point de vue des utilisateurs métiers. Le Point de Fonction est une métrique informatique de référence. Elle permet de déterminer la taille fonctionnelle d'une application, basée sur les fonctionnalités perçues par les utilisateurs, et indépendamment des techniques mises en œuvre pour la réalisation et l'exploitation du système.

Le standard de l'IFPUG [1] (*International Function Point Users Group*), normalisé à l'ISO, a notamment été retenu par Atos pour l'analyse en Points de Fonction de ses projets.

4 La méthode FLOSC

La méthode que nous proposons d'utiliser pour déterminer la complexité des logiciels se base sur la méthode reconnue IFPUG. Il nous a cependant semblé pertinent de l'adapter au contexte des logiciels libres et open source.

En effet, la méthode IFPUG propose d'évaluer la complexité des applications dans les phases amont des projets (spécification/conception) en se concentrant sur les aspects métiers sans se soucier des aspects techniques d'implémentation. Ainsi, nous proposons les adaptations suivantes :

- estimation du nombre de Points de Fonction des logiciels libres et open source à partir de nombre de lignes de codes en se basant sur des métriques fines issues de l'observation par Atos et l'industrie informatique en général ;
- utilisation de facteurs d'ajustement différents de ceux proposés par IFPUG mais s'inscrivant dans la logique proposée par cette méthode.

La démarche générale d'évaluation peut ainsi être décomposée en trois étapes, décrites dans la suite du présent document.

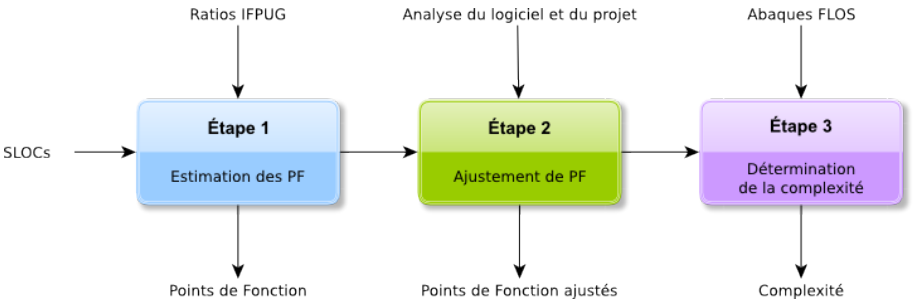

FIGURE 1 – Démarche générale de FLOSC

4.1 Étape 1 : estimation des Points de Fonction

4.1.1 Référentiel IFPUG d'Atos

Depuis 2007, Atos a généralisé au niveau du groupe l'utilisation de la méthode des Points de Fonction et plus particulierment d'IFPUG. Nous disposons ainsi d'un référentiel conséquent de projets évalués selon cette méthode et disposons

1. http://www.ifpug.org

donc de métriques utilisables pour estimer le nombre de Points de Fonction de nouveaux projets.

La métrique que nous proposons d'utiliser est la correspondance entre le nombre de lignes de code source (*SLOC* en anglais) et de nombre de Points de Fonction. En effet, les projets estimés en nombre de Points de Fonction sont ensuite, une fois développés, mesurés en nombre SLOC afin de tenir à jour le référentiel IFPUG d'Atos.

4.1.2 Nombre de SLOC

La notion de nombre de lignes de code source (SLOC) utilisée est celle définie et détaillée par Robert Park [2] du Software Engineering Institute et dont le comptage est implémenté par de nombreux outils. Nous utilisons l'implémentation libre que nous considérons comme la plus maintenue et supportant le plus de langages : cloc [3].

4.1.3 Ratios PF/SLOC par langages de programmation

Le référentiel différencie les ratios PF/SLOC en fonction des langages de programmation utilisés pour développer les logiciels. En effet, certains langages étant plus verbeux que d'autres, il convient de prendre ce paramètre en ligne de compte.

En outre les ratios utilisés proviennent des médianes des valeurs mesurées par langage. Le tableau ci-dessus présente ces ratios pour les langages les plus connus.

