

Rapport du Projet webmagic

Membres du groupe :

- BENMAOUHCE Mohamed
- HADJOU Tayeb

Présentation globale du projet :

L'activité de **crawling** d'un moteur de recherche désigne le processus par lequel un moteur va explorer et indexer les différents contenus rencontrés sur Internet en général ou sur un site web ou une application en particulier.



Pour un site web ou une application, le fait que ses contenus soient "crawlés" est le préalable à une indexation et à une visibilité plus ou moins favorable dans les résultats des moteurs de recherche.

WebMagic est une **implémentation d'un framework** de crawler simple mais évolutif qui peut servir à développer un robot d'exploration, parmi les caractéristiques de ce dernière, on citera :

- Noyau simple avec une grande flexibilité.
- API simple pour l'extraction **html**.
- Annotation avec **POJO** pour personnaliser un crawler
- Prise en charge du **multithread** et de la distribution
- Facile à intégrer.

L'utilisation de la librairie WebMagic se fait par déclaration de dépendance, par exemple pour les projets Maven il faut ajouter les dépendances respectives.

Afin de lancer ce projet, il faudra implémenter une classe qui implémentera la classe `pageProcessor` qui permet entre autre de récupérer les informations spécifiques à un site donné.

1.2) Description du projet :

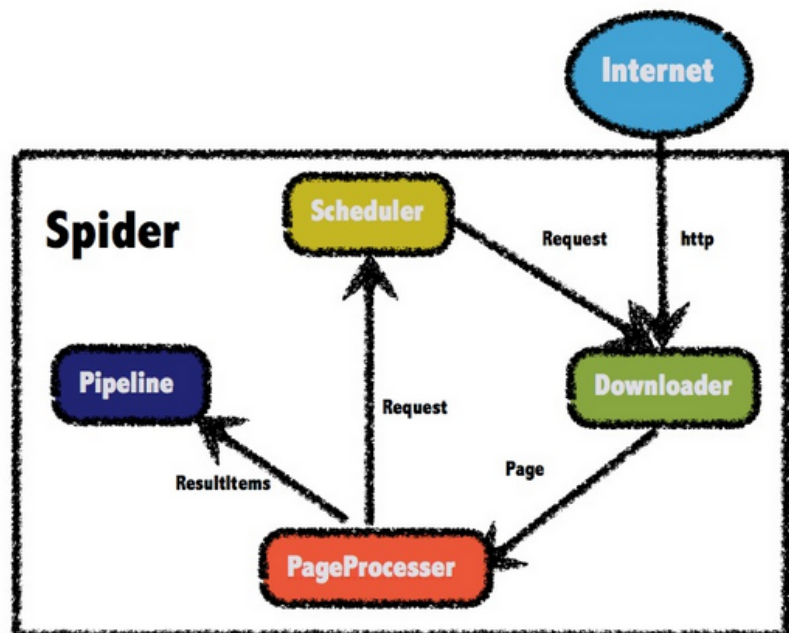
Le dépôt du projet comprend un fichier **readme** avec notamment une description brève du projet ainsi de son fonctionnement et d'un ensemble de caractéristiques de ce dernier.

Une rubrique dédiée à la bonne installation et ajout des dépendances au fichier `pom.xml` pour une bonne configuration

la rubrique **Get Started** nous donne plus de détails sur comment créer un crawler spécifique à l'aide de l'interface `PageProcessor`.

Un lien de redirection vers un site qui permet d'accéder à la documentation du projet est cependant mis à disposition des utilisateurs.

Une architecture et maquette du projet est jointe à ce readme ainsi qu'un ensemble de liens vers des projets similaires réalisés en java et en python.



La documentation présente au sein du site vers lequel l'utilisateur est organisé en rubriques définies comme suit :

-Introduction : permettant de présenter le projet, ses composants ainsi que son fonctionnement de manière détaillée.

Install : présente un ensemble d'instructions permettant d'installer et de faire fonctionner ce projet avec et sans maven, un premier exemple est ainsi fourni.

Build form Source :

cette rubrique permettra à l'utilisateur de pouvoir récupérer et d'importer le projet depuis Git et de le faire fonctionner en suivant les instructions qui y sont

fournies afin de mener à bien la de 'l'importation, la compilation et l'exécution de ce projet .

Basic crawler : permet d'éclairer l'utilisateur sur les fonctionnalités et le comportement de chaque composant du logiciel.

Customize Components : WebMagic permet de personnaliser la fonction des composants flexibles pour obtenir la fonction souhaitée.

