Programmation C avancée

Concepts & Outils
pour le développement

maj 03/2024

Mesure de la performance

- La mesure de la performance d'un logiciel est complexe, car la prise de mesures doit perturber le moins possible le fonctionnement du système exécutant le logiciel
- Les mesures de performances peuvent être effectuées au niveau :
  - Du processeur
  - Du système d'exploitation
  - Du programme

Code profiling

- Le profilage de code est l'action d'instrumenter le code source afin d'en obtenir des mesures d'usage lors de l'exécution de jeux de tests
  - Ajout de routines de comptage de passages en différents points du code source
  - Activation de registres matériels du processeur dédiés à cette activité
    - Comptage du nombre de cycles consommés, des accès mémoire, des défauts de cache...

time

- Commande Unix classique donnant un résumé des ressources consommées par le programme passé en paramètre
  - Temps CPU consommé en mode utilisateur
  - Temps CPU consommé en mode système
  - Temps réel écoulé depuis le lancement
  - Taille mémoire utilisée par le code et les données
  - Nombre de défauts de page, etc...

```
% time gcc -O3 brol.c -o brol
0.05user 0.02system 0:00.18elapsed 43%CPU (0+0)k 0in+0out
(9major+2815minor)pagefaults 0swaps
```

Routines de mesure

- La routine `clock()`, appartenant à la bibliothèque standard, renvoie le temps CPU écoulé depuis une date d'origine
  - On calcule le temps consommé dans une routine par soustraction entre la valeur à la sortie et la valeur à l'entrée
- La routine `getrusage()` permet d'obtenir, au niveau du processus, les informations affichées par la commande **time**
  - Temps CPU utilisateur et système, mémoire, ...

getrusage

```
int getrusage(int who, struct rusage *usage);


 struct rusage {
        struct timeval ru_utime; /* user CPU time used */
        struct timeval ru_stime; /* system CPU time used */
        long   ru_maxrss;        /* maximum resident set size */
        long   ru_ixrss;         /* integral shared memory size */
        long   ru_idrss;         /* integral unshared data size */
        long   ru_isrss;         /* integral unshared stack size */
        long   ru_minflt;        /* page reclaims (soft page faults) */
        long   ru_majflt;        /* page faults (hard page faults) */
        long   ru_nswap;         /* swaps */
        long   ru_inblock;       /* block input operations */
        long   ru_oublock;       /* block output operations */
        long   ru_msgsnd;        /* IPC messages sent */
        long   ru_msgrcv;        /* IPC messages received */
        long   ru_nsignals;      /* signals received */
        long   ru_nvcsw;         /* voluntary context switches */
        long   ru_nivcsw;        /* involuntary context switches */
    };
```

Compteurs matériels (1)

- Les processeurs modernes disposent tous de circuits destinés à la mesure de performance
- Compteurs paramétrables d'événements internes au processeur, tels que nombres de cycles consommés, de lectures ou écritures, d'opérations à virgule flottante, de défauts de cache de premier ou deuxième niveau, ...
  - Deux registres compteurs de 40 bits disponibles sur le Pentium II, quatre de 48 bits sur l'Athlon

http://download.intel.com/design/PentiumII/manuals/24319202.pdf, table A.1

Compteurs matériels (2)

- Différentes bibliothèques permettent de sélectionner le type d'événements à compter et de lire la valeur des compteurs
  - Dépendantes du processeur et du système d'exploitation
- Tentatives d'offrir une interface unifiée pour plusieurs processeurs et systèmes
  - ***papi***, de l'Université du Tennessee

http://icl.cs.utk.edu/papi/
http://user.it.uu.se/~mikpe/linux/perfctr/

gprof (1)

- **gprof** est un outil de profilage, permettant de savoir dans quelles routines un programme passe le plus de temps, et quel est l'arbre d'appel du programme
- **gprof** analyse a posteriori les traces générées par l'exécution d'un programme, et produit un relevé statistique du temps passé dans chaque routine

gprof (2)

- Pour que l'exécution du programme génère des traces exploitables, il faut compiler avec l'option « `-pg` » de `gcc`
  - Réalise l'édition de liens avec les bibliothèques adaptées
- À la fin de l'exécution, les traces sont collectées dans le fichier « `gmon.out` »

```
% gcc -pg brol.c -o brol
% ./brol
% ls
brol
brol.c
gmon.out
```

gprof (3)

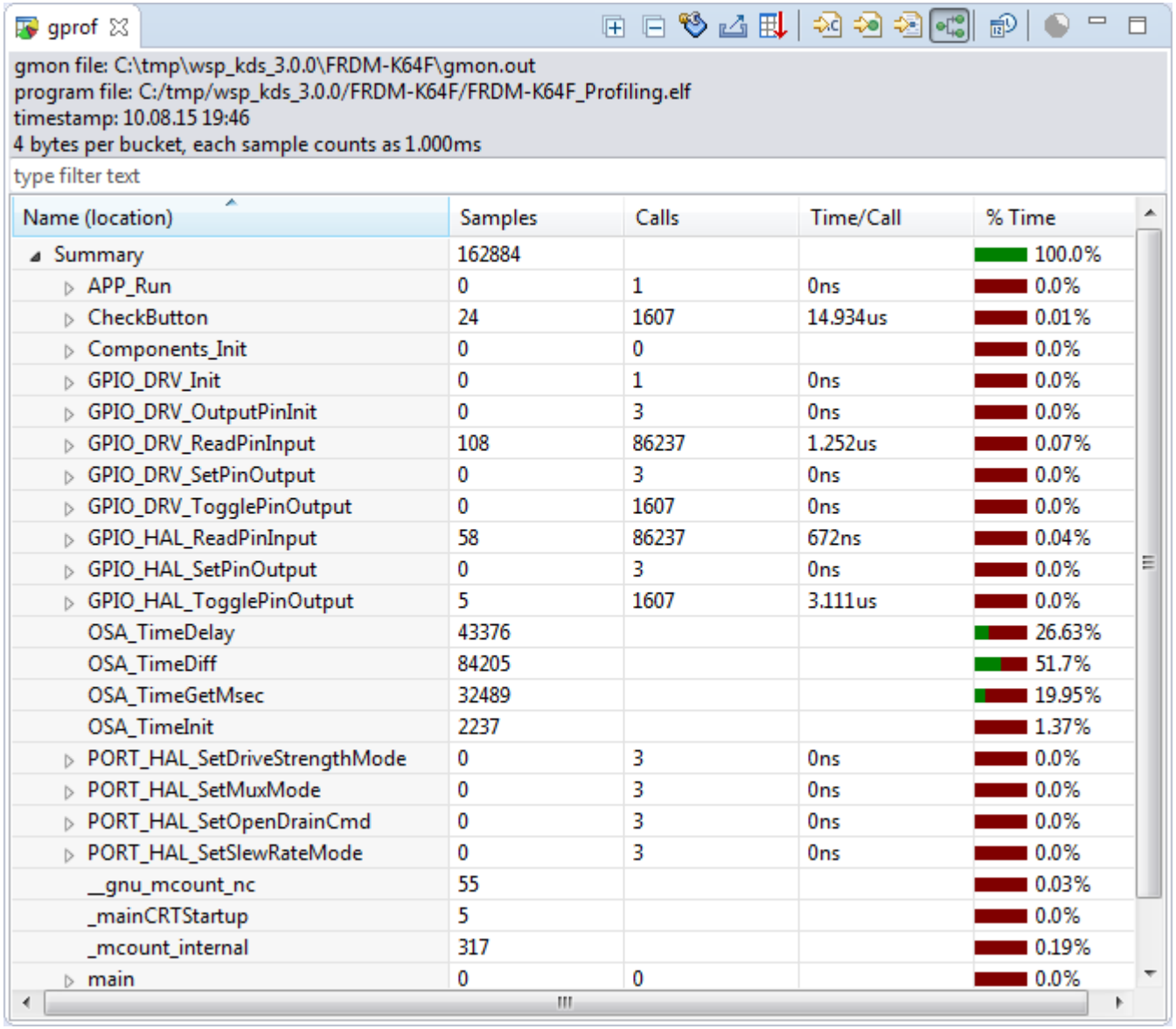
- Le rapport d'exécution est créé par la commande `gprof` proprement dite
  - Rapport d'exécution en format texte
- Classe les fonctions par ordre décroissant de temps consommé dans la fonction et dans les sous-fonctions qu'elle a appelées

```
% gprof brol gmon.out > rapport.txt
% more rapport.txt
```

