TADYSZAK Lucas BOMPARD Jules

Licence 3 Info

Projet Génie logiciel

Rapport WebMagic

Table des matières

[I. Présentation globale du projet 3](#_Toc66109928)

[1) Utilité du projet 3](#_Toc66109929)

[2) Configuration requise 3](#_Toc66109930)

[3) Quelques commandes Maven 4](#_Toc66109931)

[II. Description du projet 4](#_Toc66109932)

[III. Historique du Git 4](#_Toc66109933)

[1) Composition de l’équipe 4](#_Toc66109934)

[2) Activité sur le projet 5](#_Toc66109935)

[3) Utilisation des branches 5](#_Toc66109936)

[4) Utilisation des pull requests 5](#_Toc66109937)

[IV. Architecture logicielle 6](#_Toc66109938)

[1) Utilisation de bibliothèques extérieures 6](#_Toc66109939)

[2) Organisation en paquetages 6](#_Toc66109940)

[3) Répartition des classes dans les paquetages 9](#_Toc66109941)

[4) Organisations des classes 10](#_Toc66109942)

[V. Analyse approfondie 12](#_Toc66109943)

[1) Tests 12](#_Toc66109944)

[2) Commentaires 14](#_Toc66109945)

[3) Dépréciation 14](#_Toc66109946)

[4) Duplication de code 15](#_Toc66109947)

[5) God Classes 16](#_Toc66109948)

[6) Analyse des méthodes 17](#_Toc66109949)

[VI. Nettoyage de Code et Code smells 18](#_Toc66109950)

[1) Règles de nommage 18](#_Toc66109951)

[2) Nombre magique 18](#_Toc66109952)

[3) Structure du code 18](#_Toc66109953)

[4) Code mort 18](#_Toc66109954)

# Présentation globale du projet

## Utilité du projet

WebMagic est un framework crawler évolutif pour Java. Il couvre l'ensemble du cycle de vie du crawler : téléchargement, gestion des url, extraction de contenu et persistant. Il permet de simplifier le développement d'un crawler pour un site quelconque.

Un crawler est un logiciel qui a pour principale mission d'explorer le Web afin d'analyser le contenu des documents visités et les stocker de manière organisée dans un index. Les crawler parcourent en permanence, de manière autonome et automatique divers sites Web et pages Internet pour trouver de nouveaux contenus ou d'éventuelles mises à jour de contenu qui ont été explorées dans le passé. Derrière l'activité se cache une autre tâche : la fonction d'indexation des pages Web en fonction de la qualité du contenu (évalué selon les normes configurées en amont), aidant ainsi les moteurs de recherche à classer les pages Internet dans les résultats affichés.

## Configuration requise

Ajouter les dépendances écrites ci-dessous dans le fichier pom.xml.

WebMagic utilise slf4j avec l'implémentation de slf4j-log4j12. Si vous avez personnalisé votre implémentation slf4j, il faut exclure slf4j-log4j12. Simple Logging Facade for Java fournit une API de journalisation Java au moyen du design pattern Facade.

<dependency>

<groupId>us.codecraft</groupId>

<artifactId>webmagic-core</artifactId>

<version>0.7.4</version>

</dependency>

<dependency>

<groupId>us.codecraft</groupId>

<artifactId>webmagic-extension</artifactId>

<version>0.7.4</version>

<exclusions>

<exclusion>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</exclusion>

</exclusions>

</dependency>

## Quelques commandes Maven

WebMagic est un projet purement Java, donc il sera très simple de télécharger et compiler le code source avec Maven. Vous trouverez ci-dessous la liste des commandes Maven les plus courantes à utiliser.

mvn validate : valide si le projet est correct et que toutes les informations nécessaires sont disponibles.

mvn compile : compile le code source du projet.

mvn test : test le code source compilé en utilisant un cadre de test unitaire approprié.

mvn package : prend le code compilé et le conditionne dans son format distribuable, comme un JAR.

mvn integration-test : traite et déploie le paquet si nécessaire dans un environnement où des tests d'intégration peuvent être effectués.

mvn verify : effectue tout contrôle pour vérifier que le paquet est valable et répond aux critères de qualité.

mvn install : installe le paquet dans le dépôt local, pour l'utiliser comme dépendance dans d'autres projets locaux.

mvn deploy : effectué dans un environnement d'intégration ou de publication, il copie le paquet final dans le dépôt distant pour le partager avec d'autres développeurs et projets.

mvn clean : nettoyer les artefacts créés par des constructions antérieures.

mvn site : génère la documentation du site pour le projet.

# Description du projet

Le projet contient un Readme qui explique brièvement l’outil WebMagic. On n’apprend pas dans le Readme comment utiliser le projet WebMagic avec les commandes pour compiler et exécuter ce dernier. Mais on sait à l’aide d’un badge que la compilation de la branche Master du projet a échoué. Ce projet est découpé en plusieurs parties peu détaillées.

1. Introduction : description du projet en 2 lignes ;
2. Caractéristiques : énumération des différentes fonctionnalités disponibles ;
3. Installation : dépendances nécessaires ;
4. Exemples : morceaux de codes qui montrent comment écrire un crawler ;
5. Documentation et autres : un lien qui redirige vers le site pour consulter la documentation plus détaillée, ainsi que des informations sur le projet WebMagic.

Le projet possède une documentation accessible via le site [WebMagic](http://webmagic.io/docs/) qui explique à quoi sert ce projet dans l’introduction de la documentation, comment installer le projet, récupérer le code source, compiler ce code source avec Maven ainsi que des exemples de crawler. La documentation n’est pas très détaillée, il faut avoir des connaissances en Java et Maven pour comprendre la documentation mais aussi mettre en place le projet. Cette documentation est incomplète, il manque des informations pour compiler et exécuter le projet sans avoir à faire des recherches sur d’autres sites.

# Historique du Git

## Composition de l’équipe

Le dépôt Github est composé de 40 contributeurs. On constate le créateur du projet a beaucoup contribué au début avec plus de 800 commits, les autres font très peu de commits (entre 1 et 5).