Case Study : Même si le framework WebMagic a été très habile, il sera préparé à certains robots d'exploration et à une certaine confusion. Par exemple, comment récupérer et mettre à jour périodiquement, comment explorer le rendu dynamique des pages, etc.

Dans cette section, quelques cas courants sont présentés.

Historique du logiciel :

Analyse du git :

Le projet est en libre accès sur git, il est donc facile d'analyser l'historique du dépôt.

Depuis la création du dépôt le 21 avril 2013, 42 personnes ont contribué à la réalisation du projet. La contribution n'est pas égale entre toutes les personnes, on

distingue notamment une personne (code4craft) qui a effectué plusieurs dizaines voir centaines de milliers de modification soit 823 commits. On peut aussi mentionner le fait que la contribution au projet présente de grande disparité dans le temps. En effet, d'avril 2014 à décembre 2017, seule une personne (code4craft) mettaient à jour le dépôt, avant qu'il disparaisse des radars et laisse la place à de nouveaux contributeurs qui s'y sont mis à partir de 2020, on citera notamment sutra qui a effectué 54 commits ainsi **snyk-bot** qui a 14 commits à son compte, on constate ainsi une contribution très modeste de quelques personnes depuis 2020 qui sont venus s'ajouter au projet, certains en réalisant un seul commit.

De manière générale, le dépôt a connu des pics d'activité importants durant la première années du projet en 2014 ainsi qu'en 2017, année durant laquelle le contributeur principal a décidé de ne plus travailler dessus. Depuis 2 ans aucune modification n'a été faite avant de connaître une petite renaissance en 2020 à travers quelques commits réalisés par de modestes contributeurs

Par ailleurs, le projet comprend 29 branches dont 3 sont fermées et 26 toujours ouvertes. Sur les 26 branches ouvertes, certaines n'ont pas été utilisées

depuis plusieurs années. Concernant le mécanisme des pull requests, il est effectivement bien utilisé, on remarque que 147 ont été traités et donc fermés et que 32 sont actuellement toujours ouverts et que les derniers sont relativement récents. De ce point de vue, le dépôt doit probablement être encore actif.

Architecture logicielle :

utilisation de bibliothèques extérieure :

Dans l'ensemble du projet plusieurs importations sont faites, mais aucune librairie extérieure n'a été utilisée.

3.2) Organisation en paquetages :

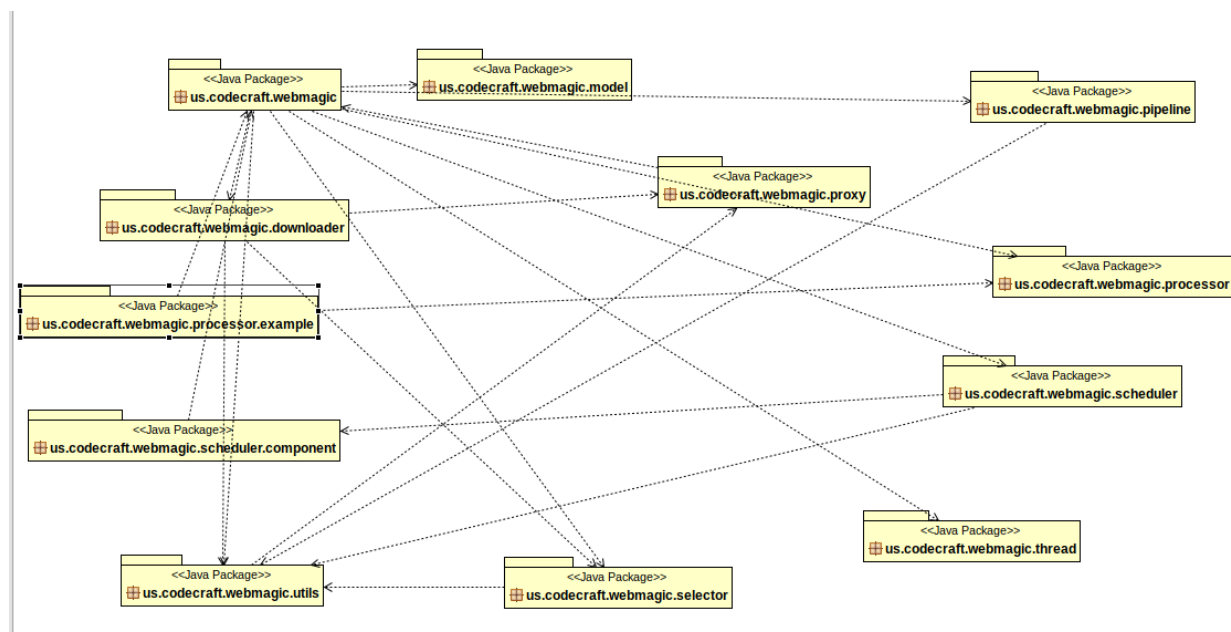
le projet est organisé en sous paquetages au sein du package webmagic,

