

12 juillet 2012

Convergence des pratiques

DÉVELOPPEURS

Convergence des pratiques Développeurs

Propositions

«Auteur principal» : Gabriel CHANDESRIS

Brouillon
(draft)

Brouillon
(draft)

Table des matières

Introduction	1
Auteurs de ce document	1
Historique de ce document	1
1 Conventions de codage	2
1.1 Généralités Java (et autres...)	2
1.2 Quelques liens...	3
1.3 Orientation C / C++	4
2 Documents, modèles / templates, forme et rendu ?	5
2.1 Généralités	5
2.2 Méthodes de travail de rédaction / bureautique	6
3 Projets et Développement logiciels	7
3.1 Gestion de projet	7
3.1.1 Définition des besoins??	7
3.1.2 Solutions??	7
3.2 Prototypage	7
3.3 Quelques outils génériques...	8
3.4 Besoins et solutions éventuelles	9
3.5 Outils de manipulation SVN	9
3.6 Petit éventail des outils de construction («builder») libres	10
3.6.1 Sommaire	10
3.6.2 Un «builder» ?	10
3.6.3 Le vénérable Make	11
3.6.4 Le peu connu OMake	11
3.6.5 Un challenger scons	12
3.6.6 Celui de Google : Ninja	12
3.6.7 L'univers Java	13
3.6.8 Les surcouches	14
3.7 Documentation	16
3.7.1 Interne au code source – commentaires	16
3.7.2 Externe au code source	16
4 Ergonomie des Interfaces Homme-Machine (IHM)	17
4.1 Informations sur l'ergonomie des IHM	17
4.2 Quelques éléments complémentaires...	18
5 Veille information	20
5.1 «Zones de recherche»	20
5.1.1 Bases de Données Biologiques : articles, termes médicaux...	20
5.1.2 Termes à utiliser	20
5.2 Outils automatiques	20
5.3 Idées de développement (en interne)	20
5.3.1 Représentation données récoltées	20
5.3.2 Outil de récolte d'information	20
6 Idées pour un Framework interne	22
6.1 Idées générales	22
6.1.1 «Problématique(s)»	22
6.1.2 Reprise paradigme MSR	22
6.1.3 Reprise RFLP	22

6.1.4	Interfaçage graphique	22
6.1.5	Éléments client-serveur	22
6.1.6	Librairies	22
6.1.7	Branchement des prototypes et autres tests, librairies	22
6.2	Convergences et «Modèles-Patterns-Templates»	22
6.2.1	Conventions de codages et exemples de définitions de classes, attributs, méthodes, fonctions...	22
6.2.2	Design-Pattern et autres méthodologies...	22
6.2.3	Définitions de méthodologies de constructions de librairies (DLL par exemple)...	22
6.2.4	Définitions de tests unitaires et tests plus généraux...	22
7	Section	23
7.1	Sous-section	23
7.1.1	sous sous sous section	23
7.1.2	sous sous sous section	23
7.2	Encore une sous-section	24
Abbréviations		25
Bibliographie		26

Table des figures

1	Une belle image	24
---	---------------------------	----

Liste des tableaux

1	Historique et versions de ce document	1
2	Termes [Publication Type] de PubMed	21
3	Un tableau référencé	23

Brouillon
(draft)

Introduction

[...]

L'objectif de ce document est de fournir des *propositions* pour des pratiques communes aux développeurs chez *INTERNE*, tiré de l'expérience de chacun(e).

Ceci dans l'idée d'en discuter et de définir des pratiques communes (conventions de codages, outils de développement, documentation...) et de donner des exemples pour chacune.

Un ensemble de reources est listé dans ce document (bien sûr amené à évoluer!!).

[...]

Auteurs de ce document

Gabriel Chandesris (GCH)

...

Historique de ce document

Date	Version	Status	Modifications	Auteur(s)
05/09/2011	0.1	En cours	Première version	GCH
29/09/2011	0.1.1	En cours	Corrections et commentaires de GCH et Autre	
27/01/2012	0.2	En cours	Reprise de ce projet, précisions...	GCH et Autre

TABLE 1 – Historique et versions de ce document

1 Conventions de codage

Autre : Ok bien sûr pour les conventions "standards" de chaque langage (java != C++) Ne pas oublier que CAA nous impose déjà de sévères conventions de codage.

1.1 Généralités Java (et autres...)

Juste pour info (plus pour la suite/ l'avenir : le code source est aussi fait pour être relu par des êtres humains). *Eclipse* facilite les choses (coloration des mots spéciaux, re-indentation automatisée, mais un autre éditeur / environnement peut être utilisé).

«Consignes» applicables précisément à Java (qui dispose d'une expérience acquise sur d'autres langages ; pour lesquels il y a quelques variantes il y a également de légères variantes C / C++, PHP, Perl, Python, Ruby...)

Nommage des packages : suffixe + nom entreprise / institution + nom projet + nom(s) sous-partie (séparation par des points) : correspond à la hiérarchie des répertoires contenant les sources (classes '.java'), puis les éléments compilés pour les binaires.

Nommage des classes : première lettre en majuscule, puis 'CamelCase' si nom composé

Nommage des attributs d'instances de classe // d'objet : minuscules ('CamelCase' pour le reste si vraiment long, mais pas conseillé que ce soit long)

++ utilisation du type `< nomInstanceDeClasse > . < attribut >`

// ne pas négliger l'utilisation de 'this' en interne dans une méthode !!

Nommage des attributs statiques de classe : Majuscules complètement, séparation composition par '_' (seul cas !!) ; utiliser le moins possible, [cas particulier des Design-Pattern ou de certaines propriétés couramment utilisées]

++ utilisation du type `< NomDeLaClasse > . < ATTRIBUT_STATIC >`

Nommage des méthodes : comme les attributs d'instance de classe // d'objet

Commentaires \Rightarrow Eclipse aide beaucoup (Java-Style : '/' * '*' + < Enter > et compléter à partir de la ligne suivante, TOUT le commentaire (y compris parties générées)...

Descriptions pour javadoc (ou équivalent dans d'autres langages) : packages, classes, attributs, constructeurs et méthodes...

Descriptions en interne (si pas évident à une relecture : parties d'une méthodes ou cas particuliers, explications pertinentes à un endroit précis).

Ne pas hésiter à espacer, indenter, utiliser les accolades (toujours même si un seul élément dans le 'if' / 'else' / 'while' / 'for'...

\Rightarrow PENDANT l'écriture du code (notamment les boucles 'for', les espaces autour des opérateurs, les attributions de valeurs aux variables, passages à la ligne pour des opérations longues à lire / appréhender, passer à la ligne et indenter si beaucoup d'arguments dans une fonction / méthode (voire regrouper dans la liste par type)...

