Chapitre 4

Métriques Orientées Objets

1. Introduction

La majorité des métriques proposées sont destinées au paradigme procédural. Elles ne s'appliquent pas au paradigme objet. La mesure logiciel pour le logiciel non orienté objet utilise le graphe de flux de contrôle comme une abstraction du logiciel. Cependant, le graphe de flux de contrôle n'apparait pas utile pour une abstraction du logiciel orienté objet. Le problème intuitive en appliquant les métriques logicielles traditionnelles est que la complexité d'un logiciel OO n'apparait pas dans la structure de contrôle.

Par conséquent, Des métriques spécifique orientée objets ont été proposées ces dernières années. Les plus connues sont celles de ***Chidamber et Kemerer.***

1.1 Abstractions OO

Dans le logiciel OO, les méthodes sont petites et le nombre de décisions dans une méthode est souvent petit. La grande complexité réside dans le modèle d'appel entre les méthodes.Les abstractions communs utilisé dans l'orienté Objet sont les diagrammes UML.

1.2 Éléments de base

- **Niveau classes**

• C : ensemble de toutes les classes du système
• Parents(c) : ensemble des superclasses directes de c
• Children(c) : ensemble des sous-classes directes de c
• Ancestors(c) : ensemble des superclasses de c
• Descendents(c) : ensemble des sous-classes de c

- **Niveau méthodes**

• M(C) : ensemble de toutes les méthodes du système
• M(c) : ensemble de toutes les méthodes d'une classe c
• M_D(c) : ensemble des méthodes déclarées dans la classe c
• M_I(c) : ensemble des méthodes implantées dans la classe c

$$M_D(c) \cup M_I(c) = M(c)$$
$$M_D(c) \cap M_I(c) = \varnothing$$

- $M_{INH}(c)$: ensemble des méthodes héritées de c
- $M_{OVR}(c)$: ensemble des méthodes redéfinies de c
- $M_{NEW}(c)$: ensemble des méthodes nouvellement créées de c

$$M_{INH}(c) \cup M_{OVR}(c) \cup M_{NEW}(c) = M(c)$$

- $M_{PUB}(c)$: ensemble des méthodes publiques de c
- $M_{NPUB}(c)$: ensemble des méthodes non publiques de c

$$M_{PUB}(c) \cup M_{NPUB}(c) = M(c)$$
$$M_{PUB}(c) \cap M_{NPUB}(c) = \varnothing$$

- SIM(m) : ensemble des méthodes statiquement invoquées dans m
- NSI(m, m') : nombre d'invocations statiques de m' par m
- PIM(m) : ensemble des méthodes invoquées dans m de manière polymorphique
- NPI(m, m') : nombre d'invocations polymorphiques de m' par m

- **Niveau d'attributs**

- $A_I(c)$: ensemble des attributs hérités dans la classe c
- $A_D(c)$: ensemble des attributs définis dans la classe c
- A(c) : ensemble des attributs de la classe c

$$A(c) = A_D(c) \cup A_I(c)$$

- AR(m) : ensemble des attributs référencés par une méthode m

2. Metriques de *Chidamber et Kemerer*

Ce sont les metriques les plus connues.

• Métrique1: Depth of Inheritance Tree (DIT)

– Définition

- Profondeur de la classe dans l'arbre d'héritage
- En cas d'héritage multiple, DIT est égale à la profondeur maximale depuis la classe racine jusqu'à la classe mesurée

$$DIT(c) = \begin{cases} 1 + \max\limits_{d \in Parents(c)} DIT(d) \\ 0 \; if \; Parents(c) = \emptyset \end{cases}$$

– Considérations

- Plus une classe se situe profondément dans l'arbre d'héritage, plus le nombre de méthodes et de comportements dont elle hérite est grand, ceci rend son comportement d'autant moins prévisible

• Plus un arbre est grand, plus la conception est complexe
• Plus une classe se situe profond dans l'arbre d'héritage, plus il est probable qu'elle réutilise des méthodes héritées

Métrique 2 : Number Of Children (NOC)

– Définition

• Nombre de descendants immédiat de la classe dans la hiérarchie de classe

$$NOC(c) = |Children(c)|$$

– Considérations

• Plus le nombre d'enfants est élevé plus la réutilisation effective est importante, car l'héritage est une forme de réutilisation

• Plus le nombre d'enfants est élevé, plus le risque d'une utilisation inadéquate du mécanisme d'héritage est grand, la classe de base peut être mal abstraite

