

## Améliorations du projet Java *Webmagic*

Matthieu Medeng Essia

Chargé de td : Maxime Savary-Leblanc  
Professeur : Anne Etien

Propriétaire : code4craft \*

Dépôt GitHub : [https ://github.com/medengessia/webmagic](https://github.com/medengessia/webmagic)

Licence Informatique-Mathématiques - Troisième année  
2021/2022

---

\*[https ://github.com/code4craft](https://github.com/code4craft)

## Table des matières

<b>1</b>	<b>Compte-rendu des améliorations apportées</b>	<b>3</b>
<b>2</b>	<b>Gestion des God Classes</b>	<b>3</b>
2.1	Les 60 méthodes de la classe Spider . . . . .	3
2.2	Les classes Site, Page et Request . . . . .	4
<b>3</b>	<b>Amélioration de la couverture de tests</b>	<b>4</b>
3.1	Héritage de tests . . . . .	4
3.2	Ajouts de nouveaux tests unitaires . . . . .	4
<b>4</b>	<b>Gestion du code déprécié</b>	<b>5</b>
4.1	Utilisation de solutions alternatives . . . . .	5
4.2	Ajouts de warnings pour prévenir l'appel au code déprécié . . . . .	5
<b>5</b>	<b>Réorganisation des paquetages</b>	<b>6</b>
5.1	Rééquilibrage des paquetages . . . . .	6
5.2	Création de nouveaux paquetages . . . . .	7
<b>6</b>	<b>Règles de nommage</b>	<b>7</b>
6.1	Travail de traduction . . . . .	7
<b>7</b>	<b>Embellissement de l'aspect global du projet</b>	<b>8</b>
7.1	Ajout de la javadoc et de commentaires . . . . .	8
7.2	Placement optimal des méthodes en fonction de l'utilité et de l'encapsulation . . .	8
<b>8</b>	<b>Bilan et perspectives</b>	<b>8</b>

## Table des figures

1	Aspect général de la classe Spider avant amélioration . . . . .	3
2	Couverture de tests avant amélioration . . . . .	4
3	Couverture de tests après amélioration . . . . .	5
4	Répartition du nombre de classes dans les paquetages avant amélioration . . . . .	6
5	Répartition du nombre de classes dans les paquetages avant amélioration . . . . .	7

# 1 Compte-rendu des améliorations apportées

Ceci est un rapport faisant office de compte-rendu des différentes améliorations apportées au projet webmagic<sup>1</sup>, suite à l'analyse approfondie de ce dernier. Il sera constitué de plusieurs points traitant de la gestion des God Classes et du code déprécié, de l'amélioration de la couverture de tests, de la réorganisation des paquetages, des règles de nommage, puis de l'embellissement de l'aspect global du projet.