\Rightarrow par exemple

```
for (int i = 0 ; i < limit ; i++) { ; }
```

plutôt que

```
for(int i=0;i<limit;i++){;}
```

\Rightarrow si plusieurs variables attribuées à la suite : "équilibrer" au niveau du '=' ; par exemple :

```
int val      = 0;
double var   = 0.0;
Object toto  = new Object();
```



1.2 Quelques liens...

<http://cyberzoide.developpez.com/java/javastyle/> JavaStyle
http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html Java Language Specification
<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html> How to Write Doc Comments for the Javadoc Tool
<http://blog.lecacheur.com/2005/01/19/analyse-de-la-qualite-du-code-source-java/> Analyse de la qualité du code source Java
<http://case.oncle-tom.net/2008/conventions-de-programmation-necessaire-maturite/> Convention(s) de programmation : nécessaire maturité
 http://fr.wikipedia.org/wiki/Notation_hongroise Notation hongroise (Wikipedia FR) : utilisée notamment en C
 <http://pear.php.net/manual/fr/standards.php> Convention PEAR
<http://hugo.developpez.com/tutoriels/genie-logiciel/utilite-normes-codage/> Utilité des normes de codage (voir les références en bas de page).
<http://www.dng-consulting.com/blogs/index.php/2010/01/30/plugin-eclipse-pour-la-revue-de-code> Plug-in Eclipse pour conventions de code
http://fr.wikipedia.org/wiki/Apache_Maven Apache **Maven** (Wikipedia FR), outil de gestion de projets et production (avec entre autres choses une analyse de la qualité du code : commentaires, cyclicité...)
<http://prendreuncafe.com/blog/post/2008/07/23/Mes-conventions-de-codage> Un exemple pour PHP

Ce qu'il faut retenir d'une convention c'est qu'elle explicite des règles de développement :

- sur le nommage des éléments
- sur l'indentation des éléments
- sur les structures de contrôle (if, else, tout ça quoi)
- sur la syntaxe des commentaires
- sur la syntaxe de la documentation (le code auto-documenté c'est bon !)
- sur l'organisation des fichiers, éventuellement

Critères de qualité – Voici une liste non-exhaustive de critères de qualité d'un code source :

- lisibilité
- uniformisation des conventions de codage
 - nommage
 - indentation
 - organisation
 - documentation
- couplage minimum (module indépendants)
- complexité cyclomatique minimum
- commentaires Javadoc maximum

1.3 Orientation C / C++

[http://www.commentcamarche.net/forum/affich-2607701-c-convention-de-codage-linux C++ et Linux](http://www.commentcamarche.net/forum/affich-2607701-c-convention-de-codage-linux-C++-et-Linux)

Pour moi la seule chose importante que ce soit codé proprement (indenté, avec des vrais noms de variables, des lignes pas trop longue (si possible < 80 caractères)), que ça compile, et *si possible sans warning*.

⇒ `gcc -W -Wall -o plop.exe plop.c`

Alors quelques petites remarques.

- Le pedantic c'est bien mais il me semble qu'il fait la tête si tu fais des commentaires avec `'/'`. Personnellement je n'utilise pas cette option.
- Pour les conventions de nommage, en général on écrit tout en minuscule en utilisant des `'_'` (contrairement au java ou les noms de classes commencent par une majuscule) :
 ex : c++ : `ma_classe`, `ma_fonction`
 ex : java : `MaClasse`, `maFonction`
 On remarquera que QT suit pourtant le second type de notation donc en fait c'est une convention discutable.
- Personnellement quand je déclare une structure une classe ou une structure je termine toujours son nom par `_t`. ex : `ma_classe_t`
- Quand je fais une boucle avec un iterator, je renomme toujours l'objet itéré avec une const référence, sauf si la boucle est très simple pour que ce soit plus lisible (sauf quand la boucle est vraiment très simple) :

```
int main() {
    std::map<std::string,int> m;
    m["janvier"] = 1;
    m["fevrier"] = 2;
    //...
    std::map<std::string,int>::const_iterator
    mit(m.begin()),
    mend(m.end());
    for( ; mit != mend ; ++mit ) {
        const std::string & mois = mit->first;
        const int & numero = mit->second;
        std::cout << mois << ' ' << numero << std::endl;
    }
    return 0;
}
```

- De manière générale utiliser le moins possible les using namespace pour plus de lisibilité. Ça alourdit un peu les types mais au moins on sait de quoi on parle et il n'y a pas d'ambiguïté. En particulier, ne jamais utiliser un using namespace dans un header (.hpp, .h)
- Après plus sur le style de codage, en C++ on n'a pas besoin de malloc et free (il y a new et delete), de cast (il y a les `static_cast`, les `dynamic_cast`...).
- Utiliser le moins possible les new, les delete, les pointeur car la plupart du temps on peut se débrouiller juste avec le constructeur et les références, un peu comme le java. De même par le jeu des héritages le cast peut être très souvent évité (et devrait l'être autant que possible).
- Ne passer à une fonction un pointeur que si c'est bien le pointeur qui est manipulé. Si c'est juste pour éviter d'empiler un gros objet en paramètre, ou pour que celui-ci soit modifiable par la fonction, utiliser une référence.
- Verrouiller le code au maximum.
- Bien mettre en `private` // `protected` ce qui doit l'être, mettre les const pour les invariants etc...



2 Documents, modèles / templates, forme et rendu ?

2.1 Généralités

Quelques questions et problématiques :

Rendu / forme des documents ?

Facilité d'utilisation ?

Gestion du contenu ?

Gestion des modèles / templates ? (++ problématique de la forme et des modifications du document)

Rédaction et suivi des modifications :

<http://forum.judgehype.com/judgehype/Discussionsgenerales/> —

— [latex-suivi-modifications-sujet_244139_1.htm](#) Dans Word, mais aussi dans TeX et LyX !!

<http://forum.hardware.fr/hfr/WindowsSoftware/Tutoriels/> —

— [topic-unique-latex-sujet_152676_94.htm](#) Même problématique ici...

<http://www.projet-plume.org/fiche/lyx> LyX : éditeur WYSIWYG (La)TeX avec suivi des modifications

<http://forum.ubuntu-fr.org/viewtopic.php?id=403960> Suivi des révisions avec L^AT_EX

<http://fr.w3support.net/index.php?db=so&id=888347> — Sinon SVN ++ L^AT_EX est une bonne option (sûrement meilleure, place sur le serveur mais gain au final dans le document)

— http://yann.moreire.free.fr/IMG/pdf/svn_for_latex.pdf Utilisation de SVN pour LaTeX

http://www.siteduzero.com/tutoriel-3-258578-qu-est-ce-que-latex.html#ss_part_2 Comparatif Word / L^AT_EX

⇒ avantage de L^AT_EX : on peut taper son texte au kilomètre sans être gêné par le soucis permanent de mise en forme (et parfois les bugs)

⇒ limitation : forme lors compilation (pas de soucis dans la plupart des cas, sauf cas tordus...) ; LyX pour éditer visuellement...

Accessoirement, il y a le coût des licences M\$ Office (quoique OpenOffice / LibreOffice soient aussi gratuits que L^AT_EX, l'apprentissage minimum est toujours présent quelque soit l'éditeur ou la suite bureautique utilisé... Il faut penser également au maintien de la documentation dans le temps!).

⇒ Orientation compatibilité avec partenaires / autres correspondants

2.2 Méthodes de travail de rédaction / bureautique

L'idée générale ici est de faciliter tant l'écriture des documents que leur historique (*versioning*)... Sachant que la méthodologie générale décrite ici est applicable partout...