• Le nombre d'enfants donne une idée de l'influence potentielle d'une classe sur la conception, si une classe a un nombre important d'enfants, elle requiert d'avantage d'efforts de test

Métrique 3 :Weighted Methods per Class (WMC)

– Définition

• Somme pondérée des complexités des méthodes d'une classe, si toutes les méthodes d'une classe sont de complexité 1, correspond au nombre de méthodes

$$WMC(c) = \sum_{m \in M_I(c)} comp(m)$$

$$WMC(c) = |M_I(c)| \; si \; \forall m \in M_I(c), comp(m) = 1$$

Cas de la complexité cyclomatique

$$WMC(c) = \sum_{m \in M_I(c)} V(m)$$

– Considérations

• Le nombre et la complexité des méthodes d'une classe est un bon indicateur de l'effort pour la développer et la maintenir
• Plus on a de méthodes dans une classe, plus l'impact pourra être grand sur les classes enfants par héritage
• Les classes qui ont un grand nombre de méthodes complexes sont en général plus spécifiques à un programme et donc moins réutilisables

Métrique 4 : Coupling Between Objects (CBO)

– Définition

• Nombre d'autres classes auxquelles une classe est couplée, deux classes sont dites couplées si une méthode de l'une utilise une méthode (ou un attribut L) de l'autre

$$CBO(c) = |\{d \in C - \{c\} \mid uses(c,d) \vee uses(d,c)\}|$$

$$uses(c,d) \Leftrightarrow \left(\exists m \in M_I(c), \exists m' \in M_I(d), m' \in PIM(m)\right) \vee \left(\exists m \in M_I(c), \exists a \in A_I(d), a \in AR(m)\right)$$

– Considérations

• Un couplage excessif entre classes se fait au détriment de la modularité et empêche la réutilisation

• Le couplage devrait être minimal pour promouvoir l'encapsulation : plus une classe est couplée à d'autres classes, plus une modification de cette classe influence les autres

• Une mesure du couplage est importante pour déterminer l'effort de test des diverses classes, l'effort de test d'une classe devrait être d'autant plus élevé que le CBO de cette classe est élevé

Métrique 5 : Response For a Class (RFC)

– Définition

• RFC vaut la cardinalité de l'ensemble de réponse d'une classe

• L'ensemble de réponse d'une classe est l'ensemble des méthodes qui peuvent être directement appelée lors de l'exécution de n'importe quelle méthode de cette classe (= réponse à un message)

$$RFC(c) = RFC_1(c)$$

$$RFC_\alpha(c) = \left|\bigcup_{i=0}^{\alpha} R_i(c)\right| \text{ avec}$$

$$R_0(c) = M(c) \text{ et } R_{i+1}(c) = \bigcup_{m \in R_i(c)} PIM(m)$$

– Considérations

• Si un grand nombre de méthodes peuvent être appelées en réponse à un message, le test et le débogage sont plus compliqués car ils requièrent une meilleure compréhension

• La complexité d'une classe est d'autant plus élevée qu'elle peut appeler un grand nombre de méthodes

Métrique 6 : Lack of COhesion in Methods (LCOM)

– Définition

• LCOM compte le nombre de paires de méthodes qui n'accèdent pas aux même attributs moins le nombre de paires de méthodes qui accèdent aux même attributs (= 0 si négative)

• LCOM1 compte le nombre de paires de méthodes qui n'accèdent pas aux mêmes attributs

$$LCOM1(c) = |P|$$

avec

$$P = \{\{m_1, m_2\} \mid m_1, m_2 \in M_I(c) \wedge m_1 \neq m_2 \wedge AR(m_1) \cap AR(m_2) \cap A_I(c) = \emptyset\}$$

LCOM2 compte le nombre de paires de méthodes qui n'accèdent pas aux mêmes attributs diminué du nombre de paires de méthodes qui accèdent aux mêmes attributs (= 0 si la différence est négative)

$$LCOM2(c) = \begin{cases} |P| - |Q| \; si \; |P| > |Q| \\ 0 \qquad\qquad sinon \end{cases}$$

avec

$$Q = \{\{m_1, m_2\} \mid m_1, m_2 \in M_I(c) \wedge m_1 \neq m_2 \wedge AR(m_1) \cap AR(m_2) \cap A_I(c) \neq \emptyset\}$$

– Considérations

• Une forte cohésion est signe d'une bonne encapsulation

• Une faible cohésion suggère qu'une classe doit être décomposée en deux ou plusieurs classes