## Activité sur le projet

Le projet a une grosse activité du 21 avril 2013 au 21 juin 2014 puis ralentit jusqu’au 1er janvier 2018. Le projet ne percevra aucune modification pendant une année et demie. Du 23 juillet 2019 à aujourd’hui, le projet a connu de nouvelles modifications avec de nouveaux contributeurs mais ces dernières sont mineures et on peut dire qu’aujourd’hui le projet est stable.

## Utilisation des branches

Une branche est utilisée pour maintenir une version particulière du projet dans un but précis et fixé à l’avance. Elles peuvent servir à développer une nouvelle fonctionnalité sans polluer la branche supérieure avec des modifications non validées ou à garder une branche correspondant à une version stable définie.

Le dépôt du créateur code4craft contient une branche par défaut qui se nomme develop, deux branches actives master et revert-977-build et 20 autres branches inactives.

## Utilisation des pull requests

Un pull request est une méthode qui permet de soumettre des contributions à un projet utilisant un système de contrôle de version distribué comme Git. Un pull request se produit lorsqu'un développeur demande que les modifications apportées à un dépôt externe soient prises en compte pour être incluses dans le dépôt principal d'un projet.

Le projet utilise le mécanisme des pull requests avec un total de 144 pull requests dont 121 sont clos et 23 sont encore en attentes. Cette information montre une importante contribution active sur le projet WebMagic.

# Architecture logicielle

## Utilisation de bibliothèques extérieures

Nous avons vu dans la présentation du projet WebMagic que ce dernier utilisé Maven. Les bibliothèques externes utilisées par Maven sont indiquées dans le fichier pom.xml dans les balises dependencies. On comptabilise 25 bibliothèques externes dans Maven.

## Organisation en paquetages

• compter le nombre de paquetages.

Webmagic-core : 20 paquetages

Webmagic-extension : 21 paquetages

Webmagic-samples : 10 paquetages

Webmagic-saxon : 2 paquetages

Webmagic-scripts : 2 paquetages

Webmagic-selenium : 4 paquetages

Total : 59 paquetages

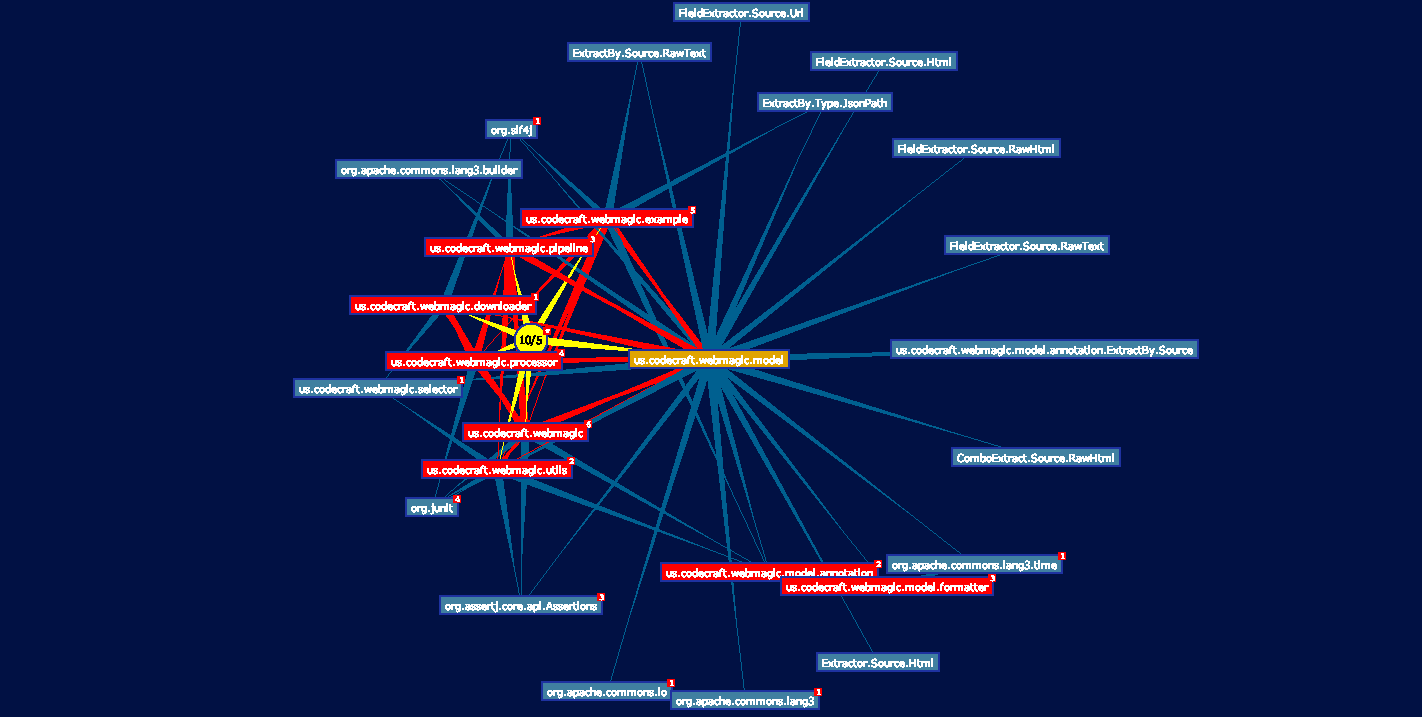
• analyser les liens entre les paquetages : quels paquetages dépendent de quels autres ? les paquetages sont-ils organisés en couche ? Existe-t-il des cycles entre paquetages ?