gprof (4)

```
index % time    self  children    called     name
...
[6]     87.7    0.04   35.95    1099+5870    <cycle 2 as a whole> [6]
                0.03   19.27    4773+1647        vgraphSeparateSt [7]
                0.00    0.00    1098             vgraphSeparateMl[42]
-----------------------------------------------
                                 1647             vgraphSeparateSt [7]
                                 3674             vgraphSeparateMl2[9]
                0.04   35.95    1099/1099        hgraphOrderNd [5]
[7]     47.0    0.03   19.27    4773+1647    vgraphSeparateSt [7]
               15.06    1.72    2576/2576        vgraphSeparateFm [8]
                2.10    0.39    1098/1098        vgraphSeparateGg[14]
                0.00    0.00    1099/1099        stratTestEval [41]
                0.00    0.00    1098/1098        vgraphStoreInit [44]
                0.00    0.00    1098/1098        vgraphStoreSave [45]
                0.00    0.00    1098/1098        vgraphStoreExit [43]
                0.00    0.00     714/714         vgraphStoreUpdt [47]
                                 1098             vgraphSeparateMl[42]
                                 1647             vgraphSeparateSt [7]
```

gprof via un ide (eclipse)

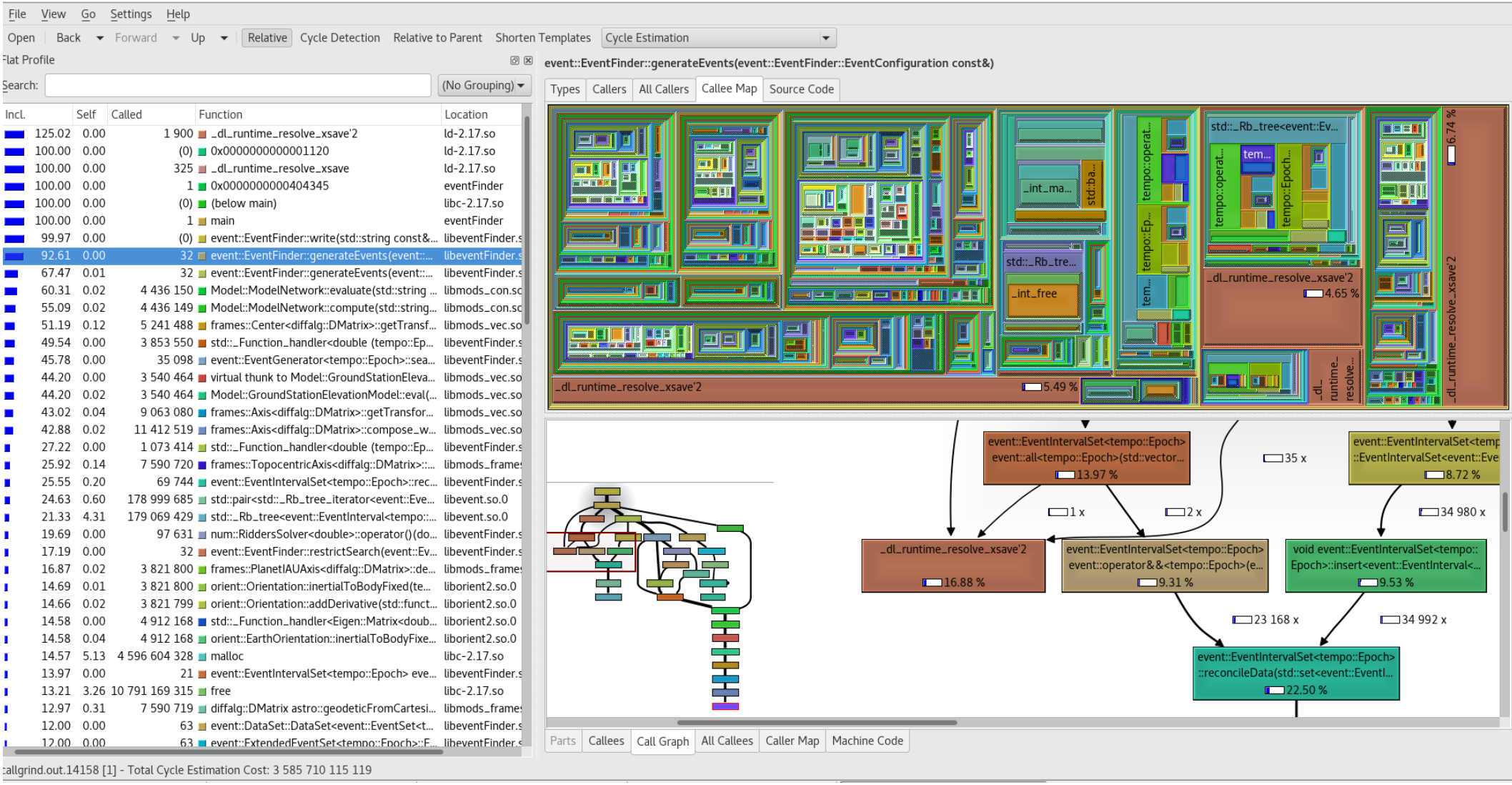


Name (location)	Samples	Calls	Time/Call	% Time
Summary	162884			100.0%
APP_Run	0	1	0ns	0.0%
CheckButton	24	1607	14.934us	0.01%
Components_Init	0	0		0.0%
GPIO_DRV_Init	0	1	0ns	0.0%
GPIO_DRV_OutputPinInit	0	3	0ns	0.0%
GPIO_DRV_ReadPinInput	108	86237	1.252us	0.07%
GPIO_DRV_SetPinOutput	0	3	0ns	0.0%
GPIO_DRV_TogglePinOutput	0	1607	0ns	0.0%
GPIO_HAL_ReadPinInput	58	86237	672ns	0.04%
GPIO_HAL_SetPinOutput	0	3	0ns	0.0%
GPIO_HAL_TogglePinOutput	5	1607	3.111us	0.0%
OSA_TimeDelay	43376			26.63%
OSA_TimeDiff	84205			51.7%
OSA_TimeGetMsec	32489			19.95%
OSA_TimeInit	2237			1.37%
PORT_HAL_SetDriveStrengthMode	0	3	0ns	0.0%
PORT_HAL_SetMuxMode	0	3	0ns	0.0%
PORT_HAL_SetOpenDrainCmd	0	3	0ns	0.0%
PORT_HAL_SetSlewRateMode	0	3	0ns	0.0%
__gnu_mcount_nc	55			0.03%
_mainCRTStartup	5			0.0%
_mcount_internal	317			0.19%
main	0	0		0.0%

valgrind

- En plus de memcheck, valgrind dispose d'autres « outils » (valgrind est une plateforme)
  - cachegrind : analyse au niveau des caches processeur
  - callgrind : cachegrind + graphe d'appel, à utiliser avec kcachegrind (visualisateur)
  - massif : analyse de l'évolution du tas
  - dhat : analyse des accès au tas

