

Journées bonnes pratiques pour le développement
13-14 octobre 2008

Le processus de développement

Fabien Spindler

SED-REN

Equipe-projet Lagadic

INRIA Rennes-Bretagne Atlantique - IRISA

<http://www.irisa.fr/lagadic>



INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Plan

1. Introduction
2. Quelques définitions
3. Un exemple de projet
4. Difficultés
5. *Build* automatique: Make
6. Conclusion

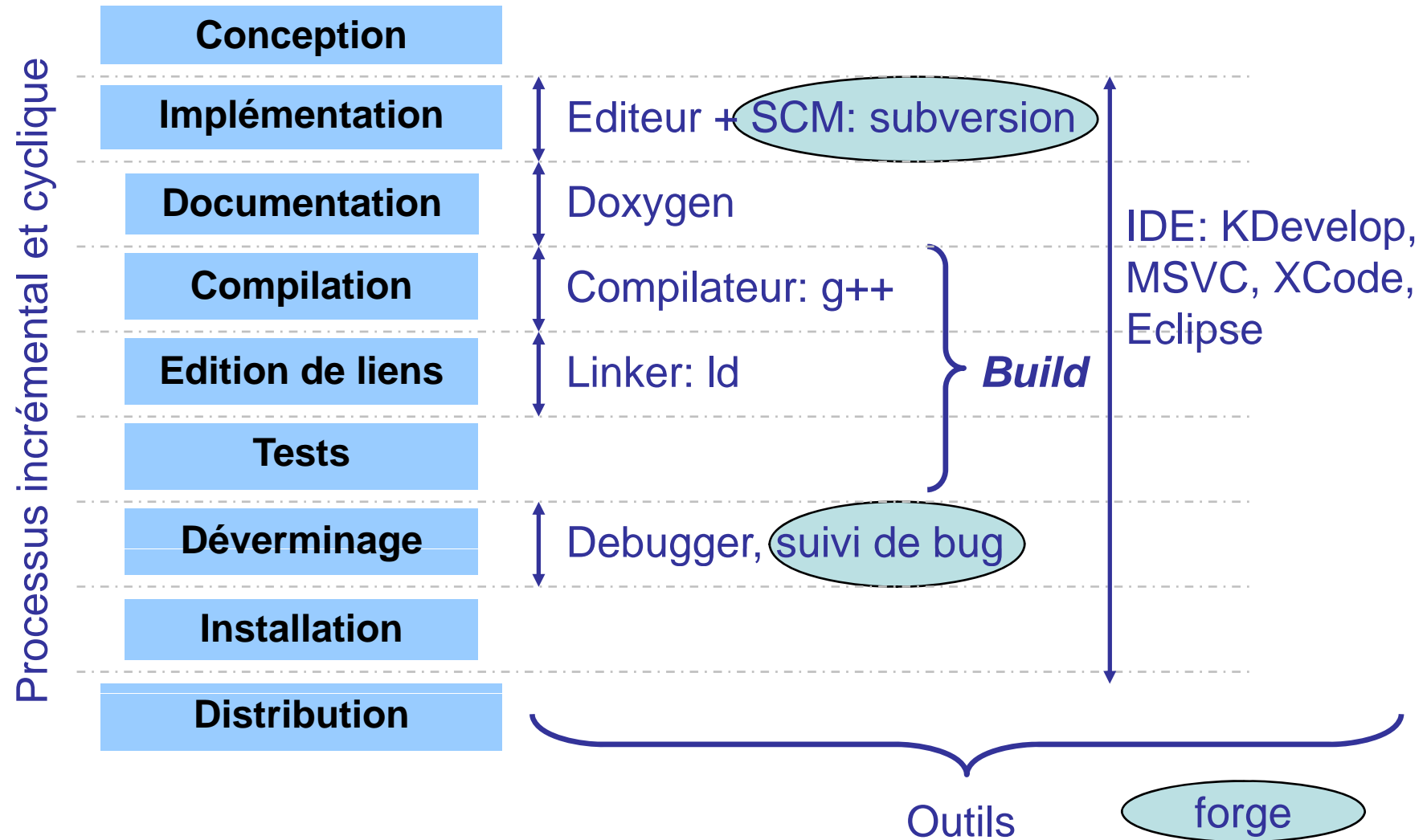


INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



centre de recherche
RENNES - BRETAGNE ATLANTIQUE

1. Introduction



2. Quelques définitions

Fichier source

- Fichier texte: foo.h, foo.cpp

Fichier objet

- Fichier binaire contenant la version compilée d'un code source: foo.o

Librairie

- Une collection de fichiers objets assemblés dans un fichier binaire
- Permet de modulariser les développements en composants