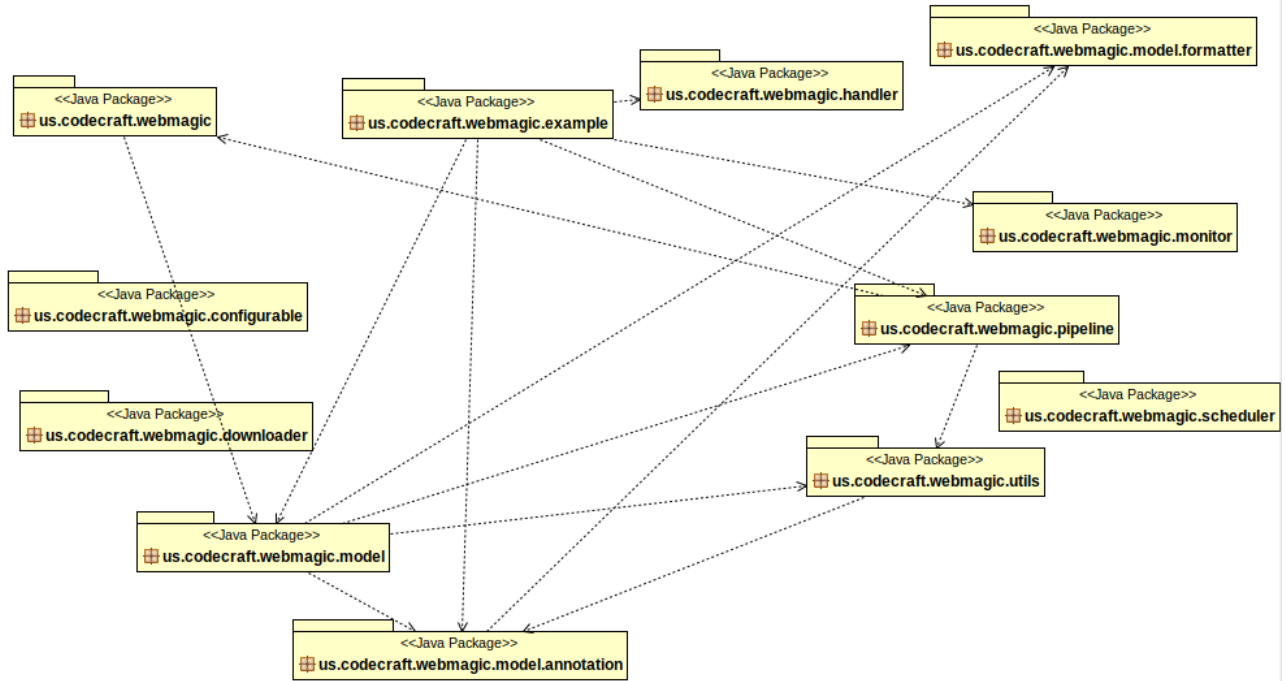
Il y a un package principal contenant 6 sous packages.

La hiérarchie des packages pour les tests suit la hiérarchie des packages sources avec

des packages en plus. Il y a donc des hiérarchie parallèle de paquetage, l'organisation et la répartition en sous-paquetage peut être illustré comme ceci :