kcachegrind



Programmation C avancée

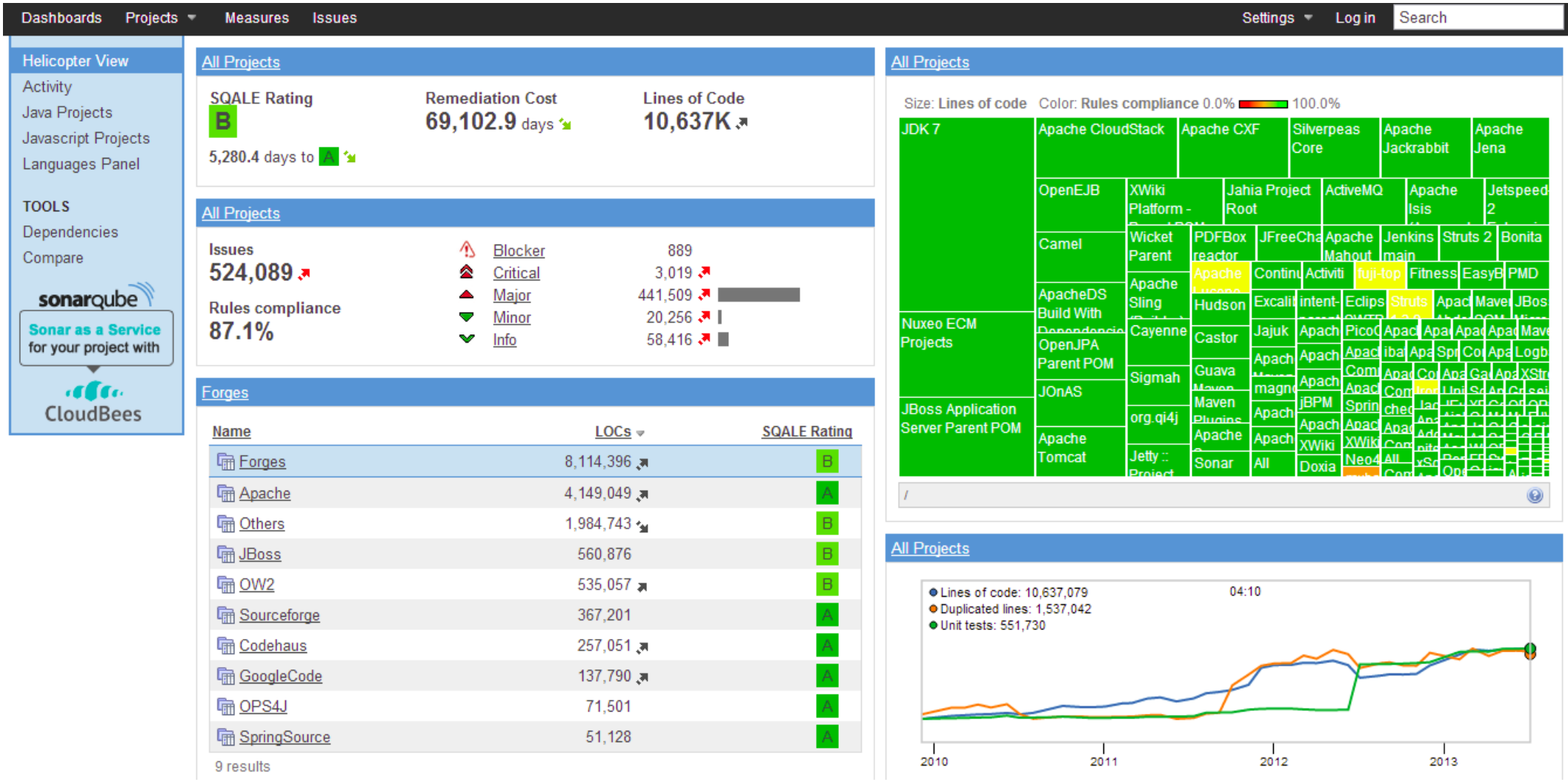
Les métriques

Qualité: outils et métriques

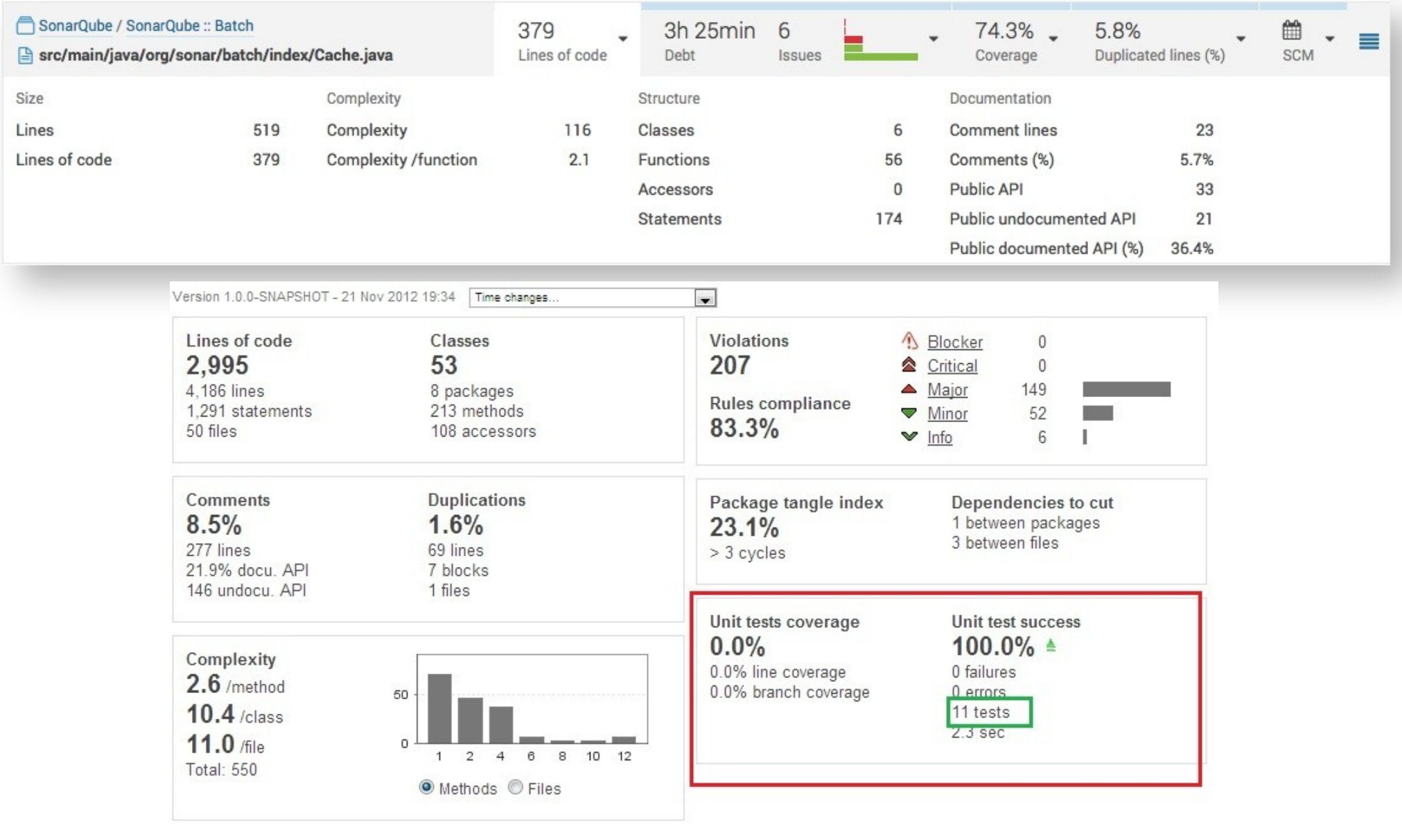
- Des métriques et propriétés du code:
  - duplications
  - commentaires
  - nombres de fichiers / classes / méthodes / fonctions / ....
  - convention de nommage
  - règles d'implémentation spécifiques
- évolution de ces métriques dans le temps
- exemple:

http://nemo.sonarsource.org/

SonarQube

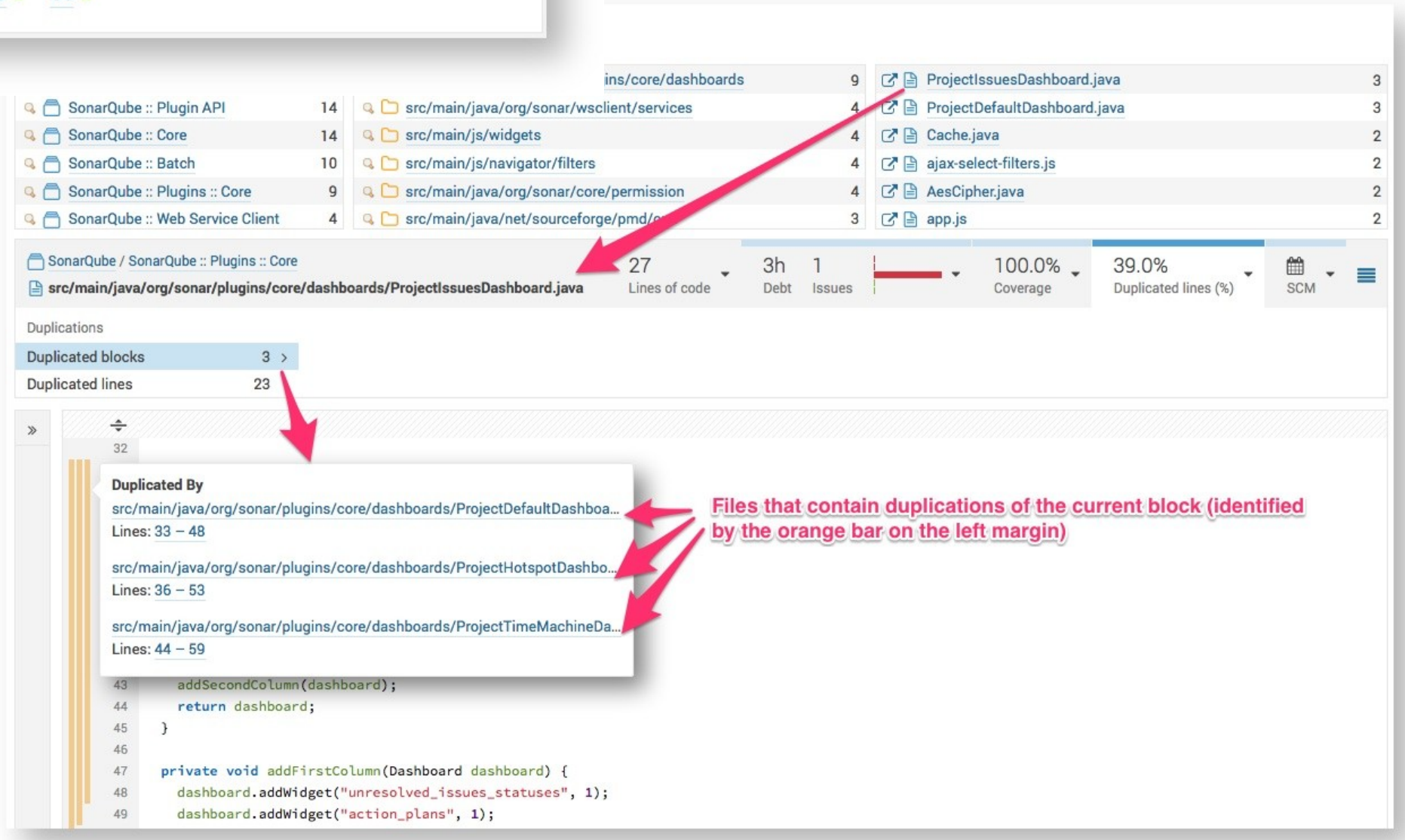

Dashboards
Projects
Measures
Issues
Settings
Log in
Search
Helicopter View
Activity
Java Projects
Javascript Projects
Languages Panel
TOOLS
Dependencies
Compare
sonarqube
Sonar as a Service
for your project with
CloudBees
All Projects
SQALE Rating
B
5,280.4 days to A
Remediation Cost
69,102.9 days
Lines of Code
10,637K
All Projects
Issues
524,089
Rules compliance
87.1%
Blocker 889
Critical 3,019
Major 441,509
Minor 20,256
Info 58,416
Forges
Name
LOCs
SQALE Rating
Forges 8,114,396 B
Apache 4,149,049 A
Others 1,984,743 B
JBoss 560,876 B
OW2 535,057 B
Sourceforge 367,201 A
Codehaus 257,051 A
GoogleCode 137,790 A
OPS4J 71,501 A
SpringSource 51,128 A
9 results
All Projects
Size: Lines of code Color: Rules compliance 0.0% 100.0%
JDK 7
Nuxeo ECM Projects
JBoss Application Server Parent POM
Apache CloudStack
Apache CXF
Silverpeas Core
Apache Jackrabbit
Apache Jena
OpenEJB
XWiki Platform -
Jahia Project Root
ActiveMQ
Apache Isis
Jetspeed 2
Camel
Wicket Parent
PDFBox reactor
JFreeCha
Apache Mahout
Jenkins main
Struts 2
Bonita
Apache Sling
Apache
Continu
Activiti
fuji-top
Fitness
EasyB
PMD
ApacheDS Build With
Hudson
Excalil
intent-
Eclips
Struts
Apac
Maver
JBos
Cayenne
Castor
Jajuk
Apach
Pico
Apac
Apa
Apac
Apac
Mave
OpenJPA Parent POM
Apach
Apach
iba
Apa
Spr
Co
Apa
Logb
Sigmah
Apach
Com
Apac
Co
Apa
Ga
Apa
XStr
JOnAS
magn
Apac
Con
Maven Plugins
jBPM
Sprin
org.qi4j
Apach
Apach
Apac
Apa
Apache Tomcat
Apache
Apach
XWiki
XWiki
Jetty :: Project
Sonar
All
Doxia
Neo4
All
All Projects
Lines of code: 10,637,079
Duplicated lines: 1,537,042
Unit tests: 551,730
04:10
2010
2011
2012
2013