À préciser selon recommandations «officielles». Quelques idées dans la catégorie «*façons de faire*» / méthodologie, dans l'ordre :

1. Taper le texte *au kilomètre*, en indiquant les titres / sous-titres / ..., images (et positions) sans se soucier de la forme.
2. Insérer les images (avec légendes), références bibliographiques.
3. Gérer le format du document (formes des titres, sous-titres...) et appliquer.
4. Insérer les tables (tables des matières / figures / tableaux...).

Certains modes d'édérations n'étant pas forcément aisé pour tout un chacun (Word et ses révisions, équivalent dans OpenOffice et assimilés, L^AT_EX...), il est parfois bien utile de repartir du «texte brut», puis de repartir dans la suite, une façon de faire pourrait être de :

1. Utiliser un répertoire / dossier de travail pour un document, appelé à contenir un fichier texte, un répertoire d'images, le document final...
2. Enregistrer le contenu textuel dans un fichier texte (?? document Word ??, utiliser Notepad, Notepad++, Jedit, Geany...).
3. Récupérer les images dans un répertoire associé.
4. Modifier le texte et les images à volonté.
5. Recomposer le document final en suivant la procédure précédente.

Cette méthode est quelque peu contraignante mais largement automatisable (inspirée de celle de L^AT_EX), sachant que l'on peut générer des *proto-document Word* avec L^AT_EX... L'objectif étant multiple et recouvre notamment l'idée de pouvoir reprendre facilement les documents dans un autre format ou version de logiciel (dans les suites bureautiques par exemple), sachant que c'est ce format qui fera foi de façon générale (en espérant que l'on puisse récupérer les données depuis ce format...).

NOTE WORD 2010 / FICHIERS [.DOCX] : les documents générés sont en fait des fichiers compressés au format ZIP [.zip] contenant un fichier XML avec du texte et également un répertoire d'images (en général au format Bitmap [.bmp], voire JPEG [.jpeg|.jpg] ou PNG [.png] facilement ré-utilisables. Pour accéder à ces données en-dehors de Word, il suffit de remplacer l'extension [.docx] du document par [.zip] pour accéder à tout cela...

3 Projets et Développement logiciels

3.1 Gestion de projet

3.1.1 Définition des besoins ??

- Définir des tâches / sous-tâches...
- Wiki ?
- Forum
- Rôles / Utilisateurs
- Liens avec projet(s)
- ...

3.1.2 Solutions ??

Redmine

<http://www.redmine.org/> Site officiel

<https://fr.wikipedia.org/wiki/RedMine> sur Wikipedia FR (voir catégories / listes)

<https://en.wikipedia.org/wiki/Redmine> sur Wikipedia EN (voir catégories / listes)

...

3.2 Prototypage

?? Un «framework» interne ??

«Propriétaire-like»

Facile à construire / utiliser

Balsamiq Mockup ??

<http://www.balsamiq.com/products/mockups> Site officiel

...

Brouillon
(draft)

3.3 Quelques outils génériques...

<http://technet.microsoft.com/fr-fr/sysinternals/> Quelques outils utiles pour FenetreS / Windows...
<https://www.teamviewer.com/> TeamViewer (accès à distance : Windows, Mac OS X, Linux... (client sous Android également)).

<https://www.eclipse.org/> ECLIPSE, développement (sous tous systèmes d'exploitation...).

<http://www.cygwin.com/> CygWin *Si on veut compiler pour Unix / Linux en étant sous Windows*

— a collection of tools which provide a Linux look and feel environment for Windows.

— a DLL (cygwin1.dll) which acts as a Linux API layer providing substantial Linux API functionality.

<http://www.mingw.org/> MinGW (Minimalist GNU for Windows) *à associer avec CygWin ou Eclipse pour une compilation en C / C++*

— **MinGW**, a contraction of "Minimalist GNU for Windows", is a minimalist development environment for native Microsoft Windows applications.

— **MinGW** provides a complete Open Source programming tool set which is suitable for the development of native MS-Windows applications, and which do not depend on any 3rd-party C-Runtime DLLs. (It does depend on a number of DLLs provided by Microsoft themselves, as components of the operating system; most notable among these is MSVCRT.DLL, the Microsoft C runtime library. Additionally, threaded applications must ship with a freely distributable thread support DLL, provided as part of **MinGW** itself).

— **MinGW** compilers provide access to the functionality of the Microsoft C runtime and some language-specific runtimes. **MinGW**, being Minimalist, **does not, and never will, attempt to provide a POSIX runtime environment for POSIX application deployment on MS-Windows**. If you want POSIX application deployment on this platform, please consider Cygwin [<http://www.cygwin.com/>] instead.

— Primarily intended for use by developers working on the native MS-Windows platform, but also available for cross-hosted use, (see note below – you may need to follow the "read more" [<http://www.mingw.org/Welcom> link to see it), **MinGW** includes :

- A port of the GNU Compiler Collection (GCC), including C, C++, ADA and Fortran compilers;
- GNU Binutils for Windows (assembler, linker, archive manager)
- A graphical and a command-line installer for **MinGW** and **MSYS** deployment on MS-Windows

— **MSYS**, a contraction of "Minimal SYStem", is a Bourne Shell command line interpreter system. Offered as an alternative to Microsoft's cmd.exe, this provides a general purpose command line environment, which is particularly suited to use with **MinGW**, for porting of many Open Source applications to the MS-Windows platform; a light-weight fork of Cygwin-1.3, it includes a small selection of Unix tools, chosen to facilitate that objective, and using it is a necessary prerequisite for building **MinGW**PORTs.

— **mingwPORT** are user contributed additions to the **MinGW** software collection. Rather than providing these "add-ons" as precompiled binary packages, they are supplied in the form of interactive Bourne shell scripts, which guide the end user through the process of automatically downloading and patching original source code, then building and installing it. Users who wish to build any application from a **MinGW**PORT must first install both **MinGW** and **MSYS**.

— ...

— **strawberry**...

3.4 Besoins et solutions éventuelles

- OUTIL + Visual Studio + Outlook...
 - Windows exclusive (au moins pour compilations et tests...)
 - Consultation courrier électronique / agendas...
 - Visual and Enovia Studio.
- sinon alternatives pour la plupart des autres outils, facilité d'usage...
 - Linux et «suites logicielles» (idem Mac OS X)
 - Eclipse et ses plug-in...
 - Courrier : Evolution / Thunderbird avec lien sur Serveur Outlook / Microsoft Exchange (package *evolution-mapi* par exemple...)

3.5 Outils de manipulation SVN

À utiliser ensembles si compatibles) :

- subversion eclipse + windows [<http://www.jmdoudoux.fr/java/dejae/chap015.htm>]
- subversive eclipse [<https://www.eclipse.org/subversive/>]
- subclipse [<http://subclipse.tigris.org/>] intégré à Eclipse : icônes++
- tortoiseshvn [<http://tortoiseshvn.tigris.org/>] intégré à l'explorateur de fichier (surtout pour windows??)