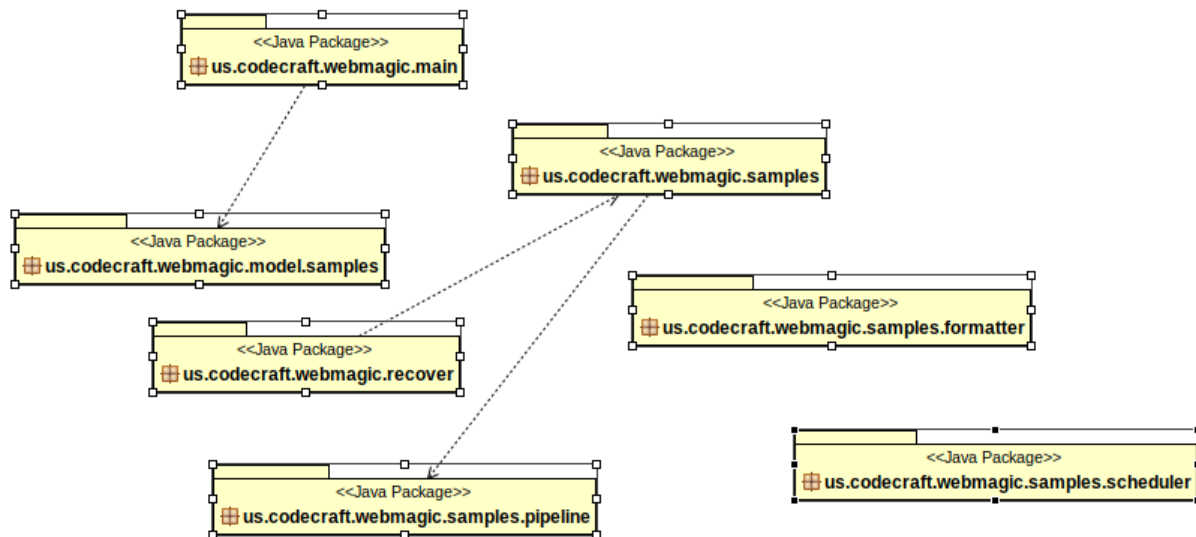
webmagic-core:



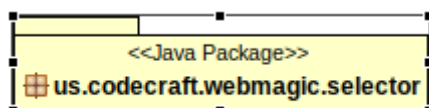


webmagic-extension:

webmagic-samples:



webmagic-saxon:



webmagic-selenium:



Répartition des classes dans les paquetages :

- d'après les ressources et données récoltés à l'aide de l'outil SonarQube, en total, 200 classes ont été créées, et réparties de la manière suivante :

webmagic-parent


View as

Tree

↑ ↓ to select files