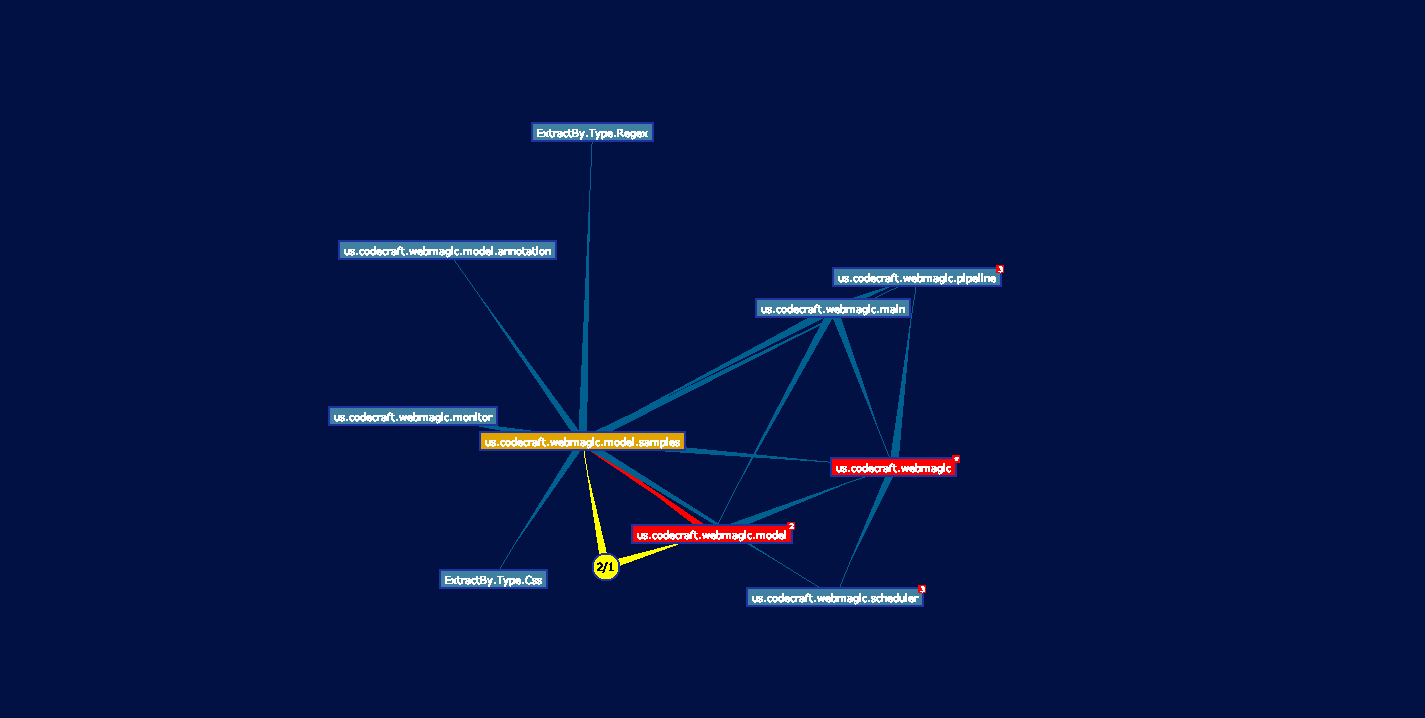
Le graphe de dépendance des paquets est analysé pour les composants forts (tous les nœuds accessibles les uns des autres, appelés "enchevêtrement") et ces nœuds sont colorés en rouge. Les autres paquets qui ne participent pas aux cycles sont colorés en bleu. Le petit cercle jaune relie tous les paquets impliqués dans l'enchevêtrement. Il indique le nombre de nœuds impliqués dans l'enchevêtrement.

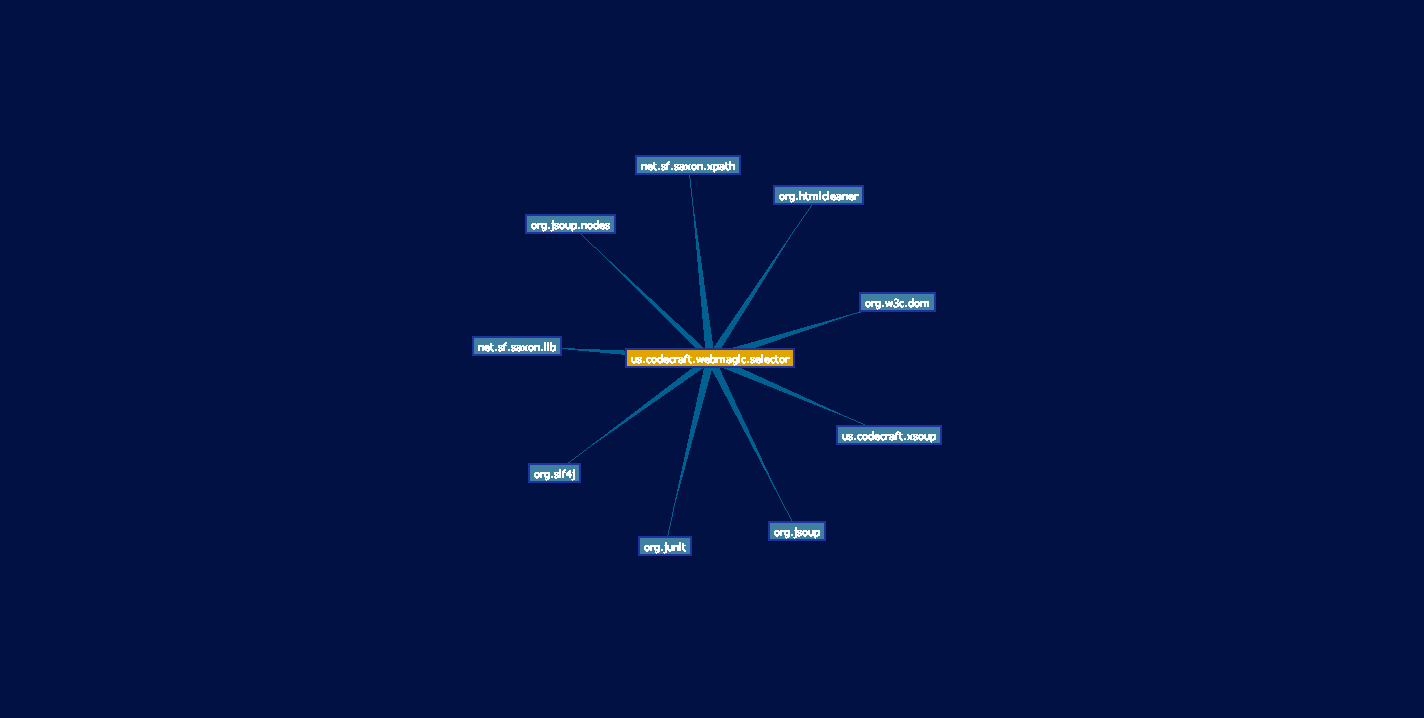
Webmagic-core :