Brouillon
(draft)

3.6 Petit éventail des outils de construction («builder») libres

Ligne de commande Petit éventail des outils de construction («builder») libres

<http://linuxfr.org/news/petit-eventail-des-outils-de-construction-builder-libres>

Posté par Barret Michel (page perso, jabber id) le 05/09/11 à 10 :20. Modéré par patrick_g. Licence CC by-sa – 5-sept.-2011

Je vous propose dans cette dépêche de revenir sur la panoplie d'outils de construction qui s'offre à nous (c'est à dire les outils permettant d'automatiser les étapes de preprocessing, compilation, éditions des liens, etc). Je ne cherche pas à faire un comparatif, mais juste à les décrire pour en faire ressortir les avantages et inconvénients ainsi que les cas d'utilisation. Cette dépêche peut être vue comme un état de l'art allégé des outils de construction libres.

Je tiens à remercier les contributeurs de cette dépêche :

GeneralZod ; tiennou ; NedFlanders ; claudex

Ce sont eux qui ont écrit la majeure partie de cette dépêche et qui l'ont améliorée et complétée grâce à leurs connaissances et au temps qu'ils y ont consacré. Cette dépêche a pour objectif de faire découvrir ou redécouvrir des outils de constructions. Si vous en connaissez d'autres n'hésitez pas à en parler en commentaire.

<http://omake.metaprl.org/index.html> OMake

<http://www.scons.org/> SCons

<https://maven.apache.org/> Maven

<https://github.com/martine/ninja> Ninja sur github

<https://ant.apache.org/> Ant chez Apache

<http://www.cmake.org/> CMake

<http://buildr.apache.org/> buildr chez Apache

<http://ant.apache.org/ivy/> Apache Ivy

<http://www.gradle.org/> Gradle

http://fr.wikipedia.org/wiki/GNU_Make GNU Make sur Wikipédia

3.6.1 Sommaire

- Un «builder» ?
- Le vénérable Make
- Le peu connu OMake
- Un challenger scons
- Celui de Google : Ninja
- L'univers Java
 - L'historique Apache Ant
 - L'usine à build Apache Maven
 - Les autres (buildr, gradle...)
- Les surcouches
 - CMake
 - Les autotools

3.6.2 Un «builder» ?

Nous avons dans nos dépôt un grands nombre d'outils qui ont chacun leur philosophie. Le reste de la dépêche s'attelle à en décrire un certains nombre parmi les plus connus. Un builder est un outil qui permet de construire un logiciel (et plus si affinité) à partir des sources. Il doit être le chef d'orchestre de la chaîne de fabrication du logiciel (le compilateur, l'éditeur de liens, etc).

Les critères importants de ce genre d'outils sont la facilité d'utilisation : pour le développeur et pour l'utilisateur. La performance : lorsqu'un projet est gros, le temps de création du logiciel peu devenir très important et la vitesse



du builder peut avoir un impact important. La flexibilité : peut-on en faire ce que l'on veut avant et après chaque étape ? Gère-t-il n'importe quel langage ?

3.6.3 Le vénérable Make

make est l'outil de base pour beaucoup. C'est un outil très vieux, mais encore beaucoup utilisé. Il est le seul outil de la liste de cette dépêche à faire partie de POSIX. Sous nos distributions GNU/Linux, c'est généralement GNU Make qui est utilisé.

Make est une sorte de langage Rule Based Programming [<https://linuxfr.org/users/montaigne/journaux/des-pa>] quand lequel on spécifie des règles sous la forme :

```
cible: dependance
cmd
```

Avec :

- **cible** : fichier ou pattern de fichier que l'on cherche construire
- **dépendance** : liste des fichiers nécessaires pour construire la cible
- **cmd** : commande pour produire la cible

Les avantages de cette approche sont :

- la **souplesse** : on utilise make pour tout et n'importe quoi aujourd'hui (créer un PDF, latex ou lout, compiler dans tout type de langage, etc.)
- la **simplicité** : si vous savez construire à partir de votre ligne de commande vous saurez le faire avec make
- **rapidité** : il construit un arbre de dépendance et exécute une parallèle ce qui peut l'être (aujourd'hui tout les builders le font)

En revanche, il possède aussi un certain nombre d'inconvénients :

- **dépendances** : on doit gérer à la main les dépendances entre les fichiers, plus le projet devient gros plus ça devient lourd à gérer ;
- **manque de portabilité** : on utilise des commandes, généralement on ne se gêne pas pour utiliser les options GNU de nos outils favoris, il faut aussi que ces outils soient installés sur la machine, etc. ;
- **manque de réutilisabilité** : quand on se met à beaucoup l'utiliser, on a tendance à recopier des blocs de Makefile car rien n'est mutualisé.

Ce dernier point est tout de même à relativiser, GNU Make intègre beaucoup de règles par défaut qui simplifient son utilisation. Par exemple, si vous avez un simple fichier coucou.c que vous souhaitez compiler, il vous suffit de taper cette commande (sans avoir créé de Makefile au préalable) pour créer l'exécutable :

```
\$ make coucou
cc coucou.c -o coucou
```

Si vous souhaitez passer des arguments au compilateur :

```
\$ export CFLAGS='-g'
\$ make coucou
cc -g coucou.c -o coucou
```

3.6.4 Le peu connu OMake

Il utilise une syntaxe similaire à Make mais offre quelques fonctionnalités supplémentaire :

- La gestion du projet dans plusieurs répertoires ou dans une hiérarchie de répertoire ;
- Une analyse automatique et rapide des dépendances, basée sur MD5 ;
- Lorsqu'une cible est spécifiée dans la ligne de commande, elle est automatiquement considérée comme obsolète et reconstruite ;

- Il est complètement scriptable et une bibliothèque est fournie pour les langages C, C++, OCaml, et LaTeX. Il est donc possible d'avoir une seule ligne pour le projet :
`.DEFAULT: \$(CProgram prog, foo bar baz)`

Cette ligne suffit pour construire la cible `prog` à partir des fichiers `foo.c`, `bar.c` et `baz.c`. OMake va aussi chercher les dépendances implicites comme les includes des fichiers C.

- Les règles qui construisent plusieurs fichiers en une fois sont complètement prises en charge ;
- Il fonctionne sur Linux, Windows, Cygwin et Mac OS ;
- Des fonctions telles que `grep`, `sed` et `awk` sont directement incluses dans le programme, cela permet d'améliorer la portabilité sous Windows ;
- Il est possible de surveiller le système de fichier en continu pour relancer la construction dès qu'un fichier est modifié ;
- Il dispose d'un interpréteur de commande, `osh`, qui permet l'utilisation interactive.

3.6.5 Un challenger scons

Scons est écrit en python, et les scripts `scons` (fichiers `SConstruct` et `SConscript`) sont eux-même en python. Ceci lui procure une très grande souplesse. La contrepartie de cette souplesse est une certaine complexité, `scons` fait un peu moins "clef-en-main" que certains autres outils de construction.

Avantages de `scons` :

- **c'est du python** : c'est un vrai langage, pas un truc tout bancal, ça roxe
- sait extraire out-of-the-box les **dépendances** des sources de nombreux langages (C, C++, Java, Fortran, etc.)
- garantie d'avoir toujours des **constructions correctes**, ce qui est assez confortable. Scons garde trace du hash du contenu de chaque fichier apparaissant dans l'arbre de dépendances.
- lance les compilations en **parallèle**
- killer feature : `scons` **cache** le résultat chaque opération (compilation, liaison, etc.), un peu comme `ccache`, mais pour toutes les opérations et tous les langages.
- gère de base les chaînes de compilation sur les systèmes d'exploitation majeurs (outils GNU sous Linux, Visual Studio sous Windows, XCode sous Mac OS X), tout ceci étant bien entendu totalement paramétrable.
- range tous les produits de la compilation dans un "build directory", pratique pour séparer les choux et les carottes.

