Qualités du logiciel

Philippe Collet

2007-2008

Qualités du logiciel

- Il faut bien distinguer
  - Les qualités utiles à l'utilisateur, donc *a priori* souhaitées par le client
    - Phases d'exploitation
  - Les qualités utiles au développeur
    - Phases de construction et de maintenance

Qualités pour l'utilisateur

- Fiabilité = Validité + Robustesse
  - Validité (Efficacité) = correction, exactitude
    - *Efficacité : qualité d'une chose ou d'une personne qui donne le résultat escompté*
    - ☞ Assurer exactement les fonctions attendues, définies dans le cahier des charges et la spécification, en supposant son environnement fiable
    - ☞ Adéquation aux besoins
  - Robustesse: faire tout ce qu'il est utile et possible de faire en cas de défaillance: pannes matérielles, erreurs humaines ou logicielles, malveillances...

Qualités pour l'utilisateur (suite)

- Performance (parfois appelée efficacité)
  - Utiliser de manière optimale les ressources matérielles : temps d'utilisation des processeurs, place en mémoire, précision...
- Convivialité
  - Réaliser tout ce qui est utile à l'utilisateur, de manière **simple, ergonomique, agréable** (documentation, aide contextuelle...

Qualités pour le développeur

- Documentation
  - Tout ce qu'il faut, rien que ce qu'il faut, là où il faut, quand il faut, correcte et adaptée au lecteur : **crucial !**
- Modularité = Fonctionnalité + Interchangeabilité + Évolutivité + Réutilisabilité
  - Fonctionnalité
    - Localiser un phénomène unique, facile à comprendre et à spécifier

Qualités pour le développeur (suite)

- Interchangeabilité
  - Pouvoir substituer une variante d'implémentation sans conséquence fonctionnelle (et souvent non-fonctionnelle) sur les autres parties
- Évolutivité
  - Facilité avec laquelle un logiciel peut être adapté à un changement ou une extension de sa spécification
- Réutilisabilité
  - Aptitude à être réutilisé, en tout ou en partie, tel que ou par adaptation, dans un autre contexte : autre application, machine, système...

Qualités pour l'entité de développement

- Client satisfait *(est-ce possible ?)*
- Coût minimum de développement
  - Nombre de développeurs
  - Formation des développeurs
  - Nombre de jours de réalisation
  - Environnement
  - Réutilisation maximale

Génie logiciel : le défi

- Contradictions apparentes
  - Qualités vs coût du logiciel
  - Qualités pour l'utilisateur vs qualités pour le développeur
  - Contrôler vs produire
- Conséquences
  - ☞ Chercher sans cesse le meilleur compromis
  - ☞ Amortir les coûts
    - Premier exemplaire de composant coûteux à produire ou à acheter, puis amortissement...

Objectifs de qualité

- Réduire le nombre d'erreurs résiduelles
- Maîtriser coût et durée du développement
- sans nuire à la créativité et à l'innovation

- Adéquation aux besoins
- Efficacité temps/espace
- Fiabilité
- Testabilité, Traçabilité
- Adaptabilité
- Maintenabilité
- Convivialité (interface et documentation)

☞ ***doivent rejoindre les objectifs de productivité***

Analyse de performances

- Motivations
- Principes
- Gprof
- Lire et interpréter les résultats
- Kprof

Motivations

- Performance ?
  - Espace & Temps
  - *We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.*

    Donald E. Knuth
- La plupart des programmes suivent la règle des 80/20
  - Ils exécutent 20 % du code pendant 80 % du temps

☞ Combien de temps passé et dans quelles parties du code
- Quelles fonctions sont souvent utilisées
- Quelles fonctions prennent le plus de temps CPU
- Quelle partie est peut-être mal conçue...