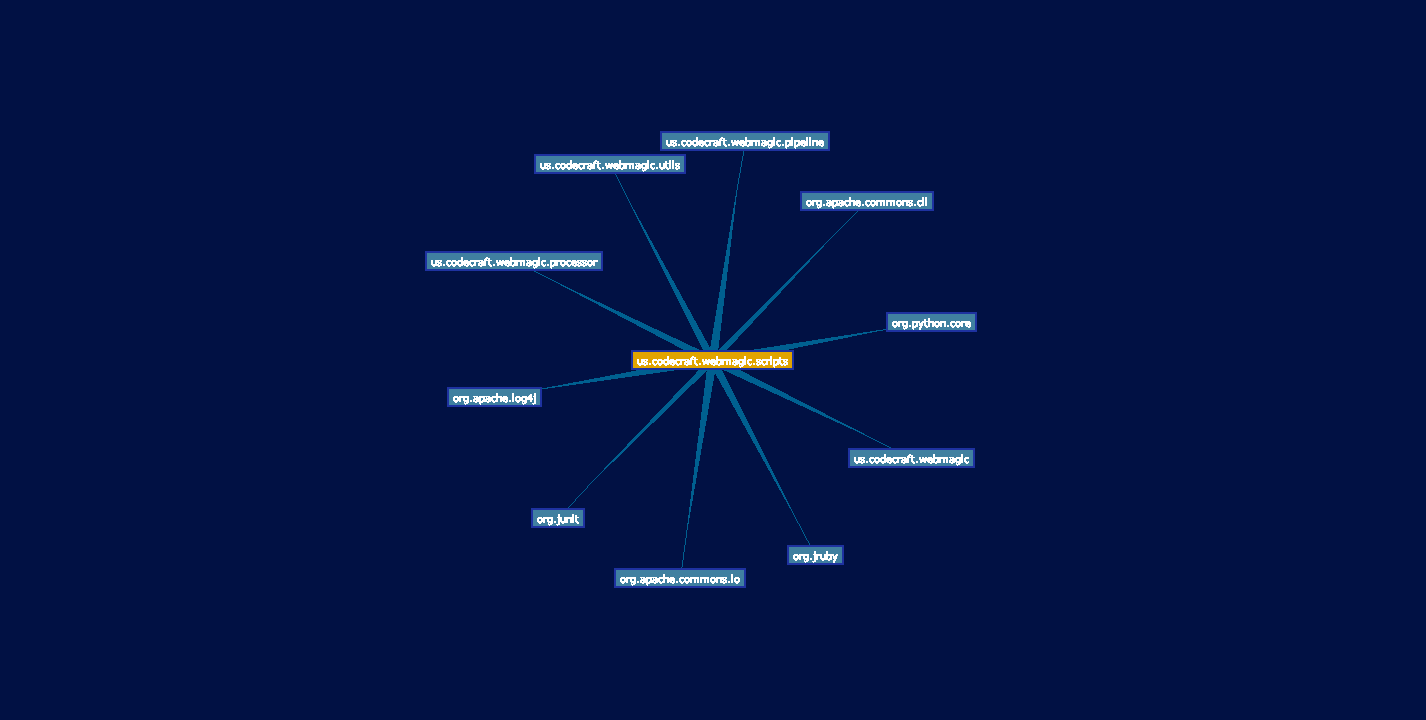
Une image contenant texte

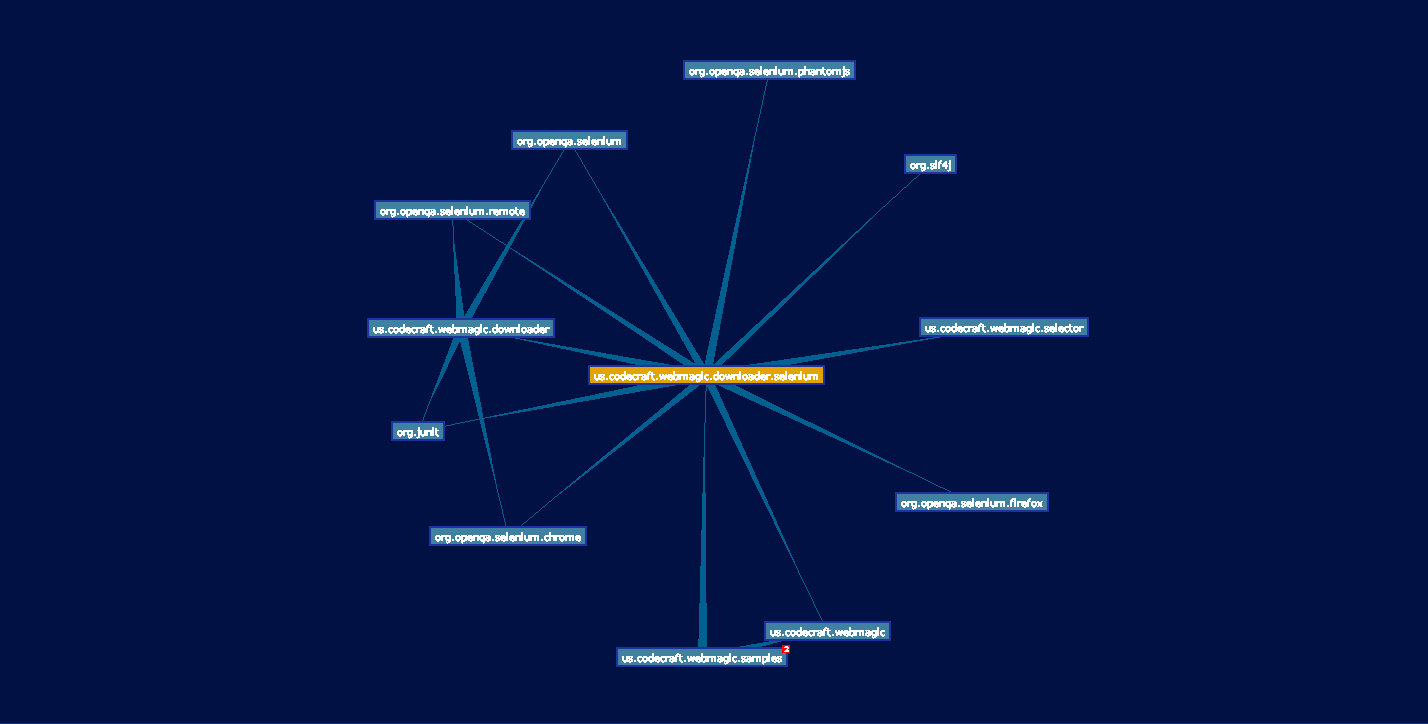
Description générée automatiquement

Webmagic-extension : 

Webmagic-samples : 

Webmagic-saxon : 

Webmagic-scripts : 

Webmagic-selenium :

• analyser la hiérarchie de paquetage :

Quel est le nombre de niveaux de paquetages ?

Niveaux de packages max : 3

La hiérarchie de paquetages pour les tests suit-elle la hiérarchie de paquetages des sources ?

Oui même architecture parallèle entre les sources java et les tests.

Existe-t-il des paquetages qui ne contiennent qu’un seul paquetage sans aucune classe ?   
Non.

## Répartition des classes dans les paquetages

• compter le nombre de classes par paquetage : minimum, maximum et moyenne. Également, on pourra fournir le nombre total de classes.

Webmagic-core :   
us.codecraft.webmagic : min = 1 ; max = 20 ; moyenne = 6 ; total = 72

Webmagic- extension :

us.codecraft.webmagic : min = 1 ; max = 20 ; moyenne = 4 ; total = 38

Webmagic-sample :

us.codecraft.webmagic : min = 1 ; max = 20 ; moyenne = 6 ; total = 39

Webmagic-saxon :

us.codecraft.webmagic.selector : min = 2 ; max = 2 ; moyenne = 2 ; total = 2

Webmagic-scipts :

us.codecraft.webmagic.scripts : min = 6 ; max = 6 ; moyenne = 6 ; total = 6

Webmagic-selenium :

us.codecraft.webmagic.downloader.selenium : min = 2 ; max = 2 ; moyenne = 2 ; total = 2

• analyser la répartition des classes dans les différents paquetages :

Toutes les classes ou presque sont-elles dans le même paquetage ?

La répartition des classes dans les différents paquetages est homogène à l’exception d’un paquetage qui regroupe plus de classes que les autres.

Est-ce que des paquetages non-feuilles contiennent des classes ?   
Oui (Webmagic-core et Webmagic-parent)

S’il y a plusieurs hiérarchies parallèles, les paquetages qui ont le plus de classes dans une hiérarchie ont-ils aussi le plus de classes dans les autres ?

C’est le cas uniquement dans Webmagic-core.

## Organisations des classes

* La hiérarchie est-elle plutôt plate (peu de niveaux de hiérarchie) ou à l’inverse profonde ?

On pourra par exemple s’appuyer sur la profondeur de l’arbre d’héritage (DIT), le nombre d’enfants par classes (min, max ou moyenne) (NOC)