Inconvénients :

- relativement **complexe** à prendre en main, et plutôt bas niveau
- plutôt **lent** par rapport à la concurrence, en partie à cause de python, et en partie à cause de la gestion très rigoureuse des dépendances.

Une comparaison avec les autres outils est disponible sur [<http://www.scons.org/wiki/SconsVsOtherBuildTools>]

Un dérivé de `scons` qui semble avoir le vent en poupe est `waf` [<https://code.google.com/p/waf/>]. Il comble les lacunes de `scons` (complexité et lenteur) en proposant des améliorations tel qu'une sortie en couleur et un meilleur support de la compilation parallèle.

3.6.6 Celui de Google : Ninja

Ninja est le moteur de production créé par l'équipe de développement de Google Chrome. Ceux-ci utilisaient Make mais trouvaient que la génération de l'arbre des dépendances était bien trop lent (le temps avant de commencer serait d'une minute).



Ils ont choisi de développer leur propre outil (pourquoi donc patcher le logiciel existant ?). Celui-ci est plus simpliste que Make car il ne possède aucune logique interne liée aux langages (contrairement à make qui possède des règles prédéfinies). Néanmoins, il a une syntaxe qui est un peu plus flexible.

L'utilisation de ninja passe par la création d'un fichier build.ninja. En voici un exemple tiré de la documentation officielle :

```
cflags = -Wall

rule cc
  command = gcc $cflags -c $in -o $out

build foo.o: cc foo.c
```

Cet exemple, bien que simpliste permet de voir l'usage basique du logiciel. Voici une petite explication de la syntaxe :

```
cflags = -Wall

rule cc
  command = gcc $cflags -c $in -o $out
```

Déclare une règle cc :

- \$in corresponde aux fichiers d'entrée
- \$out au fichier de sortie

```
build foo.o: cc foo.c
```

Ceci va invoquer la règle cc avec foo.c comme variable \$in et foo.o comme variable \$out.

3.6.7 L'univers Java

L'historique Apache Ant

Le premier moteur de production orienté Java développé par la fondation Apache. Il repose sur la notion de tâches et a la particularité d'utiliser un format XML pour son fichier de configuration. On peut étendre les fonctionnalités d'Ant en écrivant des nouvelles tâches. Bien qu'écrit en Java, il est utilisable avec d'autres langages.

Avantages :

- une riche collection d'extensions via antcontrib ;
- portable ;
- excellente intégration aux outils Java existants (tests unitaires, qualité de code etc...).

Inconvénients :

- XML ?
- Difficulté d'avoir un environnement standard (compilation, paquetages, etc) rapidement. On doit partir de zéro pour les tâches standards
- Pas d'héritage sur les fichiers build.xml et la composition est fastidieuse.

L'usine à build Apache Maven

Maven est une plateforme moderne de construction de projet, développé également par la fondation Apache. Attaché à la devise « Convention plutôt que configuration », Maven impose sa manière d'organiser les fichiers et un cycle de construction en plusieurs phases (les principales étant clean, compile, test, package, install, deploy). L'intérêt de maven est qu'il impose de la rigueur dans l'infrastructure du projet, apportant par la suite énormément de confort puisqu'on est à peu près sûr de pouvoir reconstruire le projet sur un nouvel environnement.

La gestion des dépendances est un point fort de maven. Chaque module est identifié par (au minimum) le triplet (groupId, artifactId, version). L'ajout d'une dépendance à un projet se fait donc juste en spécifiant le triplet correspondant. Maven se charge automatiquement de récupérer les dépendances sur les dépôts comme un gestionnaire de paquets le fait sur notre distribution GNU/Linux favorite. Maven est lui-même extrêmement modulaire et les nombreuses extensions disponibles seront résolues au besoin au moment de leur utilisation.

Un projet maven est décrit dans un fichier pom.xml contenant l'identification du projet, des métadonnées optionnelles mais intéressantes (licence, page web, description...), la liste des dépendances, des dépôts additionnels, la liste des extensions supplémentaires utilisées.

En résumé, les avantages :

- description standardisée d'un projet
- richesse des extensions disponibles
- plateforme modulaire et autonome, garantissant une utilisation rapide dans un nouvel environnement

Les inconvénients :

- la résolution des dépendances peut s'avérer un peu longue
- une connexion au dépôt de bibliothèques est nécessaire au moins pour la première fois, sinon le projet est inutilisable
- une description plus détaillées des inconvénients se trouve sur le blog de Sonatype : We're Used to the Axe Grinding [<https://www.sonatype.com/people/2009/05/were-used-to-the-axe-grinding/>]

Les autres (buildr, gradle...

La communauté Java - et plus particulièrement la communauté Apache - est très productive ces derniers temps en terme de builder. En plus de Ant et Maven, Apache compte plusieurs projets intéressants. Quelques-uns sont à souligner :

- Apache Ivy [<https://ant.apache.org/ivy/>] est un sous-projet d'Ant. Il vise à ajouter la gestion des dépendances (compatibles Maven) dans des projet gérés par Ant. Cela permet de simplifier la gestion des dépendances à un projet qui était déjà géré par Ant.
- Gradle [<http://www.gradle.org/>] celui-ci gère les projet via un DSL à la place de fichier XML, il vise notamment à être plus indépendant du langage que Maven, ainsi il pourra plus facilement travailler sur des projet Groovy ou Scala. Il utilise, comme Maven, la convention plutôt que la configuration ce qui lui permet de simplifier son utilisation.
- Apache Buildr [<https://buildr.apache.org/>] est fait pour être le plus simple possible. Il est aussi prévu pour gérer plus facilement Groovy et Scala. Voici un fichier montrant sa simplicité, il permet de compiler un projet my-app et de créer un jar :

```
define 'my-app' do
  project.version = '0.1.0'
  package :jar
end
```

3.6.8 Les surcouches

CMake



Contrairement aux outils présentés auparavant, CMake est un générateur de "projets". Il génère un ou des fichier(s) recette(s) qui seront manipulés par un moteur de production bas-niveau (make, nmake, ...) ou pouvant être ouverts par un environnement de développement (eclipse, XCode, Visual Studio, KDevelop, ...). Depuis son adoption par KDE, il connaît une grande popularité.

CMake fournit un langage dédié et un jeu de macros permettant de gérer un projet complet. Il est ainsi capable de retrouver seul les bibliothèques ou les utilitaires installés sur le système et ainsi s'abstraire de la plateforme de compilation.

Plus généralement, CMake fait partie de la suite de processus logiciel qualité Kitware qui comporte également :

- CTest : un outil de tests qui étend CMake. Il permet de récupérer le code source à partir d'un gestionnaire de version, de compiler ces sources, et d'exécuter les tests unitaires (CMake supporte GoogleTest, CXXTest, SQuish et s'intègre facilement aux autres systèmes). Il permet également de récupérer la couverture de code et d'exécuter les tests sous Valgrind. Il génère des rapports XML qui pourront être traités par un serveur d'intégration continue.
- CDash : serveur web de tests, associé à CTest, il fournit une plateforme d'intégration continue flexible.
- CPack : la boîte à outils pour générer des paquets très simplement. Il supporte les formats suivants : dpkg, rpm, tarballs, installeurs Mac (dmg, drag'n'drop, etc.), NSIS, Cygwin.

