Test statique

Jean-Marie Mottu
Le Traon – Baudry - Sunye

Test statique

- Ne requiert pas l ’exécution du logiciel sous-test sur des données réelles
- Plusieurs approches
  - inspection de code (lisibilité du code, spécifications complètes...)
  - mesures statiques (couplage, nombre d’imbrications...)

Inspection de code

- Réunions de 4 personnes environ pour inspecter le code
  - 1 modérateur, le programmeur, le concepteur et 1 inspecteur
- Déroulement
  - le programmeur lit et explique son programme
  - le concepteur et l 'inspecteur apportent leur expertise
  - les fautes sont listées
  - (pas corrigées-> à la charge du programmeur)

Inspection de code

- Efficacité : plus de 50 % de l ’ensemble des fautes d ’un projet sont détectées lors des inspections si il y en a (en moyenne plus de 75%)
- Défaut : mise en place lourde, nécessité de lien transversaux entre équipes, risques de tension…tâche plutôt fastidieuse

Test statique

- Règles
  - être méthodique (cf. transparents suivants)
  - un critère : le programme peut-il être repris par quelqu'un qui ne l'a pas fait
  - un second critère : les algorithmes/l'architecture de contrôle apparaît-elle clairement ?
  - décortiquer chaque algo et noter toute redondance curieuse (coller) et toute discontinuité lorsqu'il y a symétrie (ce qui peut révéler une modif incomplète du programme)

Test statique

- Exemple: vérification de la clarté
  - R1 :Détermination des paramètres globaux et de leur impact sur les fonctions propres

```
program recherche_tricho;
uses crt;
const
   max_elt = 50;
   choix1  =  1;
   choix2  =  2;
   fin     =  3;
type
   Tliste  = array[1..max_elt] of integer;
var
   liste              : Tliste;
   taille, choix, val : integer;
   complex            : integer;
```

- But du programme non exprimé
- Manque de commentaires
- Identificateurs non explicites

Test statique

Exemple: vérification de la clarté

R2 : Existence d'un entête clair pour chaque fonction

```
{-----------------------------------------------------------------
  Recherche recursive d'une valeur dans une liste triee
-----------------------------------------------------------------}
```

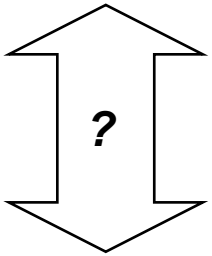
Test statique: pour chaque fonction

Commentaires minimum

manque
- le nom de la fonction
- dépendance avec autres variables/fonctions

{ parametres d'entree : liste L
la valeur recherchee
indice gauche
indice droit
resultat : complexite de la recherche }

Interface Spécifiée



Interface implantée

```
function rech_rec(L : Tliste ; val, g, d : integer) : integer ;
```

Test statique

Quézako ?

```
var i, pt, dt : integer;

 begin
   affiche(L, g, d);
   if g<d
      then
         begin
            pt :=g+(d-g) div 3;
            if val > L[pt]
               then
                  begin
                     dt := (pt+1+d) div 2;
                     if val > L[pt]
                        then rech_rec:=2+rech_rec(L, val, dt+1, d)
                        else rech_rec:=2+rech_rec(L, val, pt+1, dt)
                  end
               else rech_rec:=1+rech_rec(L, val, g, pt)
         end
      else rech_rec:=0
end; { rech_rec }
```

Action non spécifiée

Répétition ?