## 2 Gestion des God Classes

### 2.1 Les 60 méthodes de la classe Spider

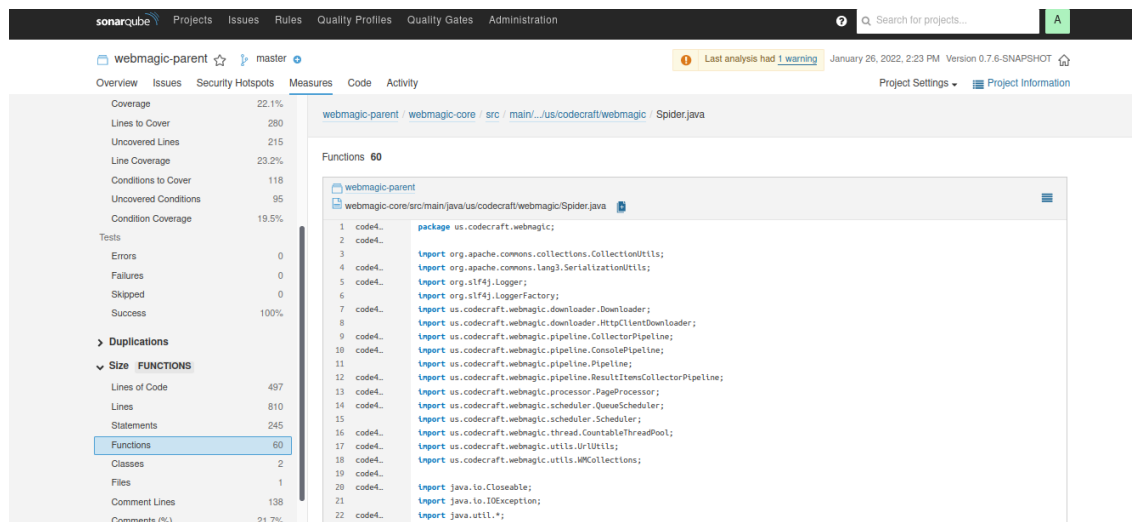


FIGURE 1 – Aspect général de la classe Spider avant amélioration

Rassemblant en tout 60 méthodes et faisant plus de 810 lignes comme l'indiquent les chiffres tirés de SonarQube, la classe Spider est la classe la plus volumineuse de toute la hiérarchie principale du projet : la hiérarchie core. Elle présente plusieurs fonctionnalités, celles de planificateur, de téléchargeur, de processeur de page et de pipeline.

L'examen de cette classe a permis de constater deux problèmes majeurs. Premièrement, une classe enum (Status) figurait dans le code de la classe Spider, ce qui est inapproprié et rallonge inutilement le nombre de lignes de code de la classe. Deuxièmement, le nombre effarant de getters et de setters ajouté aux méthodes spécifiques de mise en oeuvre du crawler ont éveillé la possibilité d'existence d'une classe qui ferait hériter ses getters et setters ainsi que d'autres méthodes plus génériques à la classe Spider.

Ainsi, l'objectif aura été dans un premier temps de dissocier la classe enum Status de la classe Spider, puis de créer une super classe SuperSpider dont Spider hériterait des getters, des setters et d'autres méthodes et prédicats, faisant ainsi basculer la majeure partie des attributs ainsi que la quasi-totalité des getters et setters dans la classe SuperSpider. Cela a permis de faire de certaines méthodes comme run(), setScheduler() ou addUrl(), des méthodes abstraites, selon les besoins du code ou de créer de nouveaux getters et setters pour réaliser des tests et augmenter la couverture du code.

Au final, d'une classe de 60 méthodes, il aura été possible de faire deux de 30, ce qui, en soi, représente une belle réduction de moitié. Par ailleurs, la classe fait désormais 547 lignes au lieu de 810.

1. <https://github.com/code4craft/webmagic>

Détail non négligeable, ces modifications, si soigneuses soient-elles, générèrent inévitablement des erreurs dans le reste du code qui ne connaît pas le super type SuperSpider et son constructeur, particulièrement lorsqu'il s'agit des tests. C'est à ce moment précis que les méthodes citées plus haut sont devenues des méthodes abstraites. Les autres erreurs étaient dues à des conflits de type et ont pu facilement être réglées à l'aide de casts sur le type Spider.

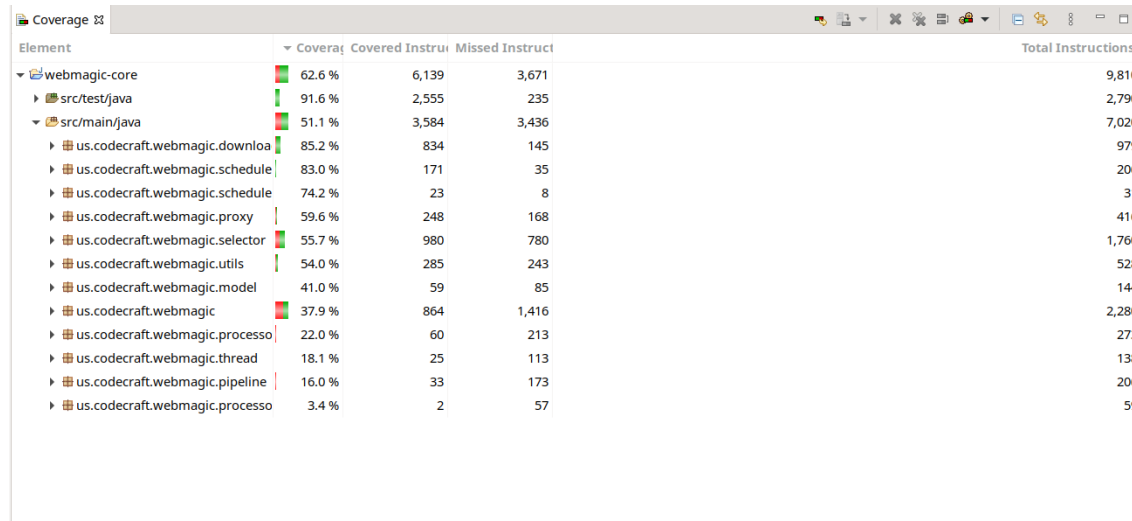
## 2.2 Les classes Site, Page et Request

Ces classes présentent en effet un nombre de méthodes supérieur à la moyenne. Toutefois, en les examinant de plus près, on se rend compte qu'elles contiennent essentiellement des getters et des setters et gèrent une seule fonctionnalité, ce qui rend leur comportement indivisible. On pourrait à la limite compléter la javadoc et/ou rajouter des commentaires pour mieux expliciter le code (cf partie 7), mais cela n'ira pas plus loin.