Avantages :

- multiplateforme
- permet de générer des projets pouvant être ouverts par un IDE
- robuste
- prise en charge de la compilation parallèle
- une collection de modules qui s'enrichit régulièrement

Inconvénients :

- un langage de script relativement pauvre comparé à python ou lua
- documentation éparse

Les autotools

Derrière la dénomination autotools se cache toute une famille d'outils GNU constituant un moteur de production complet mais également complexe :

- autoconf : un générateur de scripts shell à partir de macros m4 permettant de configurer les sources d'un projet (ex : tester la version d'une bibliothèque, la présence ou non d'une fonction etc...). C'est lui qui est responsable de la génération du fameux "configure".
- automake : générateur de makefile portables, il est écrit en perl.
- libtool : abstrait la génération des bibliothèques dynamiques.
- gnu make : bien évidemment le moteur GNU make pour exécuter le makefile résultant.

Avantages :

- facilite l'écriture de programmes multi-plateformes.
- simplifie le processus de compilation par l'utilisateur (le célèbre tryptique : `./configure && make && make install`)
- système de construction robuste et extrêmement modulaire.
- la plus large collection d'extensions parmi tout les moteurs de production de haut niveau présentés.

Inconvénients :

- complexe d'utilisation

- dépendance à la présence d'un shell bourne (ou compatible), m4, et perl
- relativement lent (même si il est possible d'améliorer ce point en passant par une construction non récursive, c'est-à-dire en utilisant un unique Makefile.am à la racine du projet)
- les différentes versions d'automake sont incompatibles entre elles
- re-générer le système de construction est une tâche relativement complexe, souvent laissé à la charge d'un script shell (autogen.sh)

3.7 Documentation

3.7.1 Interne au code source – commentaires

Les outils majeurs sont JAVADOC et quelques autres outils assimilés, comme DOXYGEN... Ceci quelque soit le code source, l'objectif est pour le moins d'avoir une *notation de commentaires normalisée / formelle* !! La nomenclature entre les deux outils précédemment est assez proche ; l'objectif est de générer ensuite une documentation lisible en-dehors du code source, dans un format standard (HTML, texte tabulé, wiki...). Le principe est l'utilisation d'un certain nombre de marqueurs de typages des données (*tags*) dans les commentaires de packages, classes, méthodes, attributs... Ceci afin d'indiquer des éléments pré-définis de documentation (polus spécifiques des commentaires généraux)

<http://java.dzone.com/articles/javadoc-or-doxygen> JavaDoc or Doxygen ?

<http://gushieblog.blogspot.com/2006/06/doxygen-versus-javadoc.html> Doxygen versus JavaDoc

<http://fr.wikipedia.org/wiki/Doxygen> Doxygen (Wikipedia FR)

3.7.2 Externe au code source

Voir pour cela les modèles (et *templates*) de documentation : cahiers des charges, spécifications, PER, soumission... La question du format d'échange et d'archivage des documents dans la durée se pose également (texte brut, L^AT_EX, document Word, PDF ou [texte + images] à partir des précédents formats...).

Brouillon
(draft)



4 Ergonomie des Interfaces Homme-Machine (IHM)

Quelques idées trouvées en février 2011, pour les projets / prototypes / spécifications, (on ne sait jamais, si on fait un *brainstorming* ou si on inclut ça dans les règles et convergences des pratiques. Pour la partie IHM, quelques inspirations possibles (à développer) :

Autre : des pointeurs intéressants.

<http://benfry.com/genetics/> Ben Fry genetics Visualisation (ergonomie IHM pour la bio...)

<http://www.genotyp.com/> Un bel exemple sur de la «génétique de polices de caractères».

4.1 Informations sur l'ergonomie des IHM

http://www.info.univ-tours.fr/~antoine/documents_enseignement/IHM_CM_chapII.pdf [pdf] –

http://www.enib.fr/~nedelec/docs/poly/ihm_ergo.pdf [pdf] –

<http://deptinfo.cnam.fr/Enseignement/CycleSpecialisation/IHM/annee56/Ergonomie.pdf> [pdf] Il y a des normes!!

— AFNOR 'Ergonomie et conception du dialogue homme-ordinateurs – Z67-110 Afnor PARIS, Janvier 1988

— AFNOR 'Définition des critères ergonomiques de conception et d'évaluation des produits logiciels' – Z67-133-1 Afnor PARIS, Décembre 1991

«En bref, du bon sens»

- prendre en compte distance et angle de vision (ISO1988)
- exprimer les différences par des couleurs très contrastées et similitudes par des couleurs de faible contraste (ISO 1988)
- formes différentes , couleurs différentes (ISO1988)
- prendre en compte le confort visuel (éviter les couleurs très éloignées dans le spectre)
- utiliser des couleurs saturées avec luminosité pour mettre en évidence et inversement
- discrimination pour les items : en fonction de la distance et de l'éloignement dans le spectre
- le rouge : plus proche, le bleu plus éloigné, couleurs chaudes : objets apparaissant plus grands
- la couleur : satisfait l'utilisateur

Pour la partie "représentation de données" / Sémiologie Graphique, des points de départ à partir desquels on peut définir quelques règles... Et aussi tous les articles connexes de ces pages...

https://secure.wikimedia.org/wikipedia/fr/wiki/Semiologie_graphique Sémiologie Graphique (Wikipedia FR)

[https://secure.wikimedia.org/wikipedia/fr/wiki/Jacques_Bertin_\(cartographe\)](https://secure.wikimedia.org/wikipedia/fr/wiki/Jacques_Bertin_(cartographe)) Jacques Bertin (Wikipedia FR)

https://secure.wikimedia.org/wikipedia/fr/wiki/Design_de_l'information Design de l'information (Wikipedia FR)

Bibliographie [5, 4, 1, 2, 3]

- * Bertin Jacques, Sémiologie graphique, Paris, Mouton/Gauthier-Villars, 1967.
- * Bertin Jacques, La Graphique et le traitement graphique de l'information, Paris, Flammarion, 1975.
- * Bertin Jacques, Sémiologie graphique. Les diagrammes, les réseaux, les cartes, Paris, École Des Hautes Études En Sciences Sociales, 1999.
- * Béguin Michelle, Pumain Denise, La représentation des données géographiques, Statistique et cartographie, Paris, Armand Colin, 2003.
- * Gilles Palsky et Marie-Claire Robic, « Aux sources de la sémiologie graphique », Cybergeog : European Journal of Geography, Colloque "30 ans de sémiologie graphique", article 147, 2000

4.2 Quelques éléments complémentaires...

<http://www.rankspirit.com/design-site-web.php> Design site Web, mais de façon générale : contrastes pour la lisibilité

http://linkmultimedia.over-blog.com/pages/ERGONOMIE_au_bureau-585435.html Ergonomie au Bureau

<http://www.mon-design-web.com/standards.php> 10 règles simples d'ergonomie du web (mais applicable ailleurs!!).