Test statique

- Métriques
- Analyse d'anomalies
- Preuve
- Exécution symbolique
- Simulation de modèle

Métriques

Métriques au niveau système

- Coupling Factor (= (total number of couplings) / (n^2-n) ; with n = number of defined classes).
  - A high COF points to a high complexity of the system.
- ANM - Average Number of Methods per Class
- ANA - Average Number of Attributes per Class
- ANP - Average Number of Parameter per Method

Métriques au niveau classe (1/2)

- DIT - Depth of Inheritance Tree: Is the maximum length of the way from the class to the root. Classes without a parent class have a DIT of 0.
- NOC - Number of Children: Number of direct successor classes.
- WMC - Weighted Methods per Class: number of methods of the considered class.
- WAC - Weighted Attributes per Class: number of attributes of the considered class.
- CBO - Coupling between Object Classes: Number of classes, which are coupled with this class. Classes are coupled, if one class uses methods or attributes of the other class.

Métriques au niveau classe (2/2)

- PIM - Number of Public Methods
- NMI - Number of Methods inherited: Number of methods of the direct parent class
- NAI - Number of Attributes inherited Number of attributes of the direct parent class
- NMO - Number of Methods overwritten
- RFC - Response for a class Number of methods used by the class plus the methods of the class. Is the highest possible number of methods, which can be invoked by a message to this class.
- LOC - Lines of Code

Métriques au niveau méthode

- NOP - Number of Parameter
- LOC - Lines of Code
  - Loc, Kloc, Mloc
- CC - Complexité cyclomatique
- NPath
- Comment rate
  - Ratio #Lines of comments/Loc
  - Degree of code reusability
  - No comment = not reusable

Code Commenting Styles

Code: `int x = (really complicated expression);`

The Stater of the Obvious
```
// Computes x.
```

The More Precise Stater of Obvious
```
// Computes the value of x.
```

The Jedi
```
// This is not the line you are looking for.
```

The Wizard
```
// This is magic. Don't edit unless
// you are Gandalf.
```

The Edison
```
// I didn't fail, I just found 10,000 wrong
// explanations for how this line works.
```

The Kubrick
```
// This line was given to us by a
// monolith from outer space.
```

The Naive Planner
```
// To be replaced with new
// implementation in 2007.
```

The Naive Coder
```
// This logic never needs to be modified,
// so we don't need to document it so well
```

The Naive Interpreter of Good Practices
```
// Consider this line to be a black box.
// Encapsulation is a good practice, right?
```

The Forward Thinker
```
// Computes the value of x
// for the next step.
```

The Advisor
```
// I wouldn't try to understand, much
// less modify, this line if I were you.
```

The Terminator
```
// Hasta la refactoring, baby...
```

The Carelessly Vague
```
// Does stuff with the input.
```

The Eloquently Vague
```
// Performs the required computations
// and produces the proper result.
```

The Fermat
```
// I'd document this but it wouldn't fit
//  in 80 columns.
```

The Carpe Diem
```
// Seems to work for now,
// let's not worry until it breaks.
```

The Zen
```
// To find happiness we must be open
// to new beliefs. I choose to believe this
// line works.
```

The Machine
```
// Humans cause bugs. So this line was
// designed to maximize machine
// performance and minimize human
// understanding.
```

The Lazy
```
// see docs
```

The Honest Lazy
```
// see docs (TODO: write docs)
```

The Liar
```
// To be documented later.
```

The Museum Curator
```
// Preserved intact from old system.
```

The Chooser
```
// We had to choose between meeting the
// deadline or writing organized code.
// (edit: we failed both)
```

The Zork Adventurer
```
// Don't look at this line too long.
// You are likely to be eaten by a grue.
```

The Purely Empirical
```
// This line produces correct results
// for all tests, so it is, by definition,
// correct.
```

The Time Traveler
```
// Future me: please forgive me for
// this code. After you forgive me, can you
// also please fix the code?
```

The Knight Guardian
```
// You had a choice: looking at this code,
// or living in blissful ignorance.
// You chose poorly.
```

Metrics on Inheritance Level

- MIF - Method Inheritance Factor: Relation of inherited methods to total number of methods.
- AIF - Attribute Inheritance Factor: Relation of inherited attributes to total number of attributes.