← → to navigate

8 files

Classes 200 

webmagic-core

70

webmagic-coverage

—

webmagic-extension

77

webmagic-samples

42

webmagic-saxon


2

webmagic-scripts

7

webmagic-selenium

2

 pom.xml

—

8 of 8 shown

- On peut notamment s'apercevoir que les paquetages **webmagic-core**, **webmagic-coverage** ainsi que **webmagic-samples** contiennent le plus de classes.

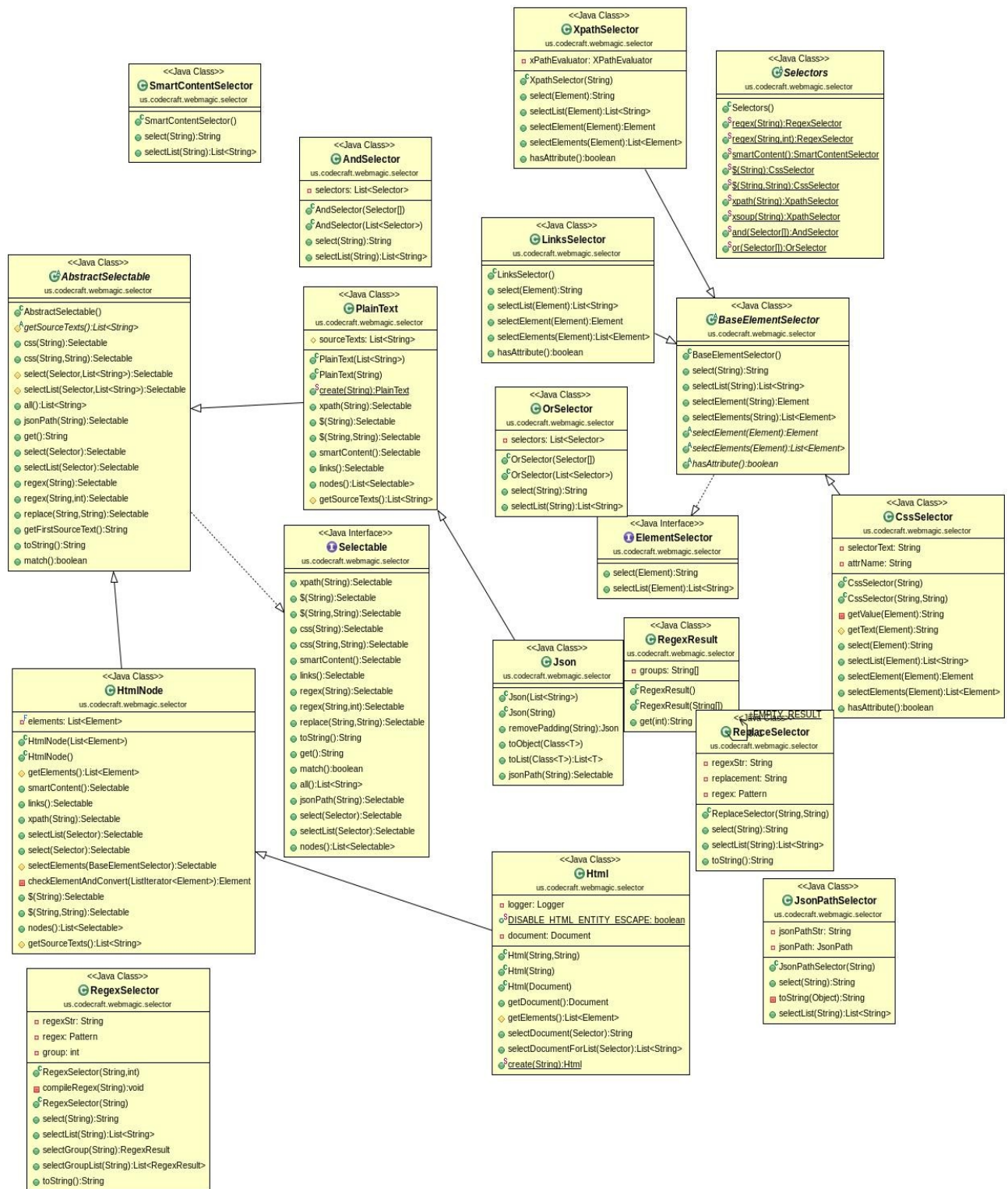
Chiffres clés:

