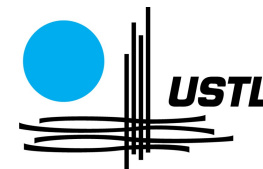


Détection semi-automatique des patrons de mauvaise conception dans les architectures orientées objet

Alban Tiberghien

sous la supervision de :
Naouel Moha et Anne-Françoise Le Meur

Université des Sciences et Technologies de Lille
Master Informatique 1ère année





Contexte (1/2)

- Recherche de qualité dans la production de logiciels.
 - Bénéfice apporté par la programmation orientée objet.
 - Introduction de la notion de patrons de conception par le « Gang of Four ».
 - Apparition des défauts de conception :
 - les **code-smells**,
 - les **anti-patrons**.
- ⇒ Problème d'évolution, de débogage et de maintenance.
- ⇒ Nécessité de détecter les défauts le plus tôt possible.



Contexte (2/2)

- Proposition de thèse de Naouel Moha : développement de l'outil Ptidej.
- Validation de la partie « détection des défauts de conception ».

Motivations:

- Évaluer l'utilisabilité et l'efficacité de Ptidej.
- Évaluez Ptidej par rapport à d'autres outils existants.

Problématique :

Comment Ptidej se positionne par rapport à d'autres outils de détection de défauts ?



Démarche

- I) Définition des différents défauts de conception.
- II) Réalisation d'une expérience « témoin »
 - Conception d'une méthodologie de détection manuelle.
 - Application de la méthodologie sur un projet open-source
- I) Détection automatique : comparaison des outils de détection.



Définition des défauts (1/3)

- Absence de formaliste « officiel ».
- Travail indispensable de classification et de spécification.
- 17 code-smells et 4 anti-patrons spécifiés.



Définition des défauts (2/3)

Exemples de code-smells :

Large Class (famille des *Bloaters*) :

- trop de responsabilités,
- trop d'attributs et/ou de méthodes.

Data Class (famille des *Dispensables*) :

- uniquement des attributs et leurs accesseurs.
- aucune fonctionnalité particulière.



Définition des défauts (3/3)

Exemple d'anti-patterns :

Le Blob, classe qui :

- centralise le traitement.
- dispose de beaucoup d'attributs et de méthodes.
- dépend d'autres classes où sont stockées les données.
- est issue d'une évolution incrémentale d'un projet.

Heuristique :

Les anti-patterns sont basés sur l'existence de code-smells.



Méthodologie de détection manuelle (1/3)

Contraintes :

- Respect de mes définitions.
- Analyse manuelle systématique et non ambiguë.
- Représentation sous forme d'arbres de décision.
- Utilisation de l'EDI Eclipse seulement.

- Élaboration de 8 méthodologies couvrant les 17 code-smells et les 4 anti-patterns.

Méthodologie de détection manuelle (2/3)