Mesure globale du temps

- Utilisation de la commande UNIX : `time`
  - fournit le temps utilisateur, le temps système, et l'elapsed time
  - CPU time = Temps utilisateur + temps système
  - **elapsed time** : temps réel de l'exécution de la commande
  - **user time** : temps CPU utilisé par le programme utilisateur
  - **sys time** : temps utilisé par le système pour gérer l'exécution du job
- Propriétés
  - Résolution de l'ordre de 10ms
  - Un "system_time" disproportionné peut indiquer un dysfonctionnement
    - Nombre important de floatig-point exceptions, de "page faults"...
  - CPU time < elapsed time
    - Partage de la machine, grand nombre d'I/O, swap important...
- Mise en oeuvre : `time <programme> <arguments du programme>`

Principes du profilage

- Deux approches : échantillonnage ou comptage
- Échantillonnage du compteur de programme
  - Le PC (*program counter)* est échantillonné, à intervalles réguliers, pour savoir quelles fonctions sont appelées
  - Avec des recompilations particulières, on peut obtenir le graphe d'appel et le rapport en fonctions appelantes/appelées
- Échantillonnage de machine virtuelle
  - Très facile
  - En Java : la machine note, à intervalles réguliers, quelle méthode est active (remontée de la pile d'appel jusqu'à un certain niveau)
  - Mais fortement dépendant du taux d'échantillonnage

```
java -Xrunhprof
```


Principes du profilage (suite)

- Comptage de blocs de base
  - Insertion du code de comptage dans des endroits clés du programme
  - Produit le comptage du nombre de fois que les instructions s'exécutent
- Exemple : pixie
  - Lit un programme **exécutable**, le partitionne en « blocs de base » et écrit un programme équivalent
    - Qui contient du code additionnel pour mesurer le temps de chaque bloc de base
    - Un bloc de base est une séquence de code sans débranchement
  - Présent sur architecture Compaq TrueUNIX 64 et d'anciennes architectures similaires

Gprof

- G pour Graph, pas pour GNU
- Profile C, Fortran, Pascal
- Produit les informations sur le graphe d'appel
- Nécessite la recompilation et la reliure de l'application
- Calcule le temps passé dans chaque routine. Les temps sont ensuite propagés le long du graphe d'appel.
- Peut aussi fournir
  - `-A[aymspec]` un source annoté, avec un profilage ligne par ligne
- Devenu un standard sous linux

Phases d'utilisation de gprof

- Compilation avec les options « gprof »
  - `CFLAGS = -pg`
  - L'option –pg est aussi utilisée en reliure pour relier les versions des bibliothèques qui ont été compilées pour le profilage
- Exécution du programme
  - Génération d'un fichier `gmon.out` qui contient les informations mesurées en format interne

Phases d'utilisation de gprof

- Traduction des informations de mesure en format « lisible »

```
gprof options [Executable file [profile data files ... ] ]
   [ > human-readable-output-file]
```

```
matrix> gprof make gmon.out > profile-make-with-Apache.txt
```

- Lecture du fichier objet (par défaut « a.out »)
- Établissement de la relation entre la table des symboles et les informations du graphe d'appel contenu dans « gmon.out »
- Si plusieurs fichiers de profile sont donnés, les informations de profilage sont additionnées

Informations obtenues

- **Flat Profile**
  - temps CPU par fonction et combien de fois elle a été appelée
  - Vue résumée des fonctions a réécrire ou à améliorer
- **Call Graph**
  - Pour chaque fonction, le nombre de fois qu'elle a été appelée par différentes fonctions (et aussi par elle-même)
  - Suggestion de quels appels peuvent être supprimés ou remplacés par des fonctions plus efficaces
  - Détermination des interrelations entre différentes fonctions (découverte de bugs)
  - Possibilité d'optimisation de certains chemins dans le graphe d'appels.

Profilons...make !

- On récupère les sources
- On « configure »
- On ajoute –pg à CFLAGS en supprimant les options d'optimisation
- On recompile le tout et on obtient un make à profiler...
- Et avec ça, on recompile Apache !

```
matrix> gprof make gmon.out > profile-make-with-Apache.txt
```


Vue « flat »

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.33      0.01     0.01      207     0.05     0.05  file_hash_2
 33.33      0.02     0.01       38     0.26     0.26  new_pattern_rule
 33.33      0.03     0.01        6     1.67     2.81  pattern_search
  0.00      0.03     0.00     2881     0.00     0.00  hash_find_slot
  0.00      0.03     0.00     2529     0.00     0.00  xmalloc
  0.00      0.03     0.00     1327     0.00     0.00  hash_find_item
  0.00      0.03     0.00     1015     0.00     0.00  directory_hash_cmp
  0.00      0.03     0.00      963     0.00     0.00  find_char_unquote
  0.00      0.03     0.00      881     0.00     0.00  file_hash_1
  0.00      0.03     0.00      870     0.00     0.00  variable_buffer_output
```

3 fonctions prennent quasiment tout le temps

Il y a 6 appels à **pattern_search**, mais cela prend 2.81ms par appel

Mais encore ?

- Vue « flat » uniquement
- Un seul jeu d'essai

- Profiler make dans plusieurs constructions (lynx, cvs, make, patch)