- Nombre minimal de classe par paquetage : 2
- Nombre moyen de classe par paquetage : 10,28
- Nombre maximal de classe par paquetage : 70

L'ensemble des classes sont dans des paquetages src et tests, plusieurs paquetages à la racine du projet contiennent ce type de sous paquetages (hiérarchie) .

Organisation des classes :

La hiérarchie est plutôt profonde car la profondeur de l'arbre d'héritage du paquetage est de 4 et le nombre d'enfant par classe est en moyenne de 2.64 .



us.codecraft.webmagic.selector package

Analyse approfondie :

Tests :

Concernant les tests, le projet comprend 74 méthodes de tests(tests unitaires tous répartis

dans 44 fichiers . Chaque méthode de test contient ou non plusieurs assertions, le pourcentage de code couvert par les tests est de 35,3 %, un grand ensemble de classes n'ont pas été couvertes comme le montre les illustrations suivantes :

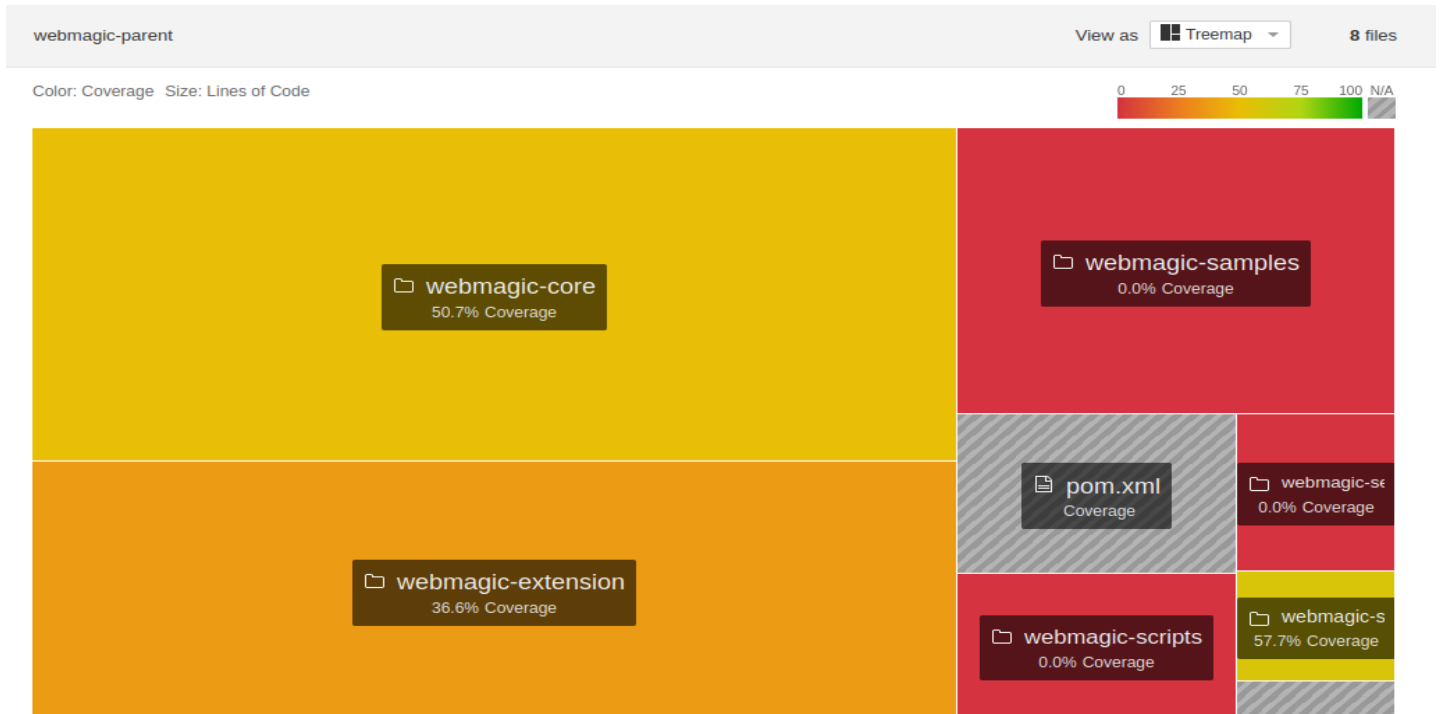


Figure 1: pourcentage de couverture des différents paquets

webmagic-parent View as List 153 files

	Coverage	Uncovered Lines	Uncovered Conditions
webmagic-samples/src/main/java/us/codecraft/webmagic/samples/AlexanderMcqueenGoodsProcessor.java	0.0%	25	4
webmagic-samples/src/main/java/us/codecraft/webmagic/samples/AmazonPageProcessor.java	0.0%	17	6
webmagic-samples/src/main/java/us/codecraft/webmagic/samples/AngularJSProcessor.java	0.0%	15	6
webmagic-extension/src/main/java/us/codecraft/webmagic/example/AppStore.java	0.0%	8	–
webmagic-extension/src/main/java/us/codecraft/webmagic/example/BaiduBaiké.java	0.0%	19	2
webmagic-core/src/main/java/us/codecraft/webmagic/processor/example/BaiduBaikéPageProcessor.java	0.0%	22	2
webmagic-samples/src/main/java/us/codecraft/webmagic/model/samples/BaiduNews.java	0.0%	9	–
webmagic-extension/src/main/java/us/codecraft/webmagic/pipeline/CollectorPageModelPipeline.java	0.0%	5	–
webmagic-extension/src/main/java/us/codecraft/webmagic/handler/CompositePageProcessor.java	0.0%	20	10
webmagic-extension/src/main/java/us/codecraft/webmagic/handler/CompositePipeline.java	0.0%	15	10
webmagic-extension/src/main/java/us/codecraft/webmagic/model/ConsolePageModelPipeline.java	0.0%	3	–
webmagic-core/src/main/java/us/codecraft/webmagic/pipeline/ConsolePipeline.java	0.0%	6	2
webmagic-samples/src/main/java/us/codecraft/webmagic/samples/scheduler/DelayQueueScheduler.java	0.0%	26	4
webmagic-samples/src/main/java/us/codecraft/webmagic/samples/DiandianBlogProcessor.java	0.0%	12	2
webmagic-samples/src/main/java/us/codecraft/webmagic/model/samples/DianpingFtldataScanner.java	0.0%	9	6
webmagic-samples/src/main/java/us/codecraft/webmagic/samples/DiaoyuwengProcessor.java	0.0%	17	4
webmagic-extension/src/main/java/us/codecraft/webmagic/utills/DoubleKeyMap.java	0.0%	31	10