SonarQube

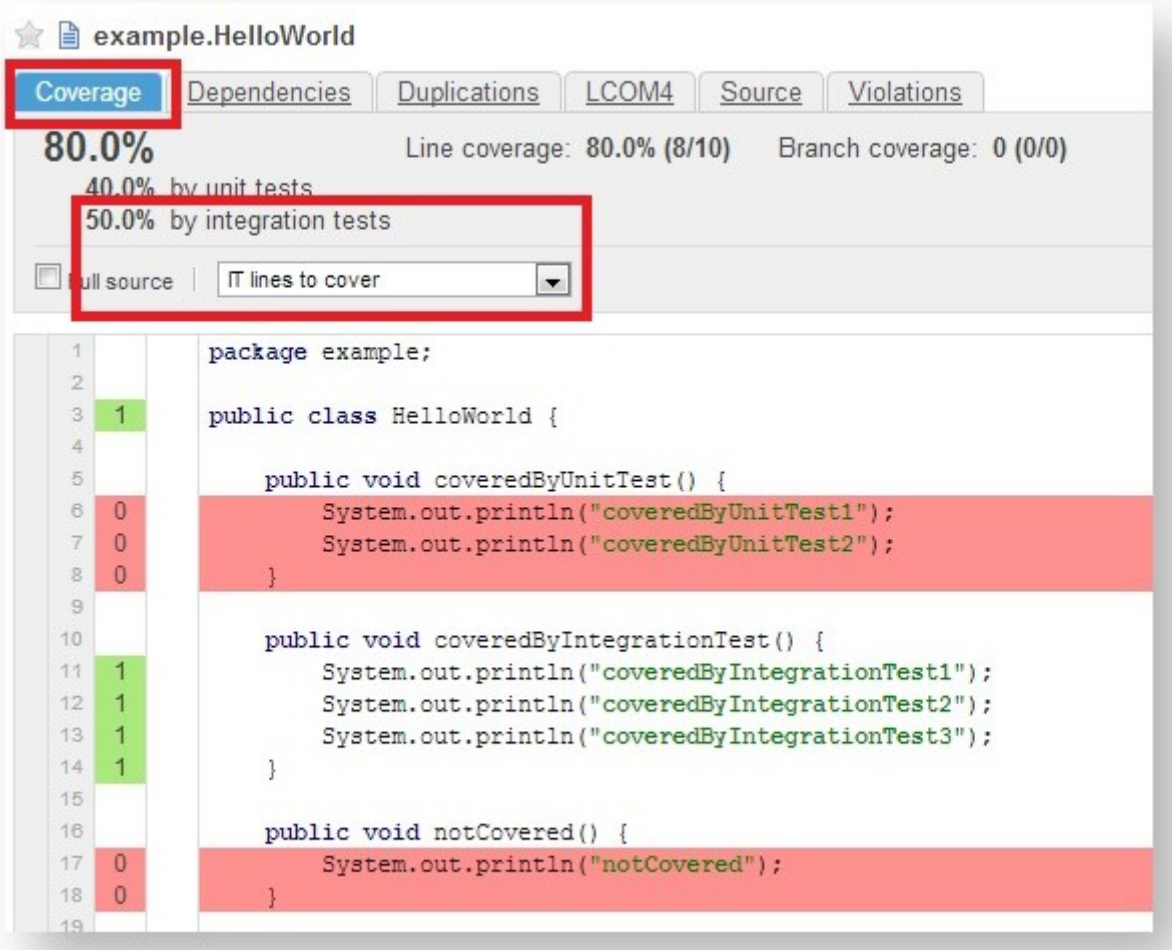

SonarQube / SonarQube :: Batch
src/main/java/org/sonar/batch/index/Cache.java
379
Lines of code
3h 25min
Debt
6
Issues
74.3%
Coverage
5.8%
Duplicated lines (%)
SCM
Size
Lines 519
Lines of code 379
Complexity
Complexity 116
Complexity /function 2.1
Structure
Classes 6
Functions 56
Accessors 0
Statements 174
Documentation
Comment lines 23
Comments (%) 5.7%
Public API 33
Public undocumented API 21
Public documented API (%) 36.4%
Version 1.0.0-SNAPSHOT - 21 Nov 2012 19:34
Time changes...
Lines of code
2,995
4,186 lines
1,291 statements
50 files
Classes
53
8 packages
213 methods
108 accessors
Violations
207
Blocker 0
Critical 0
Major 149
Minor 52
Info 6
Rules compliance
83.3%
Comments
8.5%
277 lines
21.9% docu. API
146 undocu. API
Duplications
1.6%
69 lines
7 blocks
1 files
Package tangle index
23.1%
> 3 cycles
Dependencies to cut
1 between packages
3 between files
Unit tests coverage
0.0%
0.0% line coverage
0.0% branch coverage
Unit test success
100.0%
0 failures
0 errors
11 tests
2.3 sec
Complexity
2.6 /method
10.4 /class
11.0 /file
Total: 550
50
0
1 2 4 6 8 10 12
Methods
Files

SonarQube

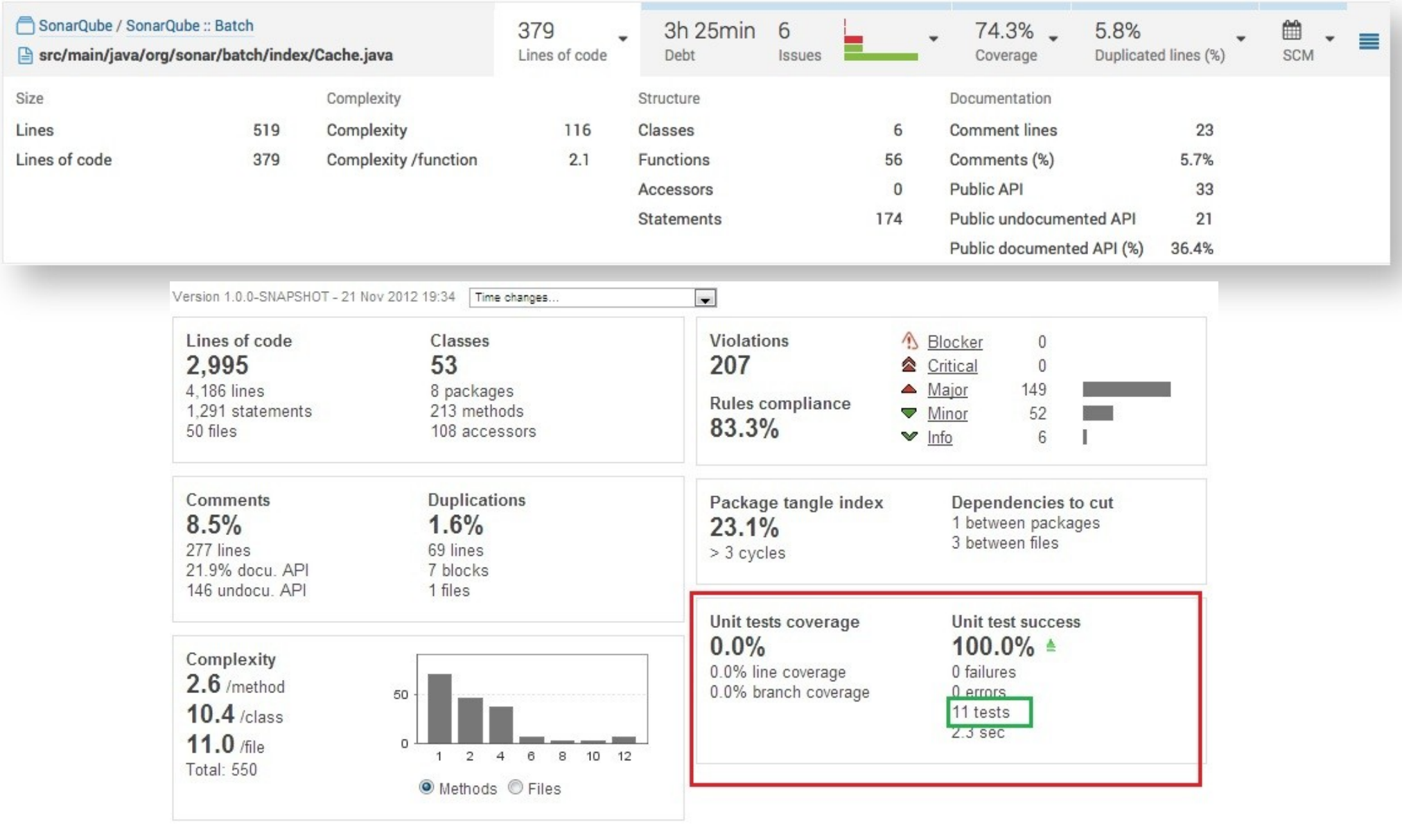
Duplications
0.9%
Lines
1,781
Blocks
85
Files
65
Drilldown to find where duplications are