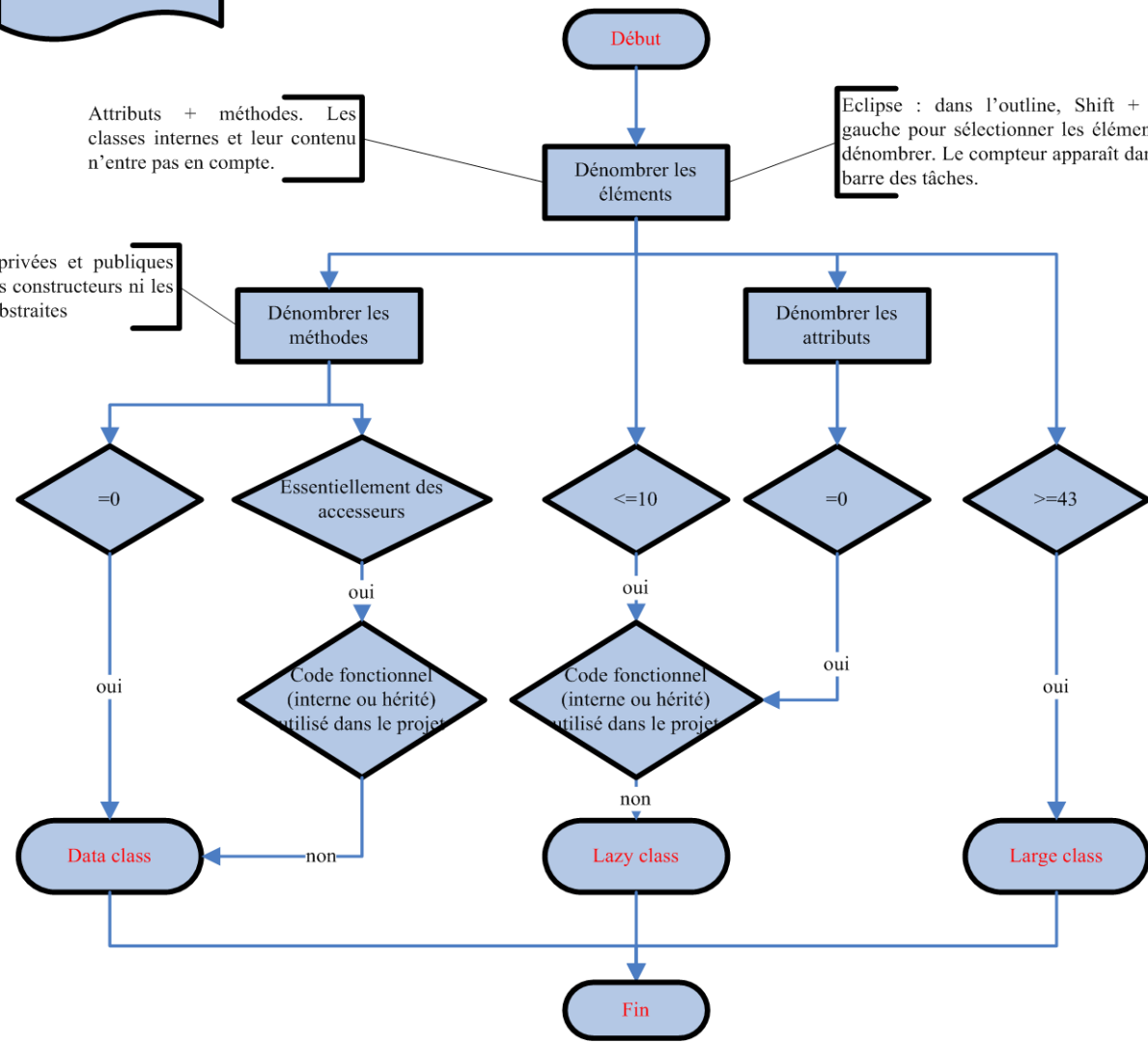
Inspection des méthodes
& attributs

Attributs + méthodes. Les
classes internes et leur contenu
n'entre pas en compte.

Eclipse : dans l'outline, Shift + clic
gauche pour sélectionner les éléments à
dénombrer. Le compteur apparaît dans la
barre des tâches.