• Toute mesure de disparité entre les méthodes aide à trouver les erreurs de conception

• Une faible cohésion augmente la complexité et la probabilité d'existence d'erreurs dans le processus de développement

Remarques sur les métriques de Chidamber

Métriques Objet de Chidamber

- Ensemble de métriques (Metric Suite for Object Oriented Design)
  - Evaluation des classes d'un système
  - La plupart des métriques sont calculées classe par classe
  - Le passage au global n'est pas clair, une moyenne n'étant pas très satisfaisante

3. Métriques de Henry-Kafura

Flux d'informations d'Henry-Kafura

Mesurer la complexité des modules d'un programme en fonction des liens qu'ils entretiennent

- On utilise pour chaque module i :
- Le nombre de flux d'information entrant noté in_i
- Le nombre de flux d'information sortant noté out_i
- Le poids du module noté $poids_i$ calculé en fonction de son nombre de LOC et de sa complexité

$$HK_i = poids_i \times (out_i \times in_i)^2$$

- La mesure totale HK correspond à la somme des HK_i

Exemple

- A partir des in_i et out_i ci-dessous, calcul des métriques HK en supposant que le poids de chaque module vaut 1

Module	a	b	c	d	e	f	g	h
in_i	4	3	1	5	2	5	6	1
out_i	3	3	4	3	4	4	2	6
HK_i	144	81	16	225	64	400	144	36

$$HK = 1110$$

4. Métriques MOOD

Métriques MOOD

- Ensemble de métriques pour mesuer les attributs des propriétés suivantes :
  - Encapsulation
  - Héritage
  - Couplage
  - Polymorphisme

Encapsulation

- MHF : Method Hiding Factor (10-30%)

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}$$

avec

- $M_d(C_i)$ le nombre de méthodes déclarées dans une classe C_i
- $M_h(C_i)$ le nombre de méthodes cachées
- TC le nombre total de classes.
- AHF : Attribute Hiding Factor (70-100%)

$$\textbf{AHF} = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}$$

avec

- $A_d(C_i)$ le nombre d'attributs déclarés dans une classe C_i
- $A_h(C_i)$ le nombre d'atributs cachés

5. Afferent and Efferent coupling

C'est une mesure de couplage définit au niveau de deux pachages. Ils indiquent le couplage d'un package aux classes dans un autre package.

- Afferent coupling : C_a
  C'est le nombre de classes appartenant à d'autres packages qui dependent des classes appartenant au package sujet. Uniquement les relations entre classes sont comptés (telles que association, heritage).
  C_a est réellement le Fan-out d'un package.

- Efferent coupling : C_e

C'est le nombre de classes dans d'autres packages telles que les classes du package sujet dependent d'elles, par des relations entre classes.
C_e est réellement le Fan-in d'un package.

Martin considère qu'un couplage efferent élevé rends le package instablevue qu'il depent sur beaucoup de classes importées. Il définit la métrique d'instabilité I :

$$I = C_e / (C_e + C_a)$$

Facteurs d'héritage

- MIF : Method Inheritance Factor (65-80%)

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

avec

- $M_i(C_i)$ le nombre de méthodes héritées (et non surchargées) de C_i
- $M_a(C_i)$ le nombre de méthodes qui peuvent être appelées depuis la classe i
- AIF : Attribute Inheritance Factor (50-60%)

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

avec

- $A_i(C_i)$ le nombre d'attributs hérités de C_i
- $A_a(C_i)$ le nombre d'attributs auxquels C_i peut accéder

Facteur de couplage

- CF : Coupling Factor (5-30%)
  - Mesure le couplage entre les classes sans prendre en compte celui dû à l'héritage

$$CF = \frac{\sum_{i=1}^{TC}\sum_{j=1}^{TC} client(C_i, C_j)}{TC^2 - TC}$$

avec

- $client(C_i, C_j) = 1$ si la classe i a une relation avec la classe j, et 0 sinon
- Les relations d'héritage ne sont pas prises en compte dans les relations

Facteur de polymorphisme

- PF : Polymorphism Factor (3-10%)
  - Mesure le potentiel de polymorphisme d'un système

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} M_n(C_i) \times DC(C_i)}$$

avec

- $M_o(C_i)$ le nombre de méthodes surchargées dans la classe i
- $M_n(C_i)$ le nombre de nouvelles méthodes dans la classe i
- $DC(C_i)$ le nombre de descendants de la classe i