Webmagic-core :

Profondeur de l’arbre d’héritage (DIT) : max : 3 ; moyenne : 1

Nombre d’enfants par classes (NOT) : min = 1 ; max = 7 ; moyenne = 0.5 ; total = 46

Webmagic-extension :

Profondeur de l’arbre d’héritage (DIT) : min : 1 ; max : 3 ; moyenne : 1.2

Nombre d’enfants par classes NOT) : min = 1 ; max = 8 ; moyenne = 0.4 ; total = 34

Webmagic-sample :

Profondeur de l’arbre d’héritage (DIT) : min : 1 ; max : 3 ; moyenne : 1,1

Nombre d’enfants par classes NOT) : min = 1 ; max = 1 ; moyenne = 1 ; total = 1

Webmagic-saxon :

Profondeur de l’arbre d’héritage (DIT) : min : 1 ; max : 2 ; moyenne : 1.2

Nombre d’enfants par classes NOT) : min = 0 ; max = 0 ; moyenne = 0 ; total = 0

Webmagic-scipts :

Profondeur de l’arbre d’héritage (DIT) : min : 1 ; max : 2 ; moyenne : 1.5

Nombre d’enfants par classes NOT) : min = 0 ; max = 0 ; moyenne = 0 ; total = 0

Webmagic-selenium :

Profondeur de l’arbre d’héritage (DIT) : min : 1 ; max : 1 ; moyenne : 1

Nombre d’enfants par classes NOT) : min = 0 ; max = 0 ; moyenne = 0 ; total = 0

• étudier la stabilité des classes en général ou de quelques-unes en particulier. Pour cela, on pourra s’appuyer sur la notion de couplage.

• étudier la cohésion des classes au sein d’un paquetage en particulier.

Deux classes sont couplées, si une modification d'une des deux classes nécessite des modifications apportées dans l'autre. Si le couplage est faible alors la classe est autonome.

On dit qu’une classe a une forte cohésion si tous ses attributs sont utilisés par ses méthodes. Si la cohésion est forte alors les méthodes de la classe coopèrent ensemble pour concevoir une tâche.

D’où l’importance de créer des applications avec une forte cohésion et, un faible couplage pour ainsi gagner en qualité tout en diminuant la complexité.

Nous disposons de quelques métriques simples comme le couplage afférent (Ca) et le couplage efférent (Ce).

Le couplage afférent pour un élément de code est le nombre d’éléments qui l’utilisent. Cette métrique témoigne de la « responsabilité » de cet élément dans l’ensemble du code.  
A l’opposé, le couplage efférent est le nombre d’éléments différents qu’utilise un élément, témoignant de son « indépendance » vis à vis du reste du code.