Langage	Ratio PF/SLOC médian
C	107
C++	53
HTML	42
Java	53
JavaScript	55
Perl	57
PHP	56
Python	57
SQL	30

2. http://www.sei.cmu.edu/library/abstracts/reports/92tr020.cfm
3. http://cloc.sourceforge.net

La liste détaillée des ratios pour les différents langages reconnus par cloc est accessible en annexe du présent document.

4.2 Étape 2 : ajustement des Points de Fonction

Les Points de Fonction ainsi calculés sont ajustés selon les critères suivants :

- niveau d'industrialisation du projet en charge du développement du logiciel (critère C1) ;
- niveau d'organisation et d'architecture du code source (critère C2) ;
- dépendances du logiciel en termes de librairies externes et de greffons (critère C3).

Ces critères sont évalués avec des notes entières entre 0 et 2, dont les significations sont listées ci-après.

Critère C1 : niveau d'industrialisation du projet :

- 0 : pas d'utilisation d'outils pour gérer le développement (bugtracker, forums...), code source difficile à trouver, pas de roadmap ;
- 1 : seuls quelques processus sont exposés, outillés et utilisés ;
- 2 : processus de développement, de demandes d'évolution, des tests, d'intégration continue... mis en œuvre, documentés et respectés.

Critère C2 : niveau d'organisation et d'architecture du code :

- 0 : code monolithique, langage non objet, pas d'organisation du code en couches ou en modules ;
- 1 : code faiblement architecturé mais proposant quelques principes de factorisation du code, sans que cela soit généralisé à l'ensemble de ce dernier ;
- 2 : code très modulaire, orienté objet avec héritage et utilisation d'interfaces, organisé en modules correspondant à des couches fonctionnelles différentes.

Critère C3 : dépendances du logiciels en termes de librairies externes et de greffons :

- 0 : logiciel embarquant de nombreuses librairies externes ou de nombreux greffons ;
- 1 : logiciel embarquant quelques librairies externes ou pouvant être étendu via quelques greffons ;
- 2 : logiciel n'embarquant aucune librairie externe et n'étant pas conçu pour être étendu via des greffons.

L'analyse en Points de Fonction IFPUG décrit la manière dont sont utilisés les paramètres d'ajustement pour modifier le nombre de PF, via la formule suivante :

$$PF_{ajustés} = PF_{non-ajustés} * (0,65 + \frac{\sum paramètres}{100})$$

Sachant que le nombre de paramètres proposés par IFPUG est de 14 et qu'ils sont notés de 0 à 5, ces derniers peuvent donc faire varier le nombre de PF de 65% à 135%.

Dans la version adaptée que nous proposons, nous n'utilisons que trois paramètres, pour rester dans le même ordre de variabilité que la méthode IFPUG standard nous utilisons donc la formule suivante :

$PF_{ajustés} = PF_{non-ajustés} * (0,65 + \frac{(2-C1)+(2-C2)+(2-C3)}{20})$

4.3 Étape 3 : détermination de la complexité

Sur la base de métriques obtenues lors de l'utilisation de cette démarche basée sur IFPUG, les abaques suivantes sont utilisées pour déterminer le niveau de complexité d'un logiciel open source.

Nombre de PF ajustés	Niveau de complexité
Moins de 1000	Simple
Entre 1000 et 10 000	Complexe
Plus de 10 000	Très complexe

5 Outils

5.1 Calcul du nombre de SLOC

Comme précédemment évoqué, FLOSC se base sur l'outil libre cloc [4] pour compter le nombre de SLOC d'un composant open source.

5.2 Ajustement des Points de Fonction

Le projet FLOSC développe et maintient un application Web dédiée à la saisie et à la visualisation des ajustements des Points de Fonction.