Brouillon
(draft)

Évitez les textes soulignés : sur le Web, un mot ou une expression soulignée signalent généralement l'existence d'un lien. N'utilisez jamais le soulignement pour d'autres raisons. Utilisez la couleur, la taille ou le style (gras et italique), mais ne soulignez pas une portion de texte pour en signaler l'importance.

Largeur de colonne

Évitez les colonnes trop larges : les colonnes larges obligent l'œil à parcourir de longues distances de gauche à droite. On risque alors de "perdre la ligne" ce qui lassera rapidement vos lecteurs. Dans l'idéal, la largeur de vos colonnes doit être calculée pour contenir 10 à 15 mots maximum. Cette règle est évidemment bousculée par les design "fluides" (largeur de page variable en fonction de la taille de la fenêtre). Lire à ce sujet "Design fluide ou design fixe ?".

Nous vous conseillons de faire un choix entre un design fixe de 800 pixels de large (pour rester compatible avec la majorité des écrans) et un design "semi-fluide" qui empêche (d'une façon ou d'une autre) vos colonnes de s'étendre sur une trop grande largeur (à l'aide d'un script javascript, par exemple).

Contraste et lisibilité

Faites en sorte que vos textes soient lisibles : ça vous semble évident ? Tant mieux ! De nombreux sites proposent du texte jaune sur fond gris, du vert sur du rouge ou du rouge sur du vert, des teintes claires sur un fond clair ou foncées sur un fond foncé. Autant d'associations qui rendent la lecture très difficile. Le texte blanc sur fond noir doit être absolument évité pour les sites proposant des articles longs. Le texte le plus lisible et le moins fatigant est le texte noir sur fond blanc. Vous pouvez vous éloigner légèrement de ce grand classique à condition de toujours garder un bon contraste entre le fond et le texte.

Inutilité des gadgets

Évitez les gadgets : Compteurs, animations flash et autres gadgets n'apportent rien à votre site, mais distraient les visiteurs de votre véritable contenu. Notre page sur le contraste vous rappelle qu'un élément attirant l'attention doit toujours être au service du contenu de la page.

Sous-titre

Une taille de texte suffisante : vous avez 20 ans et une vue parfaite ? Tant mieux pour vous ! Ce n'est malheureusement pas le cas de la plupart des internautes. Les sites utilisant des caractères microscopiques sont fatigants, notamment pour les internautes ayant plus de 40 ans. Vous aussi, vous serez sans doute presbyte, à cet âge là !

Léger et rapide

Des pages "légères" : les visiteurs sont pressés et impatientés. Dans l'idéal, vos pages ne devraient pas peser plus de 30 à 50 Ko (tout compris avec les images et les animations). Ce point est absolument essentiel, mais Vincent Bernard le nuance de façon intéressante dans son article sur le sujet.

Structure du texte

Des titres de page clairs et informatifs : Où suis-je ? D'où vins-je ? Où vais-je ? Évitez à vos visiteurs de se perdre dans ces questions métaphysiques ! Offrez leur des repères et des invitations !

Sous-titre

Des sous-titres, une mise en page structurée et aérée : Découpez votre contenu en chapitres clairement identifiés. Sousepoudrez généreusement de sous-titres qui serviront de repères. Vos visiteurs doivent pouvoir choisir ce qu'ils veulent lire ou sauter.

Simplicité

Des menus simples : vos visiteurs ne passeront pas plus de quelques secondes à essayer de comprendre l'architecture de votre site. Le système de menu le plus répandu consiste à placer :

- Le menu principal sous la forme d'une barre située en haut de la page.
- Les sous menus à gauche de la page.
- Les renvois, les publicités, à droite de la page.
- Les références (bibliographie) en bas de la page.

En respectant ces habitudes, vous faciliterez la navigation sur votre site. Si vous souhaitez en changer, assurez-vous que le résultat est immédiatement compréhensible pour un internaute découvrant votre site pour la première fois.

Pas de gadgets

Des caractères standards : pour être sûr que vos visiteurs pourront voir vos pages de la même façon que vous les voyez, vous devez impérativement rester dans les grands "standards" de caractères : arial, times, verdana,... Si vous souhaitez absolument utiliser d'autres polices pour vos titres, affichez-les en tant qu'images, mais n'oubliez-pas de doter ces images d'un texte alternatif (balise "alt").

5 Veille information

5.1 «Zones de recherche»

5.1.1 Bases de Données Biologiques : articles, termes médicaux...

- Articles scientifiques PubMed => termes MeSH (thesaurus / ontologie) ++ recherche "full-text"

5.1.2 Termes à utiliser

Types de publications dans PubMed (indiquer le terme + "[Publication Type]" dans le champs de recherche (les 'plus' intéressants en gras) : voir le tableau 2, page 21.

5.2 Outils automatiques

- Base PubMed et autres bases du NCBI et assimilés...
- Recherches régulières automatisées
- Régularité recherche (termes) : fréquence!!

5.3 Idées de développement (en interne)

5.3.1 Représentation données récoltées

Représenter avec graphe (visuellement) les liens de citations (à partir d'un bibTex notamment... généré par JabRef par exemple...)

5.3.2 Outil de récolte d'information

- À la demande (interrogation des bases distantes, consultation et sauvegarde en local).
- Généralisable sur plusieurs bases de données, plusieurs sources...
- Configurable comme outil de veille techno / info / stratégique (automatique).

<ul style="list-style-type: none"> — Addresses — Autobiography — Bibliography — Biography — Case Reports — Classical Article — Clinical Conference — Clinical Trial — Clinical Trial, Phase I — Clinical Trial, Phase II — Clinical Trial, Phase III — Clinical Trial, Phase IV — Collected Works — Comment — Comparative Study — Congresses — Consensus Development Conference — Consensus Development Conference, NIH — Controlled Clinical Trial — Corrected and Republished Article — Dictionary — Directory — Duplicate Publication — Editorial — English Abstract — Evaluation Studies — Festschrift — Government Publications — Guideline — Historical Article — In Vitro — Interactive Tutorial — Interview — Introductory Journal Article — Journal Article 	<ul style="list-style-type: none"> — Lectures — Legal Cases — Legislation — Letter — Meta-Analysis — Multicenter Study — News — Newspaper Article — Overall — Patient Education Handout — Periodical Index — Portraits — Practice Guideline — Publication Components — Publication Formats — Publication Type Category — Published Erratum — Randomized Controlled Trial — Research Support, American Recovery and Reinvestment Act — Research Support, N.I.H., Extramural — Research Support, N.I.H., Intramural — Research Support, Non-U.S. Gov't — Research Support, U.S. Gov't, Non-P.H.S. — Research Support, U.S. Gov't, P.H.S. — Retracted Publication — Retraction of Publication — Review — Scientific Integrity Review — Study Characteristics — Support of Research — Technical Report — Twin Study — Validation Studies — Video-Audio Media — Webcasts
--	--

TABLE 2 – Termes [Publication Type] de PubMed

6 Idées pour un Framework interne

Idées Framework (hors fourniture spécifiques)

- Objectif(s) principal(aux) : avoir en interne un outil de développement rapide et propre pour faciliter le travail de façon général

6.1 Idées générales

6.1.1 «Problématique(s)»

- "temps de latence" de la plate-forme prévue BioP
- Temps de développement et contraintes associées
- Hétérogénéité potentielle (modèles / implémentations / ...)