Webmagic-core :

Ca par package : max = 129 ; min 1 ; moyenne = 23

Ce par package : max = 12 ; min 0 ; moyenne = 4

Webmagic-extension :

Ca par package : max = 26 ; min 0 ; moyenne = 7

Ce par package : max = 11 ; min 1 ; moyenne = 4

Webmagic-sample :

Ca par package : max = 2 ; min 0 ; moyenne = 1

Ce par package : max = 19 ; min 1 ; moyenne = 6

Webmagic-saxon :

Ca par package : max = 0 ; min 0 ; moyenne = 0

Ce par package : max = 1 ; min 1 ; moyenne = 1

Webmagic-scipts :

Ca par package : max = 0 ; min 0 ; moyenne = 0

Ce par package : max = 3 ; min 3; moyenne = 3

Webmagic-selenium :

Ca par package : max = 2 ; min 2 ; moyenne = 2

Ce par package : max = 2 ; min 2 ; moyenne = 2

Une valeur faible indique une classe cohésive et une valeur proche de 1 indique un manque de cohésion et suggère que la classe pourrait être mieux divisée en un certain nombre de (sous-)classes.

Nous ne sommes pas sûr de l'utilité de cette métrique en Java car elle pénalise l'utilisation correcte des getters et des setters en tant que seules méthodes qui accèdent directement à un attribut et les autres méthodes utilisant les méthodes gettter/setter.

Webmagic-core :

Cohésion par package : max = 0.8 ; min = 0 ; moyenne = 0.2

Webmagic-extension :

Cohésion par package : max = 0.8 ; min = 0 ; moyenne = 0.1

Webmagic-sample :

Cohésion par package : max = 0.5 ; min = 0 ; moyenne = 0.2

Webmagic-saxon :

Cohésion par package : max = 0.6 ; min = 0.6 ; moyenne = 0.6

Webmagic-scipts :

Cohésion par package : max = 0.6 ; min = 0.6 ; moyenne = 0.6

Webmagic-selenium :

Cohésion par package : max = 0.8 ; min = 0.8 ; moyenne = 0.8

# Analyse approfondie

## Tests

Nombre de classes de tests :

Webmagic-core : max = 6 ; total = 25 ; moyenne = 3

Webmagic-extension : max = 13 ; total = 77 ; moyenne = 6

Webmagic-samples : max = 1 ; total = 4 ; moyenne = 1

Webmagic-saxon : max = 1 ; total = 1 ; moyenne = 1

Webmagic-scripts : max = 1 ; total = 1 ; moyenne = 1

Webmagic-selenium : max = 2 ; total = 5 ; moyenne = 2

Total : 113

Nombre de méthodes de tests :

Webmagic-core : max = 15 ; total = 58 ; moyenne = 2

Webmagic-extension : max = 10 ; total = 56 ; moyenne = 1

Webmagic-samples : max =4 ; total = 7 ; moyenne = 2

Webmagic-saxon : max = 5 ; total = 9 ; moyenne = 4

Webmagic-scripts : max = 3 ; total = 3 ; moyenne = 3

Webmagic-selenium : max = 2 ; total = 8 ; moyenne = 2

Total : 141

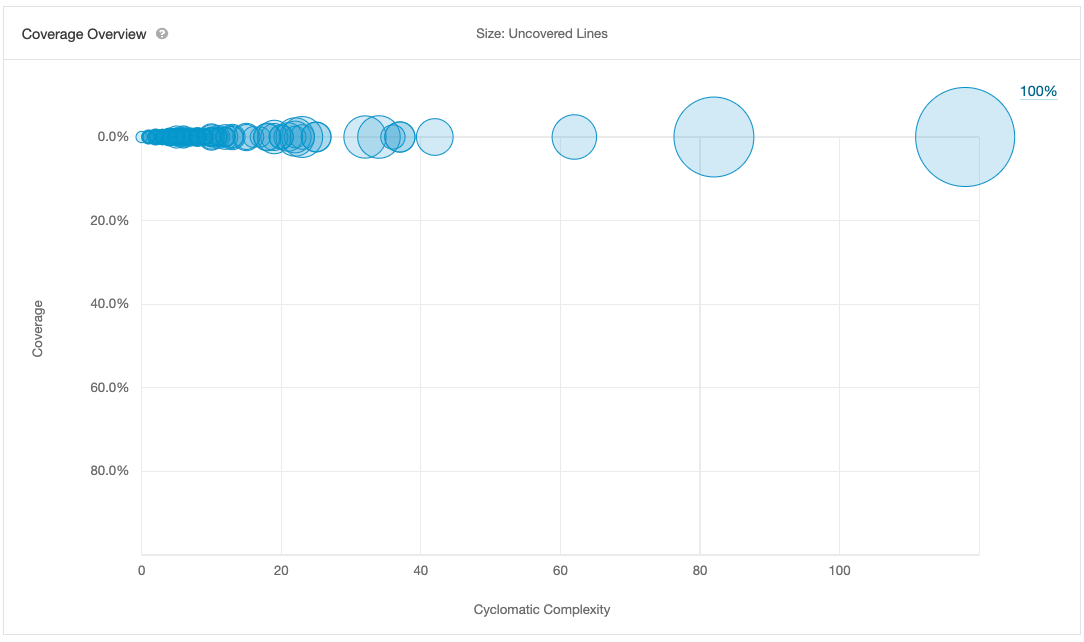
* Que peut-on en déduire sur la structuration des tests ?