Exécutable

- Fichier binaire pouvant être exécuté sur une cible

Compilateur

- Outil utilisé pour générer un fichier objet à partir de fichiers sources

Linker

```
g++ -W -Wall -I<inc path> -c foo.cpp
```

- Outil utilisé pour générer un exécutable ou une librairie à partir de fichiers objets

```
g++ -o foo foo.o -L<lib path> -lbar
```



2. Quelques définitions

Librairie statique (.a, .lib)

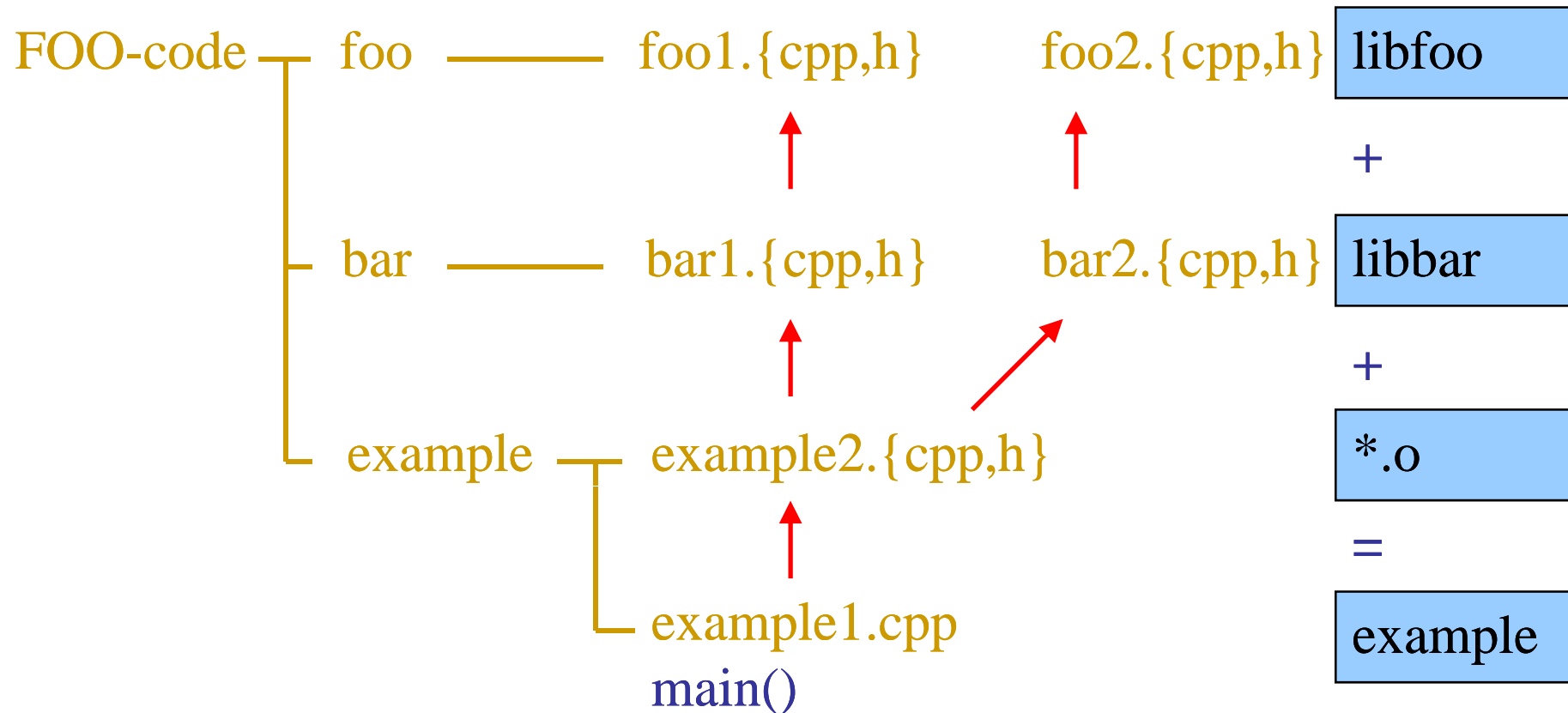
- Pendant l'édition de liens (production de l'exécutable), copie dans l'exécutable des objets requis de la librairie statique
- La librairie statique est ignorée durant l'exécution

Librairie dynamique (.so, .dll)

- Pendant l'édition de liens, enregistre dans l'exécutable une dépendance vis à vis de la librairie
- Au niveau de l'exécution, recherche la librairie dynamique, la charge dans la mémoire puis exécute le binaire



3. Un exemple de projet



4. Difficultés

Création des bibliothèques

- Très dépendant de la plate-forme

Gestion des dépendances

- La modification d'un fichier ne doit pas conduire à un *build* complet
- Une gestion optimale des dépendances est nécessaire pour optimiser la durée du *build*