## 3 Amélioration de la couverture de tests

### 3.1 Héritage de tests

Il fut particulièrement plaisant de traiter cette partie car elle fut l'occasion d'appliquer le principe d'héritage de tests étant donné l'apparition de la classe SuperSpider. Il était donc logique que cette nouvelle classe se devait d'être testée. C'est ainsi qu'apparaît la classe SuperSpiderTest qui dispose d'une factory méthode protégée, se nommant create() et qui attend que le bon type de sous-classe (Spider en l'occurrence) soit créé et passe tous les tests qu'elle aura défini.



Element	Coverage	Covered Instru	Missed Instruct	Total Instructions
webmagic-core	62.6 %	6,139	3,671	9,810
src/test/java	91.6 %	2,555	235	2,790
src/main/java	51.1 %	3,584	3,436	7,020
us.codecraft.webmagic.download	85.2 %	834	145	979
us.codecraft.webmagic.schedule	83.0 %	171	35	206
us.codecraft.webmagic.schedule	74.2 %	23	8	31
us.codecraft.webmagic.proxy	59.6 %	248	168	416
us.codecraft.webmagic.selector	55.7 %	980	780	1,760
us.codecraft.webmagic.utils	54.0 %	285	243	528
us.codecraft.webmagic.model	41.0 %	59	85	144
us.codecraft.webmagic	37.9 %	864	1,416	2,280
us.codecraft.webmagic.processor	22.0 %	60	213	273
us.codecraft.webmagic.thread	18.1 %	25	113	138
us.codecraft.webmagic.pipeline	16.0 %	33	173	206
us.codecraft.webmagic.processor	3.4 %	2	57	59

FIGURE 2 – Couverture de tests avant amélioration

### 3.2 Ajouts de nouveaux tests unitaires

Il est tout d'abord à noter qu'au départ, la hiérarchie principale possède une couverture de tests de 62.6%. L'objectif ici fut de couvrir 80% du code de la hiérarchie principale.

Pour ce faire, un examen des classes les moins couvertes et des méthodes qu'il restait à couvrir s'imposait et a permis d'orienter les nouveaux tests. Ainsi, dès qu'une méthode était surlignée en rouge après un test de couverture sur eclipse, un test unitaire spécifique à cette méthode était créé. Par exemple, la classe HttpRequestBodyTest a elle aussi été rajoutée en raison de la trop faible couverture de la classe HttpRequestBody.

Dans la plupart des cas, les getters et les setters faisaient partie du code non couvert, ce qui voulait dire qu'ils n'étaient pas appelés au cours des différents tests. Or, comment les appeler étant donné que ces méthodes n'ont pas vocation d'être testées ? On teste la création d'une instance de la classe qui les contient. Et effectivement, ces classes (SpiderTest en fait partie) ne possédaient pas de méthode de test dédiée à la création d'instances. De nombreux tests de ce type ont donc été ajoutés et ont drastiquement amélioré la couverture de tests.

Un autre point critique était la gestion des exceptions. Celles-ci n'étaient pas testées. C'est le cas de l'IllegalArgumentException et l'IllegalStateException de la classe SuperSpider ou encore de HttpRequestBody. D'autres tests unitaires de ce type ont donc été ajoutés. Au final, cette démarche a pu mener à une progression de 15% dans la couverture de tests. La hiérarchie principale du projet est désormais couverte à 77.5% .