ins/core/dashboards 9
SonarQube :: Plugin API 14
SonarQube :: Core 14
SonarQube :: Batch 10
SonarQube :: Plugins :: Core 9
SonarQube :: Web Service Client 4
src/main/java/org/sonar/wsclient/services 4
src/main/js/widgets 4
src/main/js/navigator/filters 4
src/main/java/org/sonar/core/permission 4
src/main/java/net/sourceforge/pmd/ 3
ProjectIssuesDashboard.java 3
ProjectDefaultDashboard.java 3
Cache.java 2
ajax-select-filters.js 2
AesCipher.java 2
app.js 2
SonarQube / SonarQube :: Plugins :: Core
src/main/java/org/sonar/plugins/core/dashboards/ProjectIssuesDashboard.java
27 Lines of code
3h Debt
1 Issues
100.0% Coverage
39.0% Duplicated lines (%)
SCM
Duplications
Duplicated blocks 3
Duplicated lines 23
Duplicated By
src/main/java/org/sonar/plugins/core/dashboards/ProjectDefaultDashboa...
Lines: 33 – 48
src/main/java/org/sonar/plugins/core/dashboards/ProjectHotspotDashbo...
Lines: 36 – 53
src/main/java/org/sonar/plugins/core/dashboards/ProjectTimeMachineDa...
Lines: 44 – 59
Files that contain duplications of the current block (identified by the orange bar on the left margin)
addSecondColumn(dashboard);
return dashboard;
}
private void addFirstColumn(Dashboard dashboard) {
dashboard.addWidget("unresolved_issues_statuses", 1);
dashboard.addWidget("action_plans", 1);

