Sonar fixer

Clement Maxime - Piorun Jordan

November 13, 2015

1 Introduction

Pour de nombreux projets, nous avons recours a Sonar, un logiciel permettant de mesurer la qualité de notre code en continu. Le rapport de Sonar classe les améliorations a apporter en différentes catégories, des erreurs critiques aux simples informations.

Pour notre projet, nous somme parti d'un simple constat : les erreurs sonar sont souvent les mêmes, et un nombre considérable d'entre elles pourrait être corrigées de façon automatique. Cela passe par deux étapes : premièrement il faut repérer le morceau de code qui est à l'origine de l'erreur, puis il faut ensuite le remplacer avec du code équivalent.

pour ces deux besoins, nous avons choisis d'utiliser Spoon, une librairie développée à l'Inria permettant d'analyser et de transformer du code Java.

2 Analyse du code

2.1 La méthodologie

2.1.1 Une première approche

Avant même de s'intéresser à la partie technique de ce projet, nous avons du établir la liste des erreurs que nous voulions corriger.

Sonar classe les erreurs en 5 catégories : bloquantes, critiques, majeures, mineures et informatives. Parmi celles ci, notre attention s'est tout particulièrement portée sur les erreurs majeures. En effet, elles semblent être les plus rependues dans les projets, et peuvent pour certaines d'entre elles rendre un projet difficilement maintenable.

Après cette première réduction du scope initial, nous avons fais le tri entre les alertes que nous pouvons corriger de façon automatique, et celles qui dépendent du contexte. Suite a cette seconde analyse, nous avons dressé le tableau en annexe 1 qui reprend les différentes alertes ainsi que la solution apportée.

2.1.2 Une approche plus pertinente

Suite à cette première approche du problème nous nous sommes rapidement heurté à un problème : comment tester nos résultats ? L'idée initiale était de sélectionner des projets open source, que nous récupérons sur github, afin que sonar puisse effectuer un premier diagnostic. Notre programme s'occupe ensuite de corriger les erreurs que nous avons cibler, et un deuxième diagnostique permet de confirmer que le nombre d'alerte a effectivement baissé.

Cependant, il est difficile de trouver des projets qui contiennent certaines erreurs bien spécifiques, d'autant plus que certaines d'entre elles relèvent de pratiques si mauvaises qu'il est très peu probable d'en croiser. Nous avons donc changé notre raisonnement en choisissant d'abord des projets que nous jugeons fiable, avec une communauté importante et une taille conséquente. Nous l'avons donné à analyser auprès de sonar, et nous avons choisis des alertes pertinentes parmi celles relevées.

Cette seconde approche, basée sur les projets Commons-Lang, Commons-math et Log4j, nous a permis de dresser une nouvelle liste d'erreurs dont nous voulons automatiser la réparation.

2.2 les règles définies

2.3 Correction d'erreur

Afin de détecter les erreurs que nous voulons corriger, nous nous sommes appuyé sur la librairie Spoon, qui nous apporte une solution pour analyser du code Java. Pour chaque alerte, nous avons donc un une classe Processor associée dans laquelle nous décrivons le pattern à rechercher. Lorsque cette fonction trouve du code qui match, nous pouvons commencer sa transformation.

3 Évaluation

3.1 Les test unitaires

couverture de test et pourcentage de mutation (partie courte)

3.2 Impacte de spoon sur le code

Afin de mesurer l'impact de notre programme de résolution d'alerte Sonar, notre première démarche a été de différentier les changements duent à l'exécution du code que nous avons écris, de ceux qui relèvent d'une simple exécution de la librairie spoon. En effet, afin de permettre une manipulation du code java, spoon forme un arbre syntaxique abstrait à partir du code qui lui est envoyé en entrée. Une fois qu'il effectue le processus inverse (arbre syntaxique abstrait vers du code visant à être compilé) il est possible que des éléments parasites soient insérés. Par élément parasite, nous comprenons de la syntaxe qui n'était pas présente précédemment, comme des parenthèses optionnelles, des lignes vides ou encore une indentation.

Suite à cette première analyse, nous avons pu déterminer que ...

3.3 Projets réels

Dans le but de valider notre approche, l'étape suivante était de confronter notre programme à des projets open-source de tailles conséquentes possédant les alertes que nous sommes en mesure de réparer automatiquement. Ces projets sont les suivants : commons-lang, commons-math, log4j. L'annexe 2 montre les résultats obtenus suite à nos modifications. ajouter un comparatif entre projet git et projet spoon sans processor.

A Annexe 1 - Tableaux des correctifs

Noncompliant	Compliant
`if (myClass.compareTo(arg) == -1) {...}`	`if (myClass.compareTo(arg) < 0) {...}`

`class Bar {`
`  public boolean doSomething(){...}`
`}`
`class Foo extends Bar {`
`  public boolean doSomething(){...}`
`}`
	`class Bar {`
	`  public boolean doSomething(){...}`
	`}`
	`class Foo extends Bar {`
	`  @Override`
	`  public boolean doSomething(){...}`
	`}`

`public void a(Map<String,Object> map) {`
`  for (String key : map.keySet()) {`
`  Object value = map.get(key);`
`  ...`
`  }`
`}`
	`public void a(Map<String,Object> map) {`
	`  for (Map.Entry<String,Object> entry :`
	`  map.entrySet()) {`
	`  String key = entry.getKey();`
	`  Object value = entry.getValue();`
	`  ...`
	`  }`
	`}`

`foo.equals(bar.toUpperCase());`	`foo.equalsIgnoreCase(bar);`

`switch (myVariable) {`
`  case 0: // 6 lines till next case`
`  methodCall1("");`
`  methodCall2("");`
`  methodCall3("");`
`  methodCall4("");`
`  break;`
`  case 1:`
`  ...`
`}`
	`switch (myVariable) {`
	`  case 0:`
	`  doSomething()`
	`  break;`
	`  case 1:`
	`  ...`
	`}`
	`...`
	`private void doSomething(){`
	`  methodCall1("");`
	`  methodCall2("");`
	`  methodCall3("");`
	`  methodCall4("");`
	`}`

`public class MyClass {`
`  private int foo = 42;`
 	
`  public int compute(int a) {`
`  return a * 42;`
`  }`
`}`
	`public class MyClass {`
	`  public int compute(int a) {`
	`  return a * 42;`
	`  }`
	`}`

`class A {`
`  private int field;`
`}`
	`class A {`
	`  private int field;`
	
	`  A(int field) {`
	`  this.field = field;`
	`  }`
	`}`

Table 1: Évaluation

	Alerte ID	Alertes	Alertes Fixés	% Alertes Fixés	Test degradé
Commons-lang	ID_override	2			
	ID_switch	23	23	100%	1
	ID_private_field	4			
	ID_equals	17			
	ID_instanceof_try	1			
Commons-math	ID_override	26			
	ID_switch	96			
	ID_equals	17			
Log4j	ID_override	108			
	ID_switch	29			
	ID_equals	26			
	ID_instanceof_try	36			