(à compléter)

Pourcentage de code couvert par les tests :

Une image contenant texte

Description générée automatiquement



Etude des tests :

Webmagic-core : 47 tests réussis ; 3 tests ignorés

Webmagic-extension : 0 test réussi

Webmagic-samples : 0 test réussi

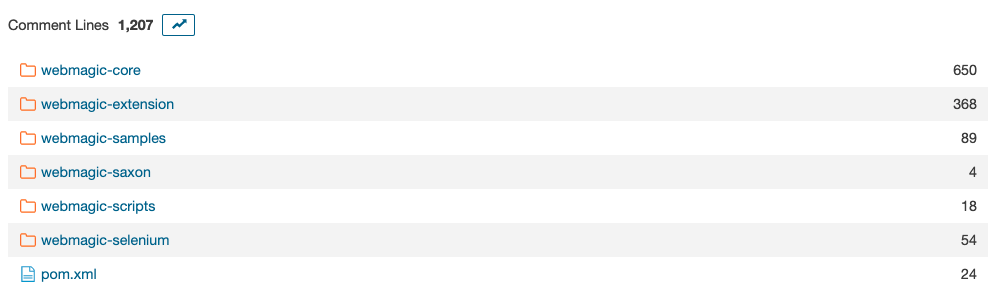
Webmagic-saxon : 4 tests réussis ; 2 tests ignorés

Webmagic-scripts : 1 test ignoré

Webmagic-selenium : 4 tests ignorés

## Commentaires

Nombre de lignes de commentaire totale : 1207



Type de commentaire utilisé :

* Javadoc : générateur de documentation pour le langage Java afin de générer une documentation API au format HTML à partir du code source Java.
* Commentaire pertinent : qui permet d’expliquer le fonctionnement de certaines méthodes et attributs afin de mieux les utiliser dans l’application

Les parties non commentées de l’application sont les fichiers HTML et ainsi que les classes de tests.

## Dépréciation

Dans l’application on constate la présence de code déprécié utilisé par du code non déprécié. Un élément de programme annoté @Deprecated est un élément dont l'utilisation est déconseillée aux programmeurs, généralement parce qu'il est dangereux ou parce qu'il existe une meilleure alternative. Les compilateurs avertissent lorsqu'un élément de programme déprécié est utilisé ou remplacé dans du code non déprécié. Notamment à l’aide de l’option de javac -deprecation permettant de donner des informations sur les méthodes dépréciées qui sont utilisées

Par exemple, dans la classe Page.java de webmagic-core:

*/\*\*  
 \** ***@param*** *html html  
 \** ***@deprecated*** *since 0.4.0  
 \* The html is parse just when first time of calling {****@link*** *#getHtml()}, so use {****@link*** *#setRawText(String)} instead.  
 \*/*

public void setHtml(Html html) {  
 this.html = html;  
}

Est une méthode dépréciée, la Javadoc précise pour cette méthode qu’elle est dépréciée depuis la version 0.4.0 et indique la bonne méthode à utiliser.

Au total on identifie 10 éléments dépréciés dans 8 classes. Pour chacun de ces éléments, on a l’information sur l’alternative à utiliser.

Les 10 éléments sont :

Projet : webmagic-extension classe : ComboExtract.java méthode : mutli() ligne : 82

Projet : webmagic-core classe : Page.java méthode : setHtml() ligne : 111

Projet : webmagic-core classe : Spider.java méthode : scheduler() ligne : 178

Projet : webmagic-core classe : Spider.java méthode : pipeline() ligne : 211

Projet : webmagic-core classe : Spider.java méthode : downloader() ligne : 261

Projet : webmagic- core classe : Html.java attribut : DISABLE\_HTML\_ENTITY\_ESCAPE ligne : 26

Projet : webmagic- core classe : Selector.java méthode : xsoup() ligne : 41

Projet : webmagic-core classe : UrlUtils.java méthode : encodeIllegalCharacterInUrl() ligne : 58

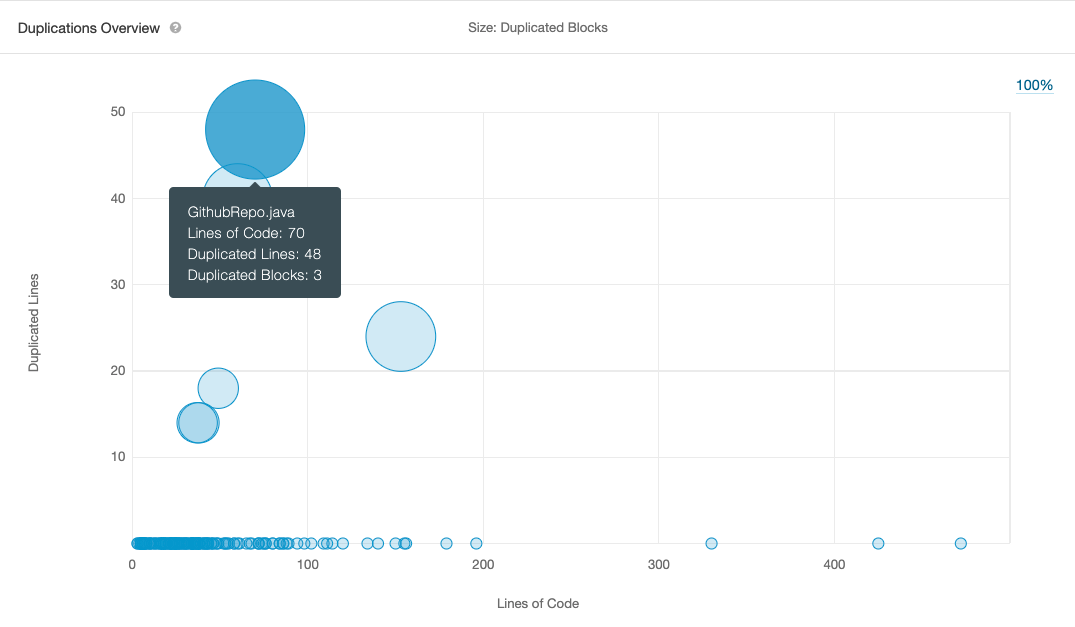
Projet : webmagic- extension classe : ExtractBy.java méthode : multi() ligne : 75

Projet : webmagic- extension classe : ExtractBy.java méthode : multi() ligne : 40

## Duplication de code

Une image contenant texte

Description générée automatiquement



## God Classes

Nombre de méthodes par projet

Webmagic-core : total = 463 ; moyenne = 4,8 ; max = 57

Webmagic-extension : total = 293 ; moyenne = 3,8 ; max = 17

Webmagic-samples : total = 94 ; moyenne = 2,4 ; max = 8

Webmagic-saxon : total = 6 ; moyenne = 6 ; max = 6

Webmagic-scripts : total = 28 ; moyenne = 4,7 ; max = 11

Webmagic-selenium : total = 15 ; moyenne = 7,5 ; max = 8

* On va analyser le nombre de méthode par classe dans le package webmagic du projet principal.