SonarQube


example.HelloWorld
Coverage
Dependencies
Duplications
LCOM4
Source
Violations
80.0%
Line coverage: 80.0% (8/10)
Branch coverage: 0 (0/0)
40.0% by unit tests
50.0% by integration tests
ull source
IT lines to cover
1 package example;
2
3 1 public class HelloWorld {
4
5 public void coveredByUnitTest() {
6 0 System.out.println("coveredByUnitTest1");
7 0 System.out.println("coveredByUnitTest2");
8 0 }
9
10 public void coveredByIntegrationTest() {
11 1 System.out.println("coveredByIntegrationTest1");
12 1 System.out.println("coveredByIntegrationTest2");
13 1 System.out.println("coveredByIntegrationTest3");
14 1 }
15
16 public void notCovered() {
17 0 System.out.println("notCovered");
18 0 }

SonarQube


SonarQube / SonarQube :: Batch
src/main/java/org/sonar/batch/index/Cache.java
379
Lines of code
3h 25min
Debt
6
Issues
74.3%
Coverage
5.8%
Duplicated lines (%)
SCM
Size
Lines 519
Lines of code 379
Complexity
Complexity 116
Complexity /function 2.1
Structure
Classes 6
Functions 56
Accessors 0
Statements 174
Documentation
Comment lines 23
Comments (%) 5.7%
Public API 33
Public undocumented API 21
Public documented API (%) 36.4%
Version 1.0.0-SNAPSHOT - 21 Nov 2012 19:34
Time changes...
Lines of code
2,995
4,186 lines
1,291 statements
50 files
Classes
53
8 packages
213 methods
108 accessors
Violations
207
Blocker 0
Critical 0
Major 149
Minor 52
Info 6
Rules compliance
83.3%
Comments
8.5%
277 lines
21.9% docu. API
146 undocu. API
Duplications
1.6%
69 lines
7 blocks
1 files
Package tangle index
23.1%
> 3 cycles
Dependencies to cut
1 between packages
3 between files
Unit tests coverage
0.0%
0.0% line coverage
0.0% branch coverage
Unit test success
100.0%
0 failures
0 errors
11 tests
2.3 sec
Complexity
2.6 /method
10.4 /class
11.0 /file
Total: 550
50
0
1 2 4 6 8 10 12
Methods
Files

Programmation C avancée

Quizz compilation

Bibliothèque...

#QDLE#Q#AB*#40#

- J’ai produit mon programme ./a.out en utilisant libtoto.a, si je supprime le fichier libtoto.a alors
  - A) ./a.out ne s’exécute pas
  - B) ./a.out s’exécute

Erreur de compilation...

#QDLE#Q#ABC*#40#

- Lors de la compilation, l'erreur :

undefined reference to ...

- est une erreur :
  - A) de pré-compilation
  - B) de compilation
  - C) d'édition de lien

Compilation...

#QDLE#Q#ABCD*EFG#40#

- Quelles sont les étapes utilisées pour la création d'un objet à partir d'un fichier source:
  - A) La pré-compilation
  - B) La compilation
  - C) L’édition de lien
  - D) A & B
  - E) A & C
  - F) B & C
  - G) A & B & C

dépendances...

- mon projet se compose de
  - list.h et list.c, qui implémente un module de listes chaînées
  - hash.h et hash.c, qui implémente une table de hachage. L'implémentation **interne** de la table repose sur les listes chaînées.
  - Ces 2 modules sont compilés dans un bibliothèque libalgo.so
  - demo.c est un programme de démonstration des tables de hachage.

dépendances...

#QDLE#Q#ABCDEF*GH#40#

- mon projet se compose de
  - list.h et list.c, qui implémente un module de listes chaînées
  - hash.h et hash.c, qui implémente une table de hachage. L'implémentation interne de la table repose sur les listes chaînées.
  - Ces 2 modules sont compilés dans un bibliothèque libalgo.so
  - demo.c est un programme de démonstration des tables de hachage.

- je modifie « list.c », je dois regénérer :
  - A) list.o
  - B) hash.o
  - C) libalgo.so
  - D) demo
  - E) A & B
  - F) A & C
  - G) A & B & C
  - H) tout

dépendances...

#QDLE#Q#ABCDEFG*H#40#

- mon projet se compose de
  - list.h et list.c, qui implémente un module de listes chaînées
  - hash.h et hash.c, qui implémente une table de hachage. L'implémentation interne de la table repose sur les listes chaînées.
  - Ces 2 modules sont compilés dans un bibliothèque libalgo.so
  - demo.c est un programme de démonstration des tables de hachage.

- je modifie « list.h », je dois regénérer :
  - A) list.o
  - B) hash.o
  - C) libalgo.so
  - D) demo
  - E) A & B
  - F) A & C
  - G) A & B & C
  - H) tout

dépendances...

#QDLE#Q#ABCDE*FGH#40#

- mon projet se compose de
  - list.h et list.c, qui implémente un module de listes chaînées
  - hash.h et hash.c, qui implémente une table de hachage. L'implémentation interne de la table repose sur les listes chaînées.
  - Ces 2 modules sont compilés dans un bibliothèque libalgo.so
  - demo.c est un programme de démonstration des tables de hachage.

- je modifie « hash.c », je dois regénérer :
  - A) list.o
  - B) hash.o
  - C) libalgo.so
  - D) demo
  - E) B & C
  - F) B & C & D
  - G) A & B & C
  - H) tout

dépendances...

#QDLE#Q#ABCDEF*GH#40#

- mon projet se compose de
  - list.h et list.c, qui implémente un module de listes chaînées
  - hash.h et hash.c, qui implémente une table de hachage. L'implémentation interne de la table repose sur les listes chaînées.
  - Ces 2 modules sont compilés dans un bibliothèque libalgo.so
  - demo.c est un programme de démonstration des tables de hachage.