Element	Coverage	Covered Instruction...	Missed Instructions	Total Instructions
webmagic-core	77.5 %	8,204	2,383	10,587
src/main/java	70.1 %	4,929	2,103	7,032
us.codecraft.webmagic.manager	73.1 %	1,551	570	2,121
us.codecraft.webmagic.processor.example	21.9 %	60	214	274
us.codecraft.webmagic.selector.extender	51.1 %	218	209	427
us.codecraft.webmagic.utils	61.9 %	327	201	528
us.codecraft.webmagic.selector.implement	77.8 %	613	175	788
us.codecraft.webmagic.selector	68.3 %	372	173	545
us.codecraft.webmagic	46.5 %	146	168	314
us.codecraft.webmagic.downloader	86.4 %	846	133	979
us.codecraft.webmagic.proxy	72.6 %	302	114	416
us.codecraft.webmagic.pipeline	63.1 %	130	76	206
us.codecraft.webmagic.thread	76.8 %	106	32	138
us.codecraft.webmagic.scheduler	85.4 %	176	30	206
us.codecraft.webmagic.scheduler.compon	74.2 %	23	8	31
us.codecraft.webmagic.processor	100.0 %	59	0	59
src/test/java	92.1 %	3,275	280	3,555

FIGURE 3 – Couverture de tests après amélioration

## 4 Gestion du code déprécié

### 4.1 Utilisation de solutions alternatives

Deux types d'appels à du code déprécié sont présents dans le projet : l'utilisation de code déprécié par le programmeur lui-même, pour lequel ce dernier prévoit des méthodes alternatives pour éviter que les méthodes dépréciées soient utilisées, et l'utilisation de code déprécié dans la librairie Java invoquée. Le premier type étant le moins compliqué à gérer, les méthodes dépréciées qui ont été appelées ont été remplacées par les alternatives prévues par le programmeur. Ce fut le cas notamment de onError() et setHtml(), respectivement remplacées par une autre version de onErro() et par setRawText().

### 4.2 Ajouts de warnings pour prévenir l'appel au code déprécié

Le deuxième type s'avère plus dérangeant que le premier car le code déprécié ne vient pas du programmeur mais plutôt de la librairie Java employée. Dans ce cas, le code déprécié ne propose pas directement une méthode alternative, ce qui requiert une investigation plus poussée.

Le temps imparti n'aura hélas pas permis de poursuivre l'analyse. Cependant, il aurait fallu soit chercher dans la librairie Java appelée, le code déprécié, sa définition et voir si une autre méthode aurait été pourvue en tant que solution alternative, ou dans le cas où une telle solution

n'existerait pas, créer une méthode alternative définissant les mêmes fonctionnalités après en avoir pris connaissance.

En attendant, des balises `@SuppressWarnings("deprecation")` ont été ajoutées lorsque ce genre de difficultés ont été rencontrées.

## 5 Réorganisation des paquetages

### 5.1 Rééquilibrage des paquetages

Dans la hiérarchie principale, depuis le rapport d'analyse, plusieurs problèmes d'organisation des paquetages se sont faits ressentir. Citons, entre autres, le nombre disproportionné de classes par paquetage, le fait que plusieurs interfaces ou classes abstraites se trouvent dans le même paquetage que les classes qui les implémentent ou héritent d'elles, ou que certains paquetages possèdent simplement trop de classes.

L'objectif aura donc été de trier par pertinence les classes pouvant figurer dans un paquetage précis, puis de déplacer celles non nécessaires dans un autre paquetage, nouvellement créé dans la plupart des cas.

À titre d'exemple, la hiérarchie core avait un nombre déséquilibré de classes par paquetages : 7 dans les paquetages `us.codecraft.webmagic` et `us.codecraft.webmagic.downloader`, 1 dans les paquetages `us.codecraft.webmagic.thread` et `us.codecraft.webmagic.model` et 20 dans le paquetage `us.codecraft.webmagic.selector`, pour ne citer que ceux-là. Pour illustrer cela, un diagramme prrsentant la proportion de classes dans les paquetages met en évidence ces disparités.