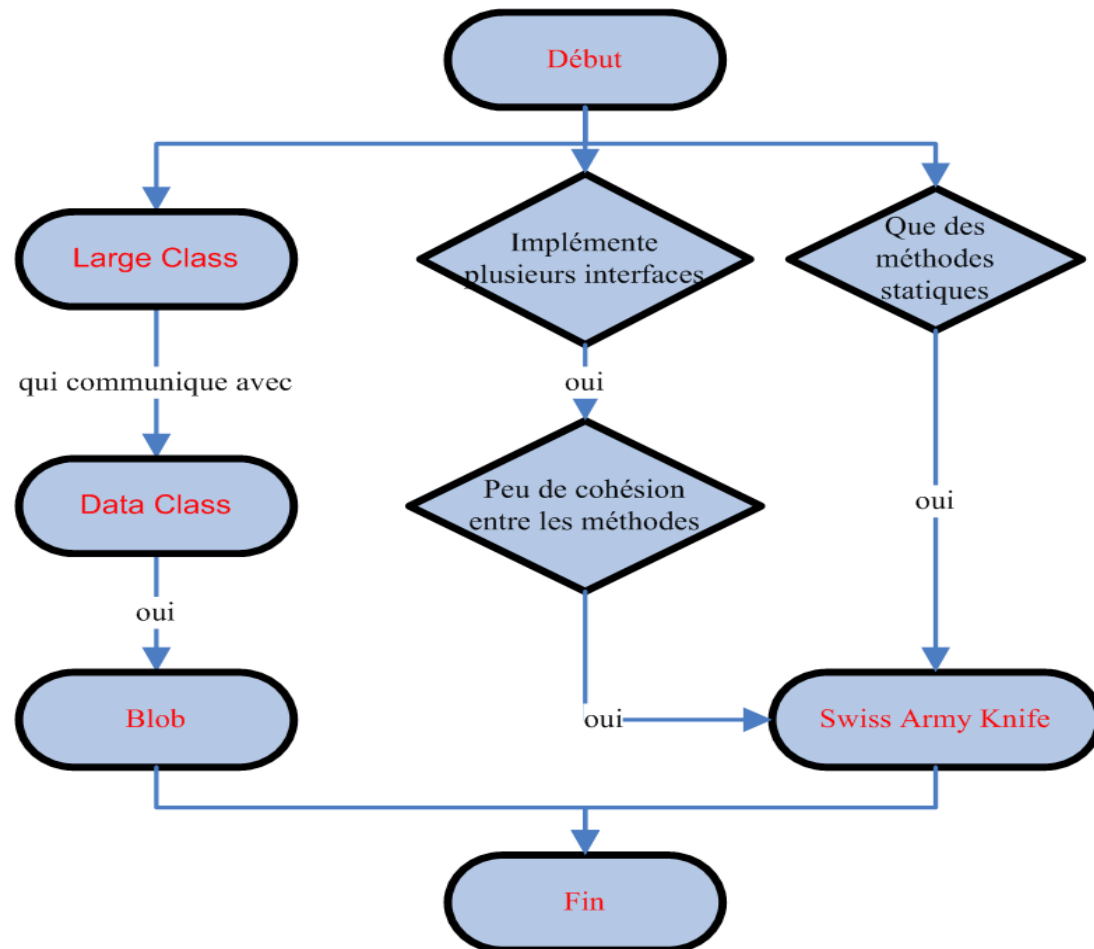
Méthodes privées et publiques
mais pas les constructeurs ni les
méthodes abstraites

Diagramme de détection manuelle
pour les code-smells Data Class, Lazy
Class et Large Class



Méthodologie de détection manuelle (3/3)

Diagramme de détection manuelle pour les anti-patterns Blob et Swiss Army Knife





Détection manuelle et constitution d'un "témoin"(1/2)

- Détection manuelle faite sur le projet GanttProject

Pourquoi GanttProject ?

- Taille significative : 240 classes et 22000 lignes de codes.
- Projet d'étudiants.
- Devenu open-source.
- Devenu un travail collaboratif.
- Utilisation de générateur de code pour sa conception.

- Fortes chances de trouver des défauts de conception.

Détection manuelle et constitution d'un "témoin" (2/2)

Résultats de l'analyse manuelle

Nom du code-smell	Nombre de défauts détectés dans GanttProject
Lazy Class	63 (dont 24 dans des classes internes)
→ Data Class	75 (dont 23 dans des classes internes)
Long Parameter List	54 (dont 10 dans des classes internes)
Not enough information hiding	54 (dont 7 dans des classes internes)
Long Method	45 (dont 4 dans des classes internes)
Refused Bequest	23 (dont 4 dans des classes internes)
Temporary Field	18
Switch Statement	11 (dont 4 dans des classes internes)
Poor Usage of Interfaces	10
Data Clumps	10 (dont 1 dans une classe interne)
→ Large Class	9
Speculative Generality	8
Message Chains	8
Aternative Classes With Different Interfaces	7 (dont 2 dans des classes internes)
Poor Usage of Abstract Classes	7
Primitive Obsession	2
Too much information hiding	2
406 code-smells détectés manuellement au total (dont 79 dans des classes internes)	

Nom de l'anti-patron	Nombre de défauts détectés dans GanttProject
Functional Decomposition	18
Spaghetti Code	12
Swiss Army Knife	8
→ Blob	8
46 anti-patrons détectés manuellement au total	



Détection automatique et comparaison des outils (1/6)

Ptidej vs iPlasma

Travaux préliminaires :

- la prise en main
- la détection automatique



Détection automatique et comparaison des outils (2/6)

Ptidej vs iPlasma **Prise en main des outils.**

- temps nécessaire identique pour les 2 outils.