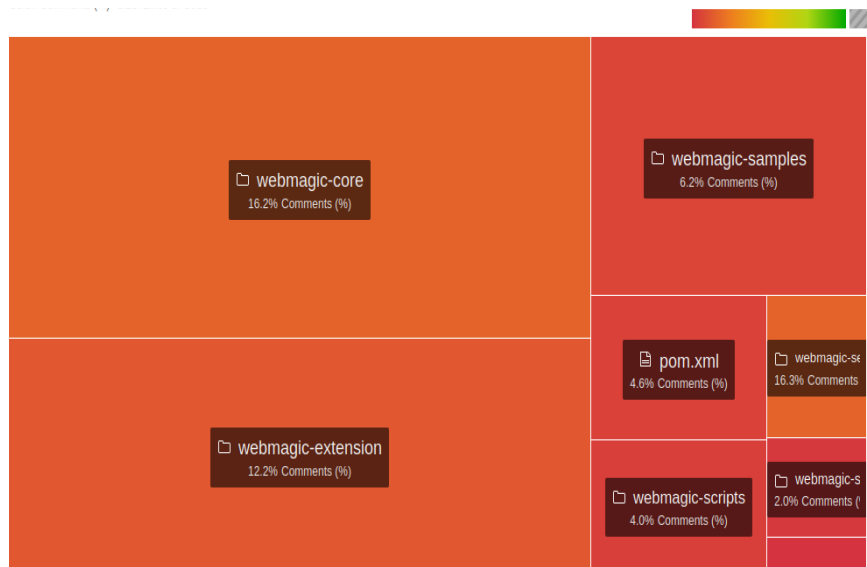
Figure 2: pourcentage de couverture de quelques classes

- Cependant 74 tests unitaires ont été réalisés avec succès et 21 tests unitaires ont été ignorés et pas exécutés d'après les informations recueillies par Sonarqube.

Commentaires :

- 1224 lignes de commentaires ont été rédigées soit 12 % du nombre total de lignes codées, il s'agirait principalement de la **javadoc**, on en trouve aussi des commentaires explicatifs et illustratif du code associé.

- un grand nombre de classes et de méthodes n'ont pas été commentées, ces illustrations viennent appuyer nos statistiques et chiffres avancés :



webmagic-parent

View as

Tree

↑

↓

to select files

←

→

to navigate

8 files

Comment Lines 1,224

webmagic-core

663

webmagic-coverage

0

webmagic-extension

368

webmagic-samples

92

webmagic-saxon

4

webmagic-scripts

19

webmagic-selenium

54

pom.xml

24

Duplication de code : après une longue observation et étude du code de ce projet et en s'appuyant notamment sur les données recueillis par Sonarqube, on déduit qu'un ensemble de 158 lignes ont été dupliquées soit 1,2 % du code total rédigé.

webmagic-parent

View as

List

↑

↓

to select files

←

→

to navigate

253 files

Duplicated Lines 158







	Duplicated Lines	Duplicated Lines (%)
 webmagic-extension/src/main/java/us/codecraft/webmagic/example/GithubRepo.java	48	50.5%
 webmagic-samples/src/main/java/us/codecraft/webmagic/model/samples/GithubRepo.java	40	48.2%
 webmagic-saxon/src/main/java/us/codecraft/webmagic/selector/Xpath2Selector.java	24	13.2%
 webmagic-extension/src/main/java/us/codecraft/webmagic/example/GithubRepoApi.java	18	25.4%
 webmagic-extension/src/main/java/us/codecraft/webmagic/pipeline/FilePageModelPipeline.java	14	25.0%
 webmagic-extension/src/main/java/us/codecraft/webmagic/pipeline/JsonFilePageModelPipeline.java	14	24.6%

Figure 2: Répartition du code dupliqué par classes

God Classes :

Dans cette partie, on s'intéresse à l'identification de god classes. Tout d'abord, il est

nécessaire de relever certaines données pour ensuite pouvoir les analyser.

Le nombre maximal de méthodes par classe est de 60 ([webmagic-core/src/main/java/us/codecraft/webmagic/Spider.java](#)) et au minimum de 0 à l'exemple de la classe :

[webmagic-samples/src/main/java/us/codecraft/webmagic/samples/pipeline/ReplacePipeline.java](#), la moyenne est cependant de 4,78 méthodes/classe.