6.1.2 Reprise paradigme MSR

- "data" : BDD / maquette connaissance / ontologies
- Coordination de données

6.1.3 Reprise RFLP

- Notion de workflow, flux de données, flux de traitement
- Succession éléments logiques

6.1.4 Interfaçage graphique

- Adaptation et possibilités (GWT / GXT, GEF... Java Swing ?? HTML ??)

6.1.5 Eléments client-serveur

- WWWeb ?? (GWT / Tomcat...) => bof... (possibilités et contraintes JavaScript / HTML / CSS ;)
- "Sockets maison" et autres services similaires

6.1.6 Bibliothèques

- Hétérogénéité ?? (Java / C-C++...)
- GUI (interface graphiques...)

6.1.7 Branchement des prototypes et autres tests, bibliothèques

- Travaux en cours et échanges avec partenaires (idées, preuves de concepts...)

6.2 Convergences et «Modèles-Patterns-Templates»

6.2.1 Conventions de codages et exemples de définitions de classes, attributs, méthodes, fonctions...

6.2.2 Design-Pattern et autres méthodologies...

6.2.3 Définitions de méthodologies de constructions de bibliothèques (DLL par exemple)...

6.2.4 Définitions de tests unitaires et tests plus généraux...

7 Section

7.1 Sous-section

7.1.1 sous sous sous section

[...]

7.1.2 sous sous sous section

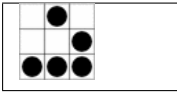
	PART-LOGO lorem ipsum dolor sit amet...
---	--

TABLE 3 – Un tableau référencé

Brouillon
(draft)

7.2 Encore une sous-section

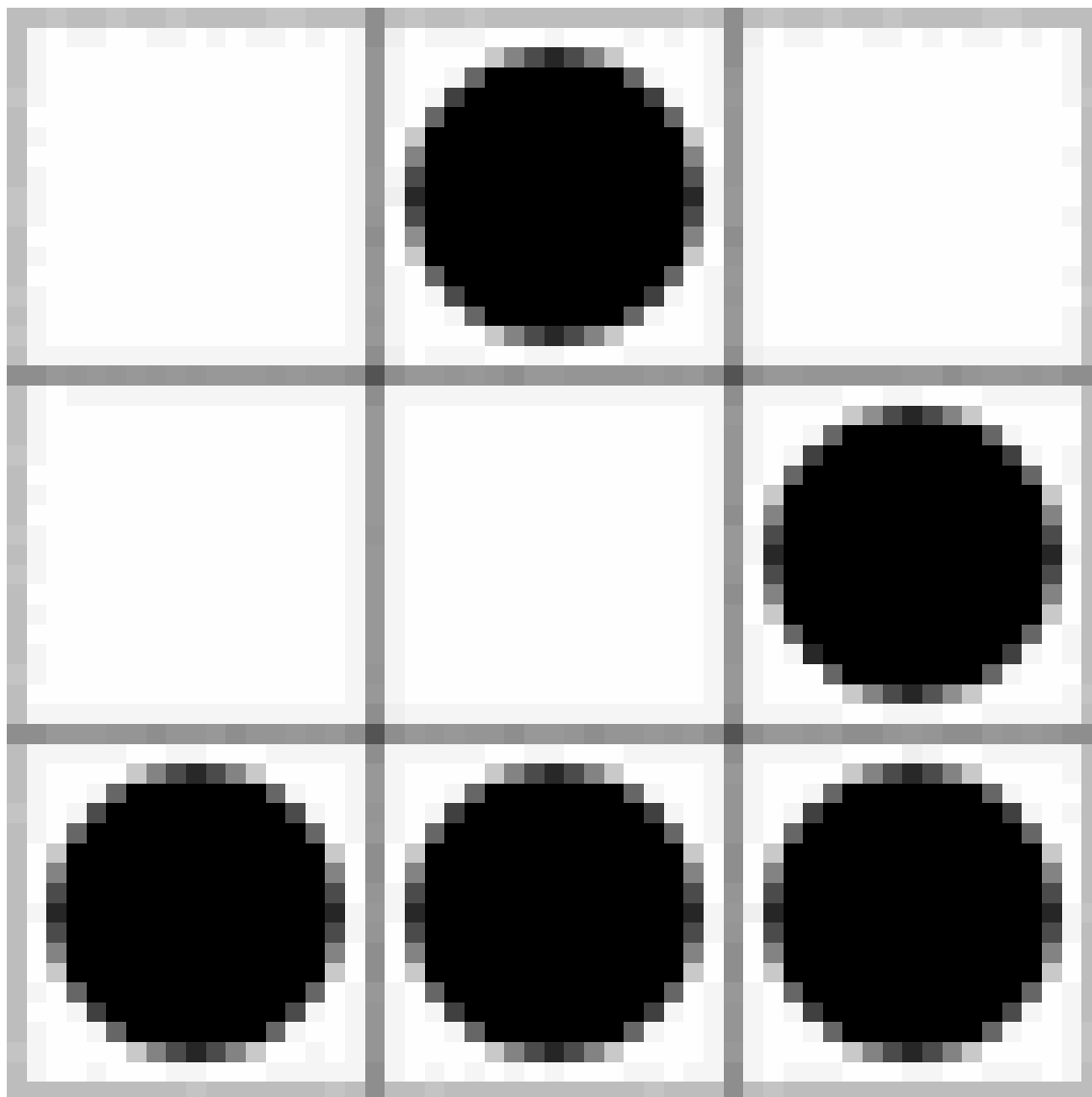


FIGURE 1 – Une belle image

Brouillon
(draft)

Abbréviations

BLAST	Basic Local Alignment Search Tool <i>algorithmes et logiciels pour l'alignement de séquences et la recherche de similarités locales</i>
BNDB	Biochemical NetWork DataBase (<i>entrepôt de données</i>)

Brouillon
(draft)

Bibliographie

Références

- [1] Jacques BERTIN : *Sémiologie graphique*. Mouton/Gauthier-Villars, Paris, 1967.
- [2] Jacques BERTIN : *La Graphique et le traitement graphique de l'information*. Flammarion, 1975.
- [3] Jacques BERTIN : *Sémiologie graphique. Les diagrammes, les réseaux, les carte*. École Des Hautes Études En Sciences Sociales, Paris, 1999.
- [4] Michelle BÉGUIN et Denise PUMAIN : *La représentation des données géographiques, Statistique et cartographie*. Armand Colin, Paris, 2003.
- [5] Gilles Palsky et Marie-Claire Robic : Aux sources de la sémiologie graphique. In Cybergeog : European Journal of GEOGRAPHY, éditeur : *Colloque "30 ans de sémiologie graphique"*, volume Aux sources de la sémiologie graphique, 2000.
- [6] WIKIPEDIAFR : Design de l'information.
- [7] WIKIPEDIAFR : Doxygen.
- [8] WIKIPEDIAFR : Jacques bertin (cartographe).
- [9] WIKIPEDIAFR : Sémiologie graphique.

Brouillon
(draft)