Mauvaises pratiques

Mauvaises pratiques

- Méthode outil.
- Code dupliqué.
- Classe donnée.
- Classe trop grande.
- Méthode trop longue.
- Paramètres rattachés.
- Excès de commentaires.
- Conditionnels imbriqués.
- Désir des attributs d'autrui.
- Généralisation spéculative.
- Nombre excessif de paramètres.
- Hiérarchies parallèles d'héritage.
- Attributs d'instance peu utilisées.
- Même nom, différentes significations.
- Nombre excessif de méthodes privées (ou protégées).

Méthodes outil

- Ce sont des méthodes sans référence (explicite ou implicite) à this (ou self, current, etc.).
- Souvent, peuvent être placées ailleurs: vérifier les paramètres.
- Doivent au moins être marquées (p.ex. «utility»).

Code dupliqué

- Tout faire une fois et seulement une fois.
- Le code dupliqué rend complexe la compréhension du code.
- Le code dupliqué est plus difficile à maintenir:
  - Tout changement doit être dupliqué.
  - Celui qui maintient le code doit le savoir.

Classes-données

- Classes contenant seulement des attributs, leurs accesseurs et leurs modificateurs (aka, Data Transfer Objects - DTO).
- Refactorings: Move Method.

Classes Dieu

- Souvent, l'excès de méthodes ou d'attributs cache une duplication de code.
- Une fois de plus, aucune métrique n'est exacte pour tous les cas.
- Trouver des ensembles disparates de méthodes et attributs.

Méthode trop longue

- Plus une méthode est longue, plus il est difficile de comprendre son fonctionnement.
- La méthode est la plus petite unité de surcharge.
- Aucune métrique n'est exact pour tous les cas.
- Le corps d'une méthode doit être homogène (au même niveau d'abstraction).

Paramètres rattachés

- Cachent souvent un manque d'abstraction.
- Par exemple: Point.
- Une fois une nouvelle classe créée, il est souvent facile d'y ajouter un comportement spécifique.

Excès de commentaires

- Un commentaire doit décrire une intention et non expliquer une action.
- L'excès de commentaires inutiles surcharge le code et le rend illisible.
- Donner un nom plus significatif à la méthode, extraire le code commenté et en créer une nouvelle méthode. Introduire des assertions.

Conditionnels imbriqués

- Symptôme d'une méthode mal placée.
  - Plutôt que faire un choix, permettre à la surcharge de le faire.
  - Les nouveaux cas ne demandent pas que le code existant soit changé (le but ultime de la conception).

Désir des attributs d'autrui

- Une méthode appelle les accesseurs d'une autre classe
  - déplacer la méthode.
- Parfois, seulement une partie de la méthode le fait
  - extraire cette partie et la déplacer.

Généralisation spéculative

- Classes trop génériques en prévision d'un futur besoin.

Nombre excessif de paramètres

- Toutes les données demandées par une méthode sont passées en paramètre.

Hiérarchies parallèles d'héritage

- L'ajout d'un classe à une hiérarchie implique l'ajout d'une classe à l'autre hiérarchie.
- Souvent, les classes des deux hiérarchies partagent le même préfixe.
- Exemples: Transaction et Comptes (assurances, bancaire, etc.)
  - Un Compte n'accepte que des Transactions de même type.

Attributs d'instance peu utilisés

- Si certaines instances les utilisent, et d'autres non: créer des sous-classes.
- Si utilisés seulement lors d'une opération particulière, considérer la création d'un objet-opérateur (voir patron de conception Commande).

Même nom, différentes significations

- Conduit à une mauvaise interprétation du code:
  - Identificateurs identiques: réutiliser une variable locale à une autre fin est souvent signe d'une méthode trop longue.
  - Vocabulaire surchargé (in english): Order, Serialize, Thread, etc.

Nombre excessif de méthodes privées (ou protégées)

- Les méthodes doivent être publiques, sauf si elles violent l'invariant de classe.
- Les méthodes publiques peuvent être testées plus facilement.