 Webmagic-core : us.codecraft.webmagic :

TOTAL = 157 | Minimum = 2 | Maximum = 57 | Moyenne = 22,4 | Médiane = 27

Nombre de variables d’instances par projet

Webmagic-core : total = 123 ; moyenne = 1,7 ; max = 21

Webmagic-extension : total = 130 ; moyenne = 1,7 ; max = 11

Webmagic-samples : total = 9 ; moyenne = 0,2 ; max = 2

Webmagic-saxon : total = 6 ; moyenne = 3 ; max = 3

Webmagic-scripts : total = 22 ; moyenne = 3,7 ; max = 6

Webmagic-selenium : total = 11 ; moyenne = 5,5 ; max = 7

* On va analyser le nombre de variables d’instances par classe dans le package webmagic du projet principal.

 Webmagic-core : us.codecraft.webmagic :

TOTAL = 64 | Minimum = 0 | Maximum = 25 | Moyenne = 9 | Médiane = 14

Nombre de lignes de code par projet

Webmagic-core : total = 1641 ; moyenne = 3,7 ; max = 55

Webmagic-extension : total = 130 ; moyenne = 1,7 ; max = 11

Webmagic-samples : total = 9 ; moyenne = 0,2 ; max = 2

Webmagic-saxon : total = 6 ; moyenne = 3 ; max = 3

Webmagic-scripts : total = 22 ; moyenne = 3,7 ; max = 6

Webmagic-selenium : total = 11 ; moyenne = 5,5 ; max = 7

* On va analyser le nombre de lignes de code par classe dans le package webmagic du projet principal.

 Webmagic-core : us.codecraft.webmagic :

TOTAL = 554 | Minimum = 0 | Maximum = 289 | Moyenne = 79 | Médiane = 289

**Comparer avec les résultats trouvés à la question précédente :**

## Analyse des méthodes

Le complexité cyclomatique est un outil de métrologie logicielle qui permet de mesurer la complexité d’un programme informatique. La complexité cyclomatique d’une méthode est définie par le nombre de chemins linéairement indépendants qu’il est possible d’emprunter dans cette méthode. Elle vaut au minimum 1 puisqu’il y a toujours au moins un chemin.

Une méthode avec une haute complexité cyclomatique est plus difficile à comprendre et à maintenir.

* Une complexité cyclomatique trop élevée (supérieur à 30) indique qu’il faut refactoriser la méthode ;
* Une complexité cyclomatique inférieur à 30 peut être acceptable si la méthode est suffisamment testée.

Complexité cyclomatique des méthodes par projet

Webmagic-core : moyenne : 1,7 ; maximum : 20

Webmagic-extension : moyenne : 1,8 ; maximum : 26

Webmagic-samples : moyenne : 1,4 ; maximum : 4

Webmagic-saxon : moyenne : 3 ; maximum : 8

Webmagic-scripts : moyenne : 1,8 ; maximum : 7

Webmagic-selenium : moyenne : 2,4 ; maximum : 9

* On va analyser la complexité cyclomatique des méthodes en général dans le package webmagic du projet principal.

Site.java : moyenne : 1,6 ; maximum : 20

Spider.java : moyenne : 2 ; maximum : 9

Request.java : moyenne : 1,3 ; maximum : 7

Page.java : moyenne : 1,4 ; maximum : 5

ResultItems.java : moyenne : 1,1 ; maximum : 2

Task.java : moyenne : 1 ; maximum : 1

SpiderListener.java : moyenne : 1 ; maximum : 1

Nombre de paramètres par méthodes pour chaque projet

Webmagic-core : moyenne : 0,9 ; maximum : 4

Webmagic-extension : moyenne : 0,8 ; maximum : 5

Webmagic-samples : moyenne : 0,6 ; maximum : 2

Webmagic-saxon : moyenne : 0,9 ; maximum : 2

Webmagic-scripts : moyenne : 0,7 ; maximum : 3

Webmagic-selenium : moyenne : 0,5 ; maximum : 2

* On va analyser le nombre de paramètres par méthodes en général dans le package webmagic du projet principal.

Site.java : moyenne : 0,6 ; maximum : 3

Spider.java : moyenne : 0,7 ; maximum : 2

Request.java : moyenne : 0,6 ; maximum : 2

Page.java : moyenne : 0,6 ; maximum : 2

ResultItems.java : moyenne : 0,6 ; maximum : 2

Task.java : moyenne : 0 ; maximum : 0

SpiderListener.java : moyenne : 1 ; maximum : 1

# Nettoyage de Code et Code smells

## Règles de nommage

On va s’intéresser ici aux règles de nommage. On pourra par exemple :

• vérifier que les noms utilisés dans le programme pour les différents éléments sont descriptifs et sans ambiguïté.

• vérifier que les noms ne participent pas à la désinformation en particulier pour les structures de données.

• vérifier que les noms sont prononçables et sont des termes du domaine.

## Nombre magique

On va s’intéresser ici aux nombres magiques, c’est à dire les nombres qui apparaissent dans le code. On pourra par exemple :

• identifier des nombres magiques dans le code.

• essayer de comprendre pourquoi telle valeur ou pas telle autre, en essayant de comprendre l’intention

• proposer de remplacer l’utilisation de ce nombre magique (dans le code et éventuellement dans les tests) par une constante.

## Structure du code

On va s’intéresser ici à la structure du code. On pourra par exemple :

• vérifier qu’au sein des classes, toutes les variables d’instance son ensemble, en début de classe.

• vérifier que les méthodes publiques et les plus utilisées sont en haut dans la définition des classes et les méthodes privés plus bas.

## Code mort

On va s’intéresser ici au code mort. On pourra par exemple :

• existe-t-il du code mort au sein du projet (rappel : code mort = code pas appelé) ? Attention si c’est une API, ça peut être normal que le code ne soit pas appelé en interne.

• ce code est-il testé ? maintenu ?

• ce code devrait-il être supprimé ?