- je modifie « hash.h », je dois regénérer :
  - A) list.o
  - B) hash.o
  - C) libalgo.so
  - D) demo
  - E) B & C
  - F) B & C & D
  - G) A & B & C
  - H) tout

Programmation C avancée

Chargement dynamique

Chargement dynamique

- Il est possible de charger dynamiquement une bibliothèque (à l'exécution)
- Les fonctions sont :
  - void *dlopen(const char *filename, int flags);
  - void *dlsym(void *handle, const char *symbol);
  - int dlclose(void *handle);

dlopen

- dlopen prend en paramètre
  - un chemin absolu ou relatif vers une bibliothèque (.so)
  - des options sur le chargement (résolution des symboles, portée des symboles…)
- La valeur de retour est un « handler » qui permet d'utiliser la bibliothèque

dlsym

- La fonction dlsym permet d'accéder aux symboles d'une bibliothèque :
  - fonctions
  - globales
- Attention : **les symboles ne sont pas typés**
- C'est à vous de « caster » la valeur de retour vers le type supposé du symbole

dlclose

- dlclose permet de libérer les ressources associées à une bibliothèque

dlopen / dlsym / dlclose

- soit le fichier a.c :

```
int i=0 ;
int f(void){
   i+=1 ;
   return i ;
}
```

- on compile : gcc -shared -fPIC a.c -o liba.so
- nm -C liba.so :

```
0000000000201020 B __bss_start
......
0000000000000680 T f
00000000000006a4 T _fini
....
0000000000201024 B i
0000000000000540 T _init
.......
```

_init et _fini

- Les fonctions _init et _fini sont exécutées automatiquement lors du chargement et du déchargement de la bibliothèque
- Il est possible d'écrire ses propres fonctions, pour cela il faut l'indiquer au « linker » :

Solution 1 :

```
#include <stdio.h>
int i=0 ;
int f(void){
    i+=1 ;
    return i ;
}
static void load_lib(void) __attribute_((constructor)) ;
void load_lib(void){
    printf(« loading \n») ;
}
static void release_lib(void) __attribute_((destructor)) ;
void release_lib(void){
    printf(« release\n ») ;
}
```

_init et _fini

- Les fonctions _init et _fini sont exécutées automatiquement lors du chargement et du déchargement de la bibliothèque
- Il est possible d'écrire ses propres fonctions, pour cela il faut l'indiquer au « linker » :

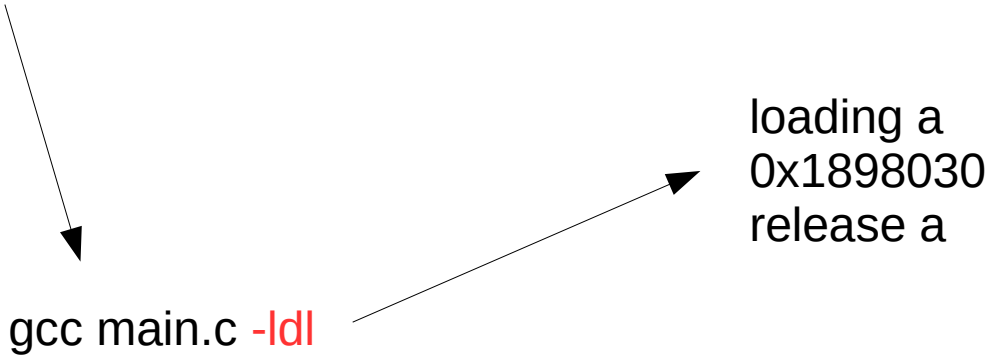
Solution 2 :

```
#include <stdio.h>
int i=0 ;
int f(void){
    i+=1 ;
    return i ;
}
void load_lib(void){
    printf(« loading \n») ;
}
void release_lib(void){
    printf(« release\n ») ;
}
```

gcc -shared -FPIC -Wl,-init,load_lib -Wl,-fini,release_lib a.c -o liba.so

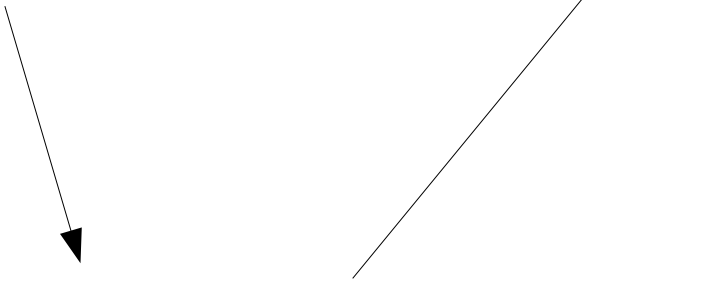
Utilisation : chargement

```
#include<stdio.h>
#include<dlfcn.h>


int main(int argc, char **argv){
  void *h=dlopen("./liba.so",RTLD_NOW);
  printf("%p\n",h);
  dlclose(h);
}
```



Utilisation : symboles

```
#include<stdio.h>
#include<dlfcn.h>


int main(int argc, char **argv){
  void *h=dlopen("./liba.so",RTLD_NOW);
  int *i;
  int (*f)(void);
  printf("%p\n",h);
  i=(int *)dlsym(h,"i");
  f=(int (*)(void ))dlsym(h,"f");
  printf("%p %d %d\n",h,f(),*i);
  printf("%p %d %d\n",h,f(),*i);
  dlclose(h);
}
```



gcc main.c -ldl

```
loading a
0x136c030
0x136c030 1 0
0x136c030 2 1
release a
```

chargement dynamique⇒ plugin

- Le chargement dynamique permet d'importer de nouvelle fonctionnalité dans un programme au cours de son exécution
- C'est un processus très utilisé pour l'implémentation de plugin
- Permet à des tiers de créer de nouvelles fonctionnalités pour un programme existant (par exemple : ajout du support de flash dans firefox)

plugins

- Pour l'auteur du programme principal :
  - Il faut décrire une série de fonctions (API) permettant d'interagir avec le plugin, par exemple :

```
void *pluginNew(void *app) ;
char *pluginGetDescription(void *) ;
char *pluginGetAuthor(void *) ;
void pluginDoOperation(void *) ;
int pluginRelease(void *) ;
```

  - On utilisera de préférence un système de handler permettant
    - d'avoir plusieurs instances d'un même plugin,
    - de permettre au plugin de stocker des données.

plugins

- Pour l'auteur du plugin :
  - Il faut décrire une série de fonctions (API) permettant d'interagir avec le plugin, par exemple :

```
void *pluginNew(void *app) ;
char *pluginGetDescription(void *) ;
char *pluginGetAuthor(void *) ;
void pluginDoOperation(void *) ;
int pluginRelease(void *) ;
```

  - On utilisera de préférence un système de handler permettant
    - d'avoir plusieurs instances d'un même plugin,
    - de permettre au plugin de stocker des données.

plugin: approche part _init

- Une autre solution est d'utiliser le fonction de chargement de la bibliothèque:
- L'application expose une API telle que:

```
enum Event { .... };

void registerCallBack(Event e, void (*cb)(void *), void
*data)

void* removeCallBack(void (*cb)(void *))
```

- Le plugin va utiliser cette API dans sa fonction _init...

plugin: approche part _init

```
#include <application_plugin.h>
void onOpen(struct param *);
struct param{…}
struct param *p=NULL ;
void _init(){
     p=malloc(sizeof(*p));
     p->value=....;
     registerCallBack(OPEN,onOpen,p);
}
void _fini(){
    void *p=removeCallBack(onOpen);
    if (p!=NULL) free(p);
}
```

- Il reste à ajouter les éléments permettant l'édition de lien pour créer le plugin...