4. http://cloc.sourceforge.net

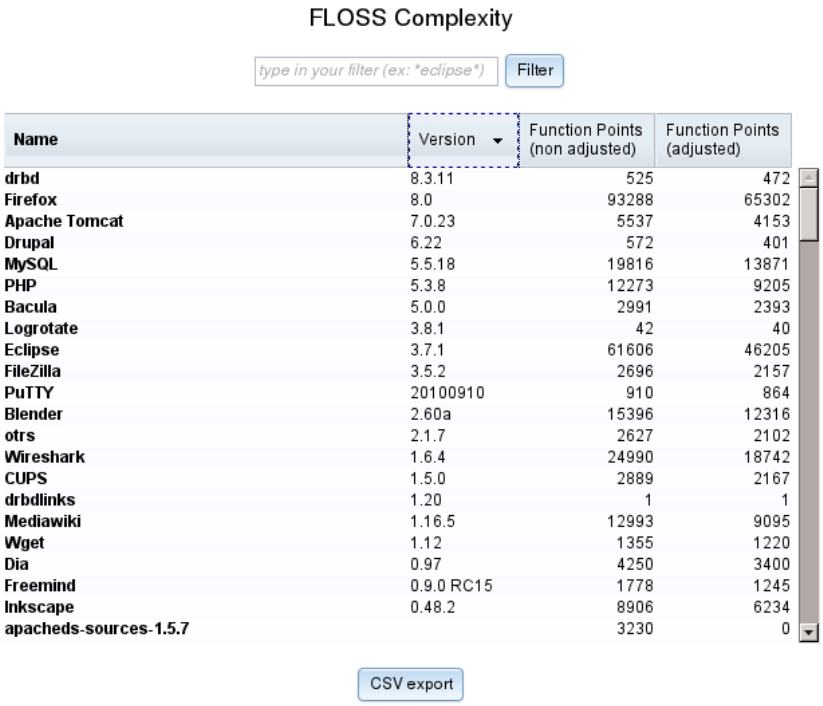

FIGURE 2 – Application WebFLOSC (menu)

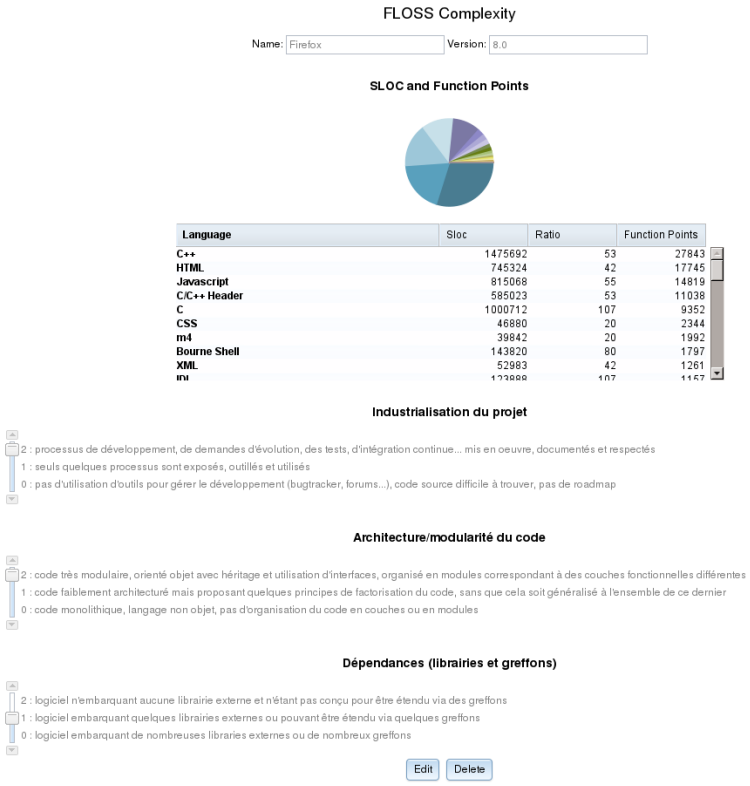

FIGURE 3 – Application WebFLOSC (détail)

6 Annexe A : ratios PF/SLOC par langages reconnus par cloc

Langage	Ratio	Description
ABAP	18	abap
ActionScript	54	as
Ada	154	ada, adb, ads, pad
ADSO/IDSM	18	adso
AMPLE	20	ample, dofile, startup
ASP	50	asa, asp
ASP.Net	50	asax, ascx, asmx, aspx, config, master, sitemap, webinfo
Assembly	203	asm, S, s
awk	20	awk
Bourne Again Shell	80	bash
Bourne Shell	80	sh
C	107	c, ec, pgc
C Shell	80	csh, tcsh
C#	59	cs
C++	53	C, cc, cpp, cxx, pcc
C/C++ Header	53	H, h, hh, hpp
CCS	20	ccs
Cmake	20	CMakeLists.txt
COBOL	78	cbl, CBL, cob, COB
ColdFusion	56	cfm
CSS	20	css
Cython	56	pyx
D	53	d
DAL	20	da
DOS Batch	80	bat, BAT