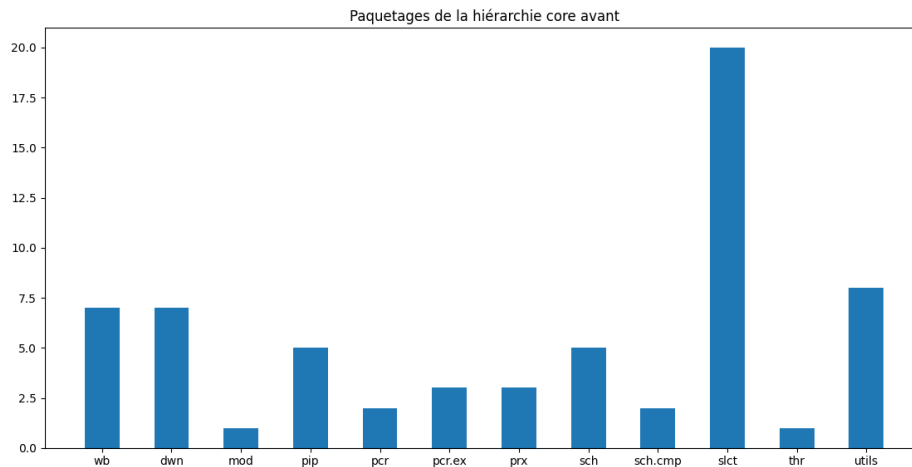


FIGURE 4 – Répartition du nombre de classes dans les paquetages avant amélioration

Maintenant, tous les paquetages contiennent chacun entre 3 et 7 classes à l'exception de certains qui en ont soit 1, soit 2, soit 8. C'est le cas du paquetage `us.codecraft.webmagic.selector` qui ne possède plus 20 classes, mais en possède désormais 8, tout comme le paquetage `us.codecraft.webmagic.utils`. Le paquetage `us.codecraft.webmagic` ne possède non plus 7, mais 4 classes.

De plus, Les paquetages `us.codecraft.webmagic.downloader` et `us.codecraft.webmagic.selector.implementer` comptent quant à eux 7 classes. Le nouveau paquetage `us.codecraft.webmagic.manager` possède 6 classes, tandis que les paquetages `us.codecraft.webmagic.pipeline`, `us.codecraft.webmagic.scheduler` et `us.codecraft.webmagic.selector.extender` en contiennent 5.

Pour finir, les paquetages `us.codecraft.webmagic.proxy` et `us.codecraft.webmagic.processor.example` possèdent 3 classes, tandis que les paquetages `us.codecraft.webmagic.processor` et `us.code-`

craft.webmagic.scheduler.component en possèdent 2, us.codecraft.webmagic.thread étant le seul à ne contenir qu'une classe.

## 5.2 Création de nouveaux paquets

Chaque paquetage nouvellement créé dans la hiérarchie principale, a été ajouté pour équilibrer le nombre de classes par paquetage et rassembler les classes accueillies selon des critères de fonctionnalités de même nature.

Essentiellement, trois paquetages ont été ajoutés dans le code source : les paquetages us.codecraft.webmagic.manager, us.codecraft.webmagic.selector.extend et us.codecraft.webmagic.selector.implementer. Le premier a pour vocation de rassembler les classes implémentant le crawler en son sein afin que celles-ci soient séparées des interfaces ou des classes abstraites qu'elles étendent. Ainsi, dans le paquetage us.codecraft.webmagic, se trouvent uniquement les interfaces SpiderListener et Task, l'enum Status et la classe abstraite SuperSpider. Les deux autres ont pour objectif de séparer tout aussi efficacement les interfaces et classes abstraites du paquetage us.codecraft.webmagic.selector des classes qui les étendent ou les implémentent. Ainsi, la partie extend contiendra toutes les classes de selector qui héritent d'une classe abstraite, tandis que la partie implementer contiendra toutes celles qui implémenteront une interface de selector. Voici à présent un diagramme indiquant la répartition du nombre de classes par paquetage après amélioration.

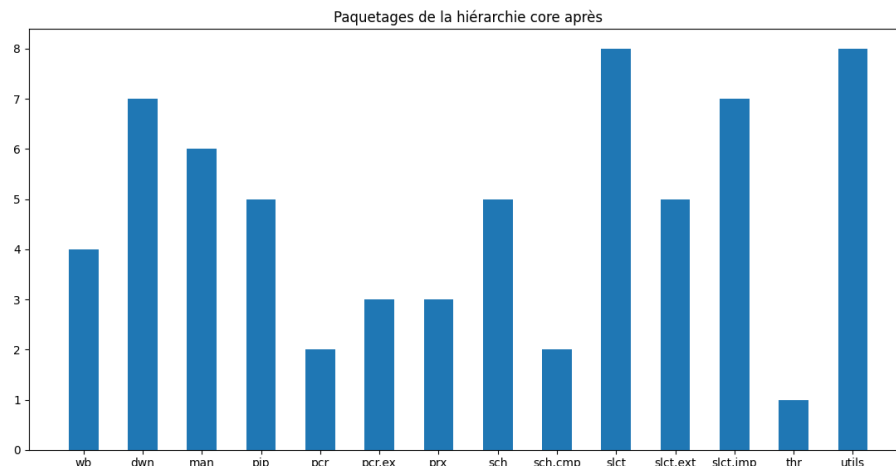


FIGURE 5 – Répartition du nombre de classes dans les paquetages avant amélioration

## 6 Règles de nommage

### 6.1 Travail de traduction

Dans la hiérarchie principale, dans les classes CustomRedirectStrategy, BaiduBaikePageProcessor et bien d'autres, toutes les informations en mandarin ont été conservées et traduites entre parenthèses ou écrites en commentaires sur la ligne où elles apparaissent.