Pour Ptidej

- règles utilisables via des onglets et des boutons
- éditeur textuel de règles de détection.
- API Ptidej pour définir les règles.

Pour iPlasma

- raffinements successifs par l'intermédiaire de menus
- enchaînement d'algorithmes de regroupement et de filtrage
- éditeur de filtres



Détection automatique et comparaison des outils (3/6)

Ptidej vs iPlasma Détection automatique

Défauts	Méthodes de détection	
	iPlasma	PtiDej
The Bloaters		
Large Class (méthode par défaut)	group "class group" + filter "class model" + filter* "not inner class" + filter* "Brain Class"	onglet "Code Smells" + bouton "LargeClassDetection"
The Dispensables		
Data Class	group "class group" + filter "class model" + filter* "not inner class" + filter "Data Class"	onglet "Code Smells" + bouton "DataClass"
Les Anti-patrons		
The Blob	group "class group" + filter "class model" + filter* "not inner class" + filter "God Class"	onglet "Anti Patterns" + checkbox "Blob"



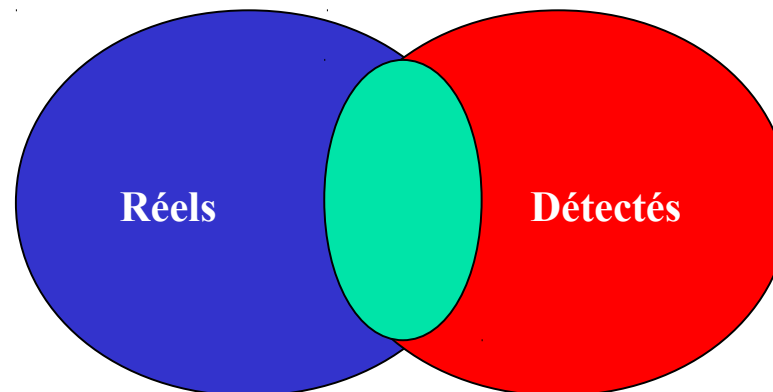
Détection automatique et comparaison des outils (4/6)

Ptidej vs iPlasma **Analyse des résultats**

- Calcul de la précision et du rappel de chaque défaut détecté et pour chaque outils.

$$\text{Précision} = \frac{|\{\text{défauts détectés manuellement}\} \cap \{\text{défauts détectés par l'outil}\}|}{|\{\text{défauts détectés par l'outil}\}|}$$

$$\text{Rappel} = \frac{|\{\text{défauts détectés manuellement}\} \cap \{\text{défauts détectés par l'outil}\}|}{|\{\text{défauts détectés manuellement}\}|}$$





Détection automatique et comparaison des outils (5/6)

Ptidej vs iPlasma **Comparaison**

Différences entre la détection manuelle et la détection automatique.

- La détection manuelle est la plus efficace.
 - Problème du temps d'analyse
 - Méthodologie algorithmiquement programmable
- La détection automatique est une source sûre.
 - Tout n'est pas implémentable.
 - Résultat satisfaisant
 - Même l'homme fait des erreurs.



Détection automatique et comparaison des outils (6/6)

Ptidej vs iPlasma **Comparaison**

Différences entre Ptidej et iPlasma.

Nom du code-smell	Réels défauts	iPlasma			Ptidej		
		Défauts détectés	Précision	Rappel	Défauts détectés	Précision	Rappel
The Bloaters							
Large Class (méthode par défaut)	9	0	0/0 (0%)	0/9 (0%)	0	0/0 (0%)	0/9 (0%)
Large Class (méthode personnelle)	9	∅			17	9/17 (52,94%)	9/9 (100%)
The Dispensables							
Data Class	52	24	18/24 (75%)	18/52 (34,62%)	55	41/55 (74,55%)	41/52 (78,85%)
The Bloated							
Nom de l'anti-patterns	Réels défauts	Défauts détectés	Précision	Rappel	Défauts détectés	Précision	Rappel
The Blob	8	11	8/11 (72,72%)	8/8 (100%)	10	7/10 (70%)	7/8 (87,5%)



Conclusion

Démarche typiquement « recherche » :

- Définition des défauts de conception
 - Mise au point d'une expérience « témoin »
 - Comparaison des outils
-
- Ptidej et iPlasma sembleraient être complémentaires.
mais ...
 - Ptidej n'est qu'au début de son développement.