Langage	Ratio	Description
DTD	42	dtd
Erlang	20	erl, hrl
Expect	20	exp
Focus	45	focexec
Fortran 77	118	F, f, f77, F77, pfo
Fortran 90	118	F90, f90
Fortran 95	118	F95, f95
Go	55	go
Groovy	54	groovy
Haskell	20	hs, lhs
HTML	1000	htm, html
IDL	107	idl, pro
Java	53	java
Javascript	55	js
JCL	59	jcl
JSP	59	jsp
Kermit	20	ksc
Korn Shell	80	ksh
lex	20	l
Lisp	80	cl, el, jl, lisp, lsp, sc, scm
LiveLink Oscript	54	oscript
Lua	54	lua
m4	20	ac, m4
make	20	am, gnumakefile, Gnumakefile, Makefile, makefile
MATLAB	20	m
Modula3	80	i3, ig, m3, mg
MSBuild scripts	20	csproj, wdproj
MUMPS	20	mps, m
MXML	42	mxml

Langage	Ratio	Description
NAnt scripts	20	build
NASTRAN DMAP	35	dmap
Objective C	107	m
Objective C++	53	mm
Ocaml	55	ml
Oracle Forms	29	fmt
Oracle Reports	29	rex
Pascal	80	dpr, p, pas, pp
Patran Command Language	55	pcl, ses
Perl	57	perl, PL, pl, plh, plx, pm
PHP	53	php, php3, php4, php5
PHP/Pascal	53	inc
Python	56	py
Rexx	50	rexx
Ruby	54	rb
Ruby HTML	54	rhtml
Scala	53	scala
sed	20	sed
SKILL	80	il
SKILL++	80	ils
Smarty	42	smarty, tpl
Softbridge Basic	52	sbl, SBL
SQL	30	psql, SQL, sql
SQL Data	30	data.sql
SQL Stored Procedure	30	spc.sql, spoc.sql, sproc.sql, udf.sql
Tcl/Tk	54	itk, tcl, tk
Teamcenter def	20	def
Teamcenter met	20	met
Teamcenter mth	20	mth

Langage	Ratio	Description
VHDL	107	vhd, VHD, VHDL, vhdl
vim script	20	vim
Visual Basic	52	bas, cls, frm, vb, VB, vba, VBA, vbs, VBS
XAML	42	xaml
XML	1000	XML, xml
XSD	42	xsd, XSD
XSLT	20	xsl, XSL, xslt, XSLT
yacc	20	y
YAML	42	yaml, yml

7 Annexe B : framework Drakkr

FLOSC est un sous-projet de l'initiative Drakkr visant à construire un framework libre dédié la gouvernance open source au sein des entreprises et administrations.

Outre FLOSC, lié à l'évaluation de la complexité des logiciels libres et open source, Drakkr propose également d'autres méthodes et outils pour mettre en oeuvre une telle gouvernance.

FIGURE 4 – Framework Drakkr

– **OSC** (Open Source Cartouche) : sous-projet dédié à l'identification unique

d'une version d'un logiciel open source ainsi qu'à la gestion des ses metadonnées ;
- **ECOS** (Evaluation des Coûts liés à l'adoption de logciels Open Source) : sous-projet relatif à l'évaluation et au calcul du coût total de prossession d'un logiciel open source ainsi qu'au retour sur investissement d'une migration ;
- **QSOS** (Qualification et Sélection de logiciels Open Source) : sous-projet proposant une méthode et des outils pour qualifier, sélectionner et comparer les logiciels open source et ainsi industrialiser et mutualiser une démarche de veille ;
- **SLIC** (Software LIcense Comparator) : sous-projet dédié à la description formelle des licences open source et de leurs compatibilités respectives ;
- **SecureIT** : sous-projet dédié à la gestion des alertes de sécurité dans les logiciels open source.

Consultez le site Web du projet Drakkr pour plus de détails : http://www.drakkr.org.