• Pour les instances :

- au total on en dénombre 3013 instances
- nombre minimal d'instance par classe : 0
- nombre maximal d'instance par classe : 245

• Pour les lignes de code :

- au total on dénombre 8078 lignes de code 8950
- nombre minimal de ligne de code par classe : 3 (Experimental.java)
- nombre maximal de ligne de code par classe : 497 (spider.java)

webmagic-parent

master

Last analysis had 2 warnings

March 9, 2022, 2:13 PM Version 0.7.6-SNAPSHOT

OverviewIssuesSecurity HotspotsMeasuresCodeActivity

Project SettingsProject Information

Project Overview

webmagic-parent

View asTree

to select files

to navigate

8 files

Statements 3,013

webmagic-core1,349

webmagic-coverage-

webmagic-extension954

webmagic-samples382

webmagic-saxon71

webmagic-scripts159

webmagic-selenium98

pom.xml-

8 of 8 shown

Size STATEMENTS

Lines of Code8,950

Lines13,334

Statements3,013

Functions956

Classes200

Files195

Comment Lines1,224

Comments (%)12.0%

Complexity

D'après les recherches qu'on a mené, on peut reconnaître une God Class par un manque de cohésion de la classe, une grande complexité cyclomatique et une forte dépendance aux autres classes. Ainsi nous avons pu constater que les classes Spider.java, Site.java et Page.java pourraient être considérées comme des god classes. Nous allons montrer cela en nous appuyant sur des chiffres:

<u>Classe :</u>	<u>Complexité cyclomatique :</u>
Spider.java	121
Site.java	62
Page.java	42

Analyse des méthodes :

On va s'intéresser à présent à la complexité cyclomatique des méthodes, il s'agit du nombre de chemins linéairement indépendants qu'il est possible d'emprunter dans une méthode. Plus simplement, cela correspond au nombre de points de décision de la méthode, la complexité cyclomatique d'une méthode vaut au minimum 1, puisqu'il y a toujours au moins un chemin. Voici les données récupérées par Sonarqube sur l'ensemble du projet :

Complexité cyclomatique minimale : 1 (ou 0 car dans certains fichiers les méthodes ne sont pas définies)

- Complexité cyclomatique maximale : 60
- Complexité cyclomatique moyenne : 15,6

- un nombre important d'erreurs est détecté principalement au sein de la classe Spider, la gestion des erreurs au sein des différentes fonctions aurait pu être mieux gérée.

Nettoyage de Code et Code smells :

- De manière générale, les noms utilisés dans le programme pour les différents éléments sont plutôt descriptifs et sans ambiguïté.

Nombre Magique :

- On peut constater qu'il n'y a pas de nombre magique et que les nombres sont contenus dans des variables qui ont des noms explicites. De ce point de vue, on peut

dire que le programme est bien conçu car les données numériques sont compréhensibles.

Code smells et bugs :

Le projet comprend 43 bugs répartis sur 17 fichiers, dont des fichiers java tel que: Spider.java ainsi qu'un grand nombre de fichier Html.

Classe	Nombre de bugs
Spider.java	4
SeleniumDownloader.java	3
/model/package.html	3

Il s'agirait principalement de problèmes liés à la gestion d'erreurs et exceptions, ces derniers ne sont pas récents puisque le plus récent date de 2017.

Il existe un ordre de gravité dans ces bugs car 39 sont classés comme majeurs par Sonarqube, 3 autres comme mineurs et 1 bloqueur.

Les bugs mineurs ne demandant pas beaucoup d'efforts chacun et peuvent corrigés sans beaucoup de difficulté.

Code Smells :

Concernant les code smells, le programme en compte 565 classés comme ceci :

- 217 mineurs
- 239 majeurs
- 70 critiques plusieurs blocs de code peuvent être refactorisés pour réduire → leur complexité cognitive
- 21 bloqueurs indiquent le manque d'assertion à certains endroits du code dans → des fichiers tests.

Structure du code :

Nous avons vérifié si toutes les variables d'instances sont bien groupées et en début de classe, c'est en effet le cas dans le programme. Aussi, nous avons pu constater que plusieurs interfaces sont implémentées, aussi la notion d'héritage est

utilisé à de multiples reprises.

Sécurité et maintenance :

Grâce à Sonarqube, on peut obtenir plusieurs indicateurs sur le projet.

Concernant la sécurité, 4 vulnérabilités (2 classées comme critiques et 2 classées comme bloqueurs) ont été détectées et signalées et l'examen de la sécurité est noté E.

Pour conclure, on peut dire que l'analyse effectuée principalement à l'aide de l'outil SonarQube ainsi qu'en s'appuyant sur les données collectées grâce à Eclipse nous a permis de nous fixer les idées par rapport au projet, à ses manquements, bugs et vulnérabilités en matière de sécurité qui sont très importantes et nécessiteront une correction adaptée et rigoureuse lors de la prochaine étape du projet.