Modularité

- L'ajout ou le renommage d'un fichier doit être aisé

Automatisation du *build*

- Prend toute son importance sur de « grands » projets
- Fortement dépendant de l'hétérogénéité des plates-formes cible
- Il existe des outils facilitant le build: Scons, make, Autotools, CMake



5. Automatisation du *build*: Make

- ❑ Permet d'automatiser le *build*
- ❑ Utilise un fichier Makefile décrivant le projet
 - Gestion des dépendances
 - Commandes permettant de générer des cibles à partir de dépendances
- ❑ Cible: un fichier à générer
- ❑ Dépendances: fichiers nécessaires pour générer une cible
- ❑ Commande: une commande à lancer pour créer une cible lorsque toutes les dépendances sont disponibles
- ❑ Éléments de syntaxe des fichiers Makefile

```
cible: dep1 dep2 dep3  
[tabulation]commande
```

5. Automatisation du *build*: Make

Exemple de fichier Makefile avec création de bibliothèques

Edition
des liens

```
OBJ_FOO = foo/foo1.o foo/foo2.o
LIB_FOO = foo/libfoo.a
OBJ_BAR = bar/bar1.o bar/bar2.o
LIB_BAR = bar/libbar.a
```

```
example/example: example/example1.o example/example2.o $(LIB_BAR) $(LIB_FOO)
    g++ -o $@ $^
```

```
$(LIB_FOO) : $(LIB_FOO)( $(OBJ_FOO) )
    ranlib $(LIB_FOO)
```

```
$(LIB_FOO)(%.o): %.cpp
    g++ -Ibar -Ifoo -Iinclude -c $*.cpp -o $*.o
    (ar r $@ $%; rm $%)
```

```
$(LIB_BAR) : $(LIB_BAR)( $(OBJ_BAR) )
    ranlib $(LIB_BAR)
```

```
$(LIB_BAR)(%.o): %.cpp
    g++ -Ibar -Ifoo -Iinclude -c $*.cpp -o $*.o
    (ar r $@ $%; rm $%)
```

...

Compilation
+
Création libfoo.a

Compilation
+
Création libbar.a



Lagadic

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



centre de recherche
RENNES - BRETAGNE ATLANTIQUE

5. Automatisation du *build*: Make

Exemple de fichier Makefile avec création de librairie: suite...

```

...

%.o: %.cpp
    g++ -Ibar -Ifoo -Iinclude -c $*.cpp -o $*.o

example/example1.o: example/example1.cpp example/example2.h
example/example2.o: example/example2.cpp example/example2.h \
    bar/bar1.h bar/bar2.h

$(LIB_BAR)(bar/bar1.o): bar/bar1.cpp bar/bar1.h foo/foo1.h
$(LIB_BAR)(bar/bar2.o): bar/bar2.cpp bar/bar2.h foo/foo2.h

$(LIB_FOO)(foo/foo1.o): foo/foo1.cpp foo/foo1.h
$(LIB_FOO)(foo/foo2.o): foo/foo2.cpp foo/foo2.h

```

Compilation

Gestion des dépendances



5. Automatisation du *build*: Make

Compilation

%make

OU: make -f Makefile

```
g++ -Ibar -Ifoo -Iinclude -c example/example1.cpp -o example/example1.o
g++ -Ibar -Ifoo -Iinclude -c example/example2.cpp -o example/example2.o
```

```
g++ -Ibar -Ifoo -Iinclude -c bar/bar1.cpp -o bar/bar1.o
(ar r bar/libbar.a bar/bar1.o; rm bar/bar1.o)
ar: creating bar/libbar.a
g++ -Ibar -Ifoo -Iinclude -c bar/bar2.cpp -o bar/bar2.o
(ar r bar/libbar.a bar/bar2.o; rm bar/bar2.o)
ranlib bar/libbar.a
```

Création
libbar.a

```
g++ -Ibar -Ifoo -Iinclude -c foo/fool.cpp -o foo/fool.o
(ar r foo/libfoo.a foo/fool.o; rm foo/fool.o)
ar: creating foo/libfoo.a
g++ -Ibar -Ifoo -Iinclude -c foo/foo2.cpp -o foo/foo2.o
(ar r foo/libfoo.a foo/foo2.o; rm foo/foo2.o)
ranlib foo/libfoo.a
```

Création
libfoo.a

```
g++ -o example/example example/example1.o example/example2.o
bar/libbar.a foo/libfoo.a
```

Edition
des liens



Lagadic

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE



centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Conclusion

Makefile écrits à la main peu adaptés à de grands projets

- Langage très bas niveau
- Difficile de générer des fichiers dynamiquement
- Difficile de détecter dynamiquement des librairies tierces

Nécessité d'utiliser des outils de *build* de plus haut niveau

- Scons, Ant, Autotools, CMake, ...
- Le choix de ces outils dépend de la plate-forme cible
- De la taille du projet
- Du langage de programmation