Il s'agissait d'utiliser un dictionnaire (et non un traducteur, j'en profitais pour enrichir mon vocabulaire car j'apprends la langue) afin de passer du mandarin à l'anglais.

Toutes les classes de la hiérarchies, y compris les tests, contenant ce genre d'informations possèdent désormais la traduction en anglais juste à côté de l'information afin de faciliter la compréhension.

Avec plus de temps d'amélioration du code, il aurait été enrichissant de poursuivre cette démarche et de l'étendre à l'ensemble des hiérarchies.

## 7 Embellissement de l'aspect global du projet

### 7.1 Ajout de la javadoc et de commentaires

Au début, très peu de classes du projet possédaient une javadoc complète. Certaines classes, quand bien même issues de la hiérarchie principale, n'en possédaient même pas.

À ce jour, toutes les classes de la hiérarchie core ont une javadoc complète. Toutes les méthodes de toutes les classes ont leur javadoc indiquant leur fonctionnement, leur(s) paramètre(s), leur valeur de retour, les exceptions pouvant être levées... Les méthodes surchargées étant définies dans des interfaces ou des classes abstraites avec leur javadoc, il n'est pas nécessaire de leur en refaire une.

Dans la classe Site par exemple, une description détaillée a été faite avec des commentaires pour préciser les différentes parties constituant un site décomposant le plus clairement possible les parties.

### 7.2 Placement optimal des méthodes en fonction de l'utilité et de l'encapsulation

De plus, par pertinence et encapsulation, les méthodes ont été placées comme suit : les privées tout en bas de la classe, les protected ensuite, les publiques tout en haut de la classe pour préserver un certain ordre.

À l'origine, certaines méthodes privées pouvaient se trouver en début de classe. On avait généralement un enchaînement de méthodes publiques protected et private qui alternaient. Témoin le cas de la classe Spider avant modification.

Pour finir, dans la classe UrlUtils, les deux attributs privés se trouvaient perdus au beau milieu du code parmi des méthodes publiques et privées et ont été ramenés en début de classe.

## 8 Bilan et perspectives

En définitive, l'on retient que le projet webmagic, en soi, constitue un code riche et bien construit, bien que présentant plusieurs points se prêtant à d'éventuelles corrections.

À cet effet, cette partie ne fera pas uniquement office de conclusion, mais également d'ouverture sur les améliorations qui n'auront pas pu être traitées dans le temps imparti et que nous tenterions de résoudre selon les solutions qui proposées.

Pour citer la classe CssSelector, nous avons remarqué que la méthode `getValue()` possède un grand nombre d'instructions conditionnelles. Pour palier à cela, il serait judicieux d'abstraire le type `CssSelector` et de redéfinir à chaque fois cette méthode où l'attribut "nom" serait contraint d'avoir une seule valeur possible parmi l'html, le text ou l'outer html d'un élément.

Par ailleurs, il aurait également été intéressant d'améliorer la couverture de tests de toutes les hiérarchies du projet, de sorte à avoir une couverture globale de plus de 75%. Cela aura également permis de mettre en oeuvre d'autres techniques que l'héritage de tests, des design patterns ou des créations de Mocks.

Enfin, le travail de traduction pourrait être étendu à l'ensemble des hiérarchies et chaque classe pourrait être soigneusement commentée et pourvue d'une javadoc complète. Le code déprécié quant à lui ne serait plus appelé nulle part dans le projet car l'objectif serait d'étudier la méthode dépréciée appelée, comprendre son fonctionnement et le reproduire dans une nouvelle méthode qui constituerait une alternative valable.

Ainsi s'achève l'analyse du projet webmagic qui nous aura réellement appris à faire preuve d'autonomie et à mettre en oeuvre de bonnes pratiques de qualité logicielle, à privilégier la maintenabilité et la bonne présentation du code et à faire des tests de ce dernier une priorité.