```
matrix> gprof make gmon-*.out > overall-profile.txt
```


Nouvelle vue « flat »

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 18.18      0.06     0.06    23480     0.00     0.00  find_char_unquote
 12.12      0.10     0.04      120     0.33     0.73  pattern_search
  9.09      0.13     0.03     5120     0.01     0.01  collapse_continuations
  9.09      0.16     0.03      148     0.20     0.88  update_file_1
  9.09      0.19     0.03       37     0.81     4.76  eval
  6.06      0.21     0.02    12484     0.00     0.00  file_hash_1
  6.06      0.23     0.02     6596     0.00     0.00  get_next_mword
  3.03      0.24     0.01    29981     0.00     0.00  hash_find_slot
  3.03      0.25     0.01    14769     0.00     0.00  next_token
  3.03      0.26     0.01     5800     0.00     0.00  variable_expand_string
```

Il y a 23480 appels à `find_char_unquote` et cela prend 1/6 du temps

`eval` n'a que 37 invocations mais elle prend 1/11 du temps et prend 4.76 ms par appel

pattern_search et update_file_1 prennent ¼ du temps en 268 appels

Vue « call graph » de make Apache

Temps total dans eval

```
index % time    self  children    called           name
-----------------------------------------------
[25]     3.7    0.00     0.00       6         eval  [25]
                0.00     0.00     219/219        try_variable_definition [28]
                0.00     0.00      48/48         record_files [40]
                0.00     0.00     122/314        variable_expand_string  [59]
                0.00     0.00       5/314        allocated_variable_expand_file  [85]
                0.00     0.00     490/490        readline [76]
                0.00     0.00     403/403        collapse_continuations [79]
                0.00     0.00     355/355        remove_comments [80]
                0.00     0.00     321/963        find_char_unquote [66]
                0.00     0.00     170/170        get_next_mword [88]
                0.00     0.00     101/111        parse_file_seq [93]
                0.00     0.00     101/111        multi_glob [92]
                0.00     0.00      48/767        next_token [70]
                0.00     0.00      19/870        variable_buffer_output [68]
                0.00     0.00    13/2529         xmalloc [64]
                0.00     0.00       2/25         xrealloc [99]
                                   5             eval_makefile  [49]
```

Index des fonctions à la fin

Nb d'appel fait par eval

Nombre total d'appels non récursifs faits à la fonction

Appels (mutuellement) récursifs

Profiler des bibliothèques

- Gprof ne profile pas les bibliothèques partagées...
- `Sprof` et `LD_PROFILE`
  - une bibliothèque à la fois...
  - Placer la variable LD_PROFILE avec le nom de la bibliothèque partagée

```
setenv LD_PROFILE my_obj
```

  - Exécuter l'application

```
my_app
```

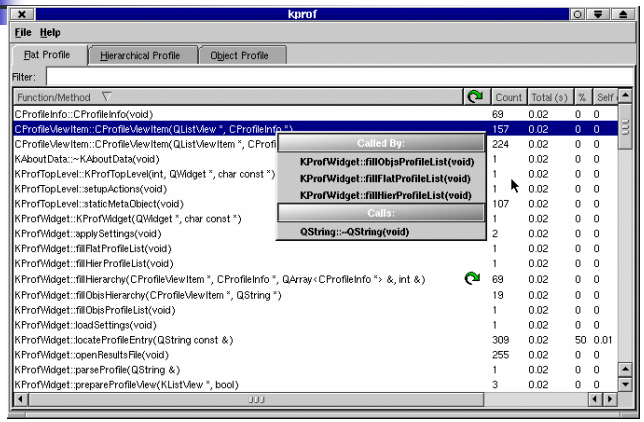
  - Cela crée un fichier `/var/tmp/my_sobj.profile`
  - Exécuter sprof

```
sprof my_sobj my_sobj.profile
```

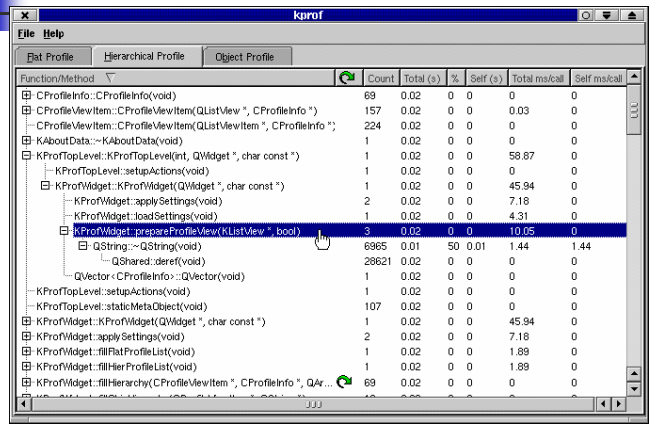
Améliorer la lecture et l'analyse

- Des outils graphiques
- Kprof
  - Vue plate
  - Vue hiérarchique (un nœud par fonction avec les appelés)
  - Vue graphique du graphe d'appel
  - Vue par objet pour C++
  - Mode *diff* pour comparer deux profils !
  - Interprète
    - Gprof
    - Function check (nouveau profileur couplé à GPC)
    - PALMOS Emulator…

Kprof : vue plate



Kprof: vue hiérarchique



Conclusion

- L'analyse de performance
  - Cela n'est pas bien compliqué
  - Cela ne prend pas beaucoup de temps
  - Cela fait souvent apparaître plus que des problèmes de performance
- Pour choisir une méthode de profilage, il faut tenir compte de :
  - Quelles statistiques vous voulez obtenir (utilisation du CPU, nombre d'appels, coût des appels, utilisation de la mémoire, opérations d'entrée/sortie…)
  - Du niveau auquel ses statistiques doivent être obtenues (procédure, instruction)
  - Est-ce que les bibliothèques partagées doivent être aussi profilées
  - La méthode pour collecter les statistiques (compilation, reliure, instrumentation)
  - Les outils pour visualiser et exploiter les données obtenues.

Introduction à la documentation

Pourquoi documenter (le code) ?

- Ce que l'on décrit bien, se conçoit bien !
- Pour le prochain programmeur ou nous même ?
- Est-ce du temps perdu ?
  ... Et jusqu'à quel point documenter ?
- Documenter, (une fonction)
  ... n'est pas commenter! (le code)

Problématiques

- Programmeur :
  - Historique: qui/quand/quoi
  - Algorithme
  - Borne d'utilisation
  - Paramètres
  - Valeur retournée
  - Structure du module
  - Mode d'utilisation
  - Code d'erreur
- Mainteneur :
  - Historique: qui/quand/quoi
  - Documentation utilisée à jour
  - Que fait la fonction
  - Quel module utilise quelle fonction
  - Effet de cascade de la modification en cours

Garder synchrone code et documentation

- Une documentation obsolète est une documentation inutile
- Documentation interne et extraction :
  - Commentaires formatés
  - Graphe de dépendance entre les classes, les fichiers
  - Hiérarchie de classes
- Mais la documentation externe a des avantages
  - Lisibilité accrue
  - Synthèse du logiciel

Exemple : Javadoc

- Extracteur spécialisé java (très simple)
  - Cohérence avec les conventions Java
  - Lancement possible sur toute une hiérarchie
  - Production de html
  - Extensible avec des « doclets » (changement du format de sortie)

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
              try {
                  return getImage(new URL(url, name));
              } catch (MalformedURLException e) {
                  return null;
              }
}
```

getImage

public Image getImage(URL url, String name)

Returns an Image object that can then be painted on the screen. The url argument must specify an absolute URL. The name argument is a specifier that is relative to the url argument. This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

url - an absolute URL giving the base location of the image

name - the location of the image, relative to the url argument

Returns:

the image at the specified URL

See Also:

Image

Exemple : Doxygen

- Multi-langages
  - C++, C, Java, Objective-C, Python, IDL (Corba et Microsoft)
  - partiellement PHP, C#
- Multi-plates-formes
  - Unix, Windows
- Multi-sorties
  - HTML, LaTeX (PDF), RTF, Man, XML, Aide Windows
- Traitements plus élaborés :
  - Fichier de configuration, assistant d'aide à la configuration
  - Génération de graphes avec graphviz/dot
  - Génération d'un moteur de recherche (pour site web avec PHP)

Doxygen : graphes

- Hiérarchie de classes
- Graphes d'inclusion
- Graphes de collaboration
- Graphes d'appel