Université du Québec
École de technologie supérieure

Rapport de labo 2(T2) : Réusinage (Refactoring)

Cours	LOG530
Session	ETE2015
Groupe	1
Numéro de votre équipe	4
Chargé(e) de laboratoire	Francis Cardinal
Étudiant(s)	Max Moreau Charly Simon Benoit Rudelle
Adresse(s) de courriel	max.moreau.1@ens.etsmtl.ca charly.simon.1@etsmtl.net benoit.rudelle.1@ens.etsmtl.ca
Code(s) permanent(s)	MORM30038905 SIMC28069108 RUDB12119006
Date	30/06/2015

CONSIGNES POUR LA RÉDACTION

Le format de ce document doit être respecté.

Remplissez complètement l'en-tête précédent.

Ne touchez pas à la grille de correction qui suit.

Ne modifiez ni l'ordre ni les titres des sections, n'omettez aucune section. Vous pouvez reformater les titres si vous voulez utiliser une table des matières et créer des sous-titres pour les sous-sections de votre travail (fortement suggéré).

Citez vos sources, c'est à dire mentionnez explicitement l'origine de vos idées, algorithmes, exemples, figures, etc., et ce quelle que soit la langue d'origine. Voir le lien *plagiat*, dans la page *laboratoires et travaux pratiques*).

Pour la présentation des textes, figures, tables et références, utilisez les *normes de rédaction des mémoires de maîtrise* (disponibles sur le site de l'ÉTS).

CRITÈRES DE CORRECTION

Dans ce laboratoire, tous les critères seront appliqués strictement (orthographe, gabarit, etc.)

Voici quelques points pour vous aider à rédiger de bons rapports.

1. Gabarit. Dans les sections, effacer toutes le texte venant du gabarit (remplacez les par votre texte!).
2. Rédaction et orthographe. On retire des points pour fautes de frappe, d'orthographe, de grammaire, et de rédaction : phrases trop longues, tournures maladroites ou embrouillées, texte incompréhensible. Relisez vous avant de livrer!
3. Figures et tables : pas de figures ou de tables dans la section introduction ni dans la section interprétation et discussion. Légendes obligatoires. Attention : on doit pouvoir comprendre parfaitement la figure à 'aide de la légende sans revenir au texte.
4. Les figures viennent en complément du texte mais ne le remplacent pas. *Le texte doit être clair et compréhensible sans les figures*.
5. Sur le contenu : tout ce que vous écrivez doit être 1) objectif (neutre, non biaisé par des préférences personnelles, etc.). 2) critique (pas de louange excessive, et aller au fond des choses). 3) justifié (par des résultats, des faits) 4) informatif (clair, et pas redondant, c'est à dire pas de redite, ni de phrases qui n'apportent rien).
6. Style : évitez le "je" et les tournures informelles du langage parlé. Les paroles s'envolent, les écrits restent.

GRILLE DE CORRECTION

1 Introduction /0.25	
2 Planification du travail /0.25	
3 Réalisation du réusinage (changement de langage) /4	
4 Nettoyage de code puant par réusinages /3	
5 Réusinage de base de données /2	
6 Conclusion /0.5	
Total partiel /10	
Points négatifs – rapport	
Références (-10% max)	
Orthographe et grammaire (-10% max)	
Présentation (-10% max)	
Retard (-10% par jour)	
Note du rapport / 10	

Tables des Matières

1. Introduction

Ce laboratoire a pour but de nous faire appliquer la réingénierie sur une application fonctionnelle. Nous voulons expérimenter un changement de code source d'un langage (autre que Java) à un autre (vers Java et autres).
Cela nous permettra d'expérimenter les différents outils a disposition tous en appliquant les principes de réingénierie vue en cours.

Ce laboratoire se divisera en 3 parties la première étant la conversion de l'application suivie d'une analyse des zones sombre (code puant).

Une partie2 constitue d'une analyse de code puant sur un autre programme Java fournit.

Enfin une analyse de code puant dans un modèle de base de données également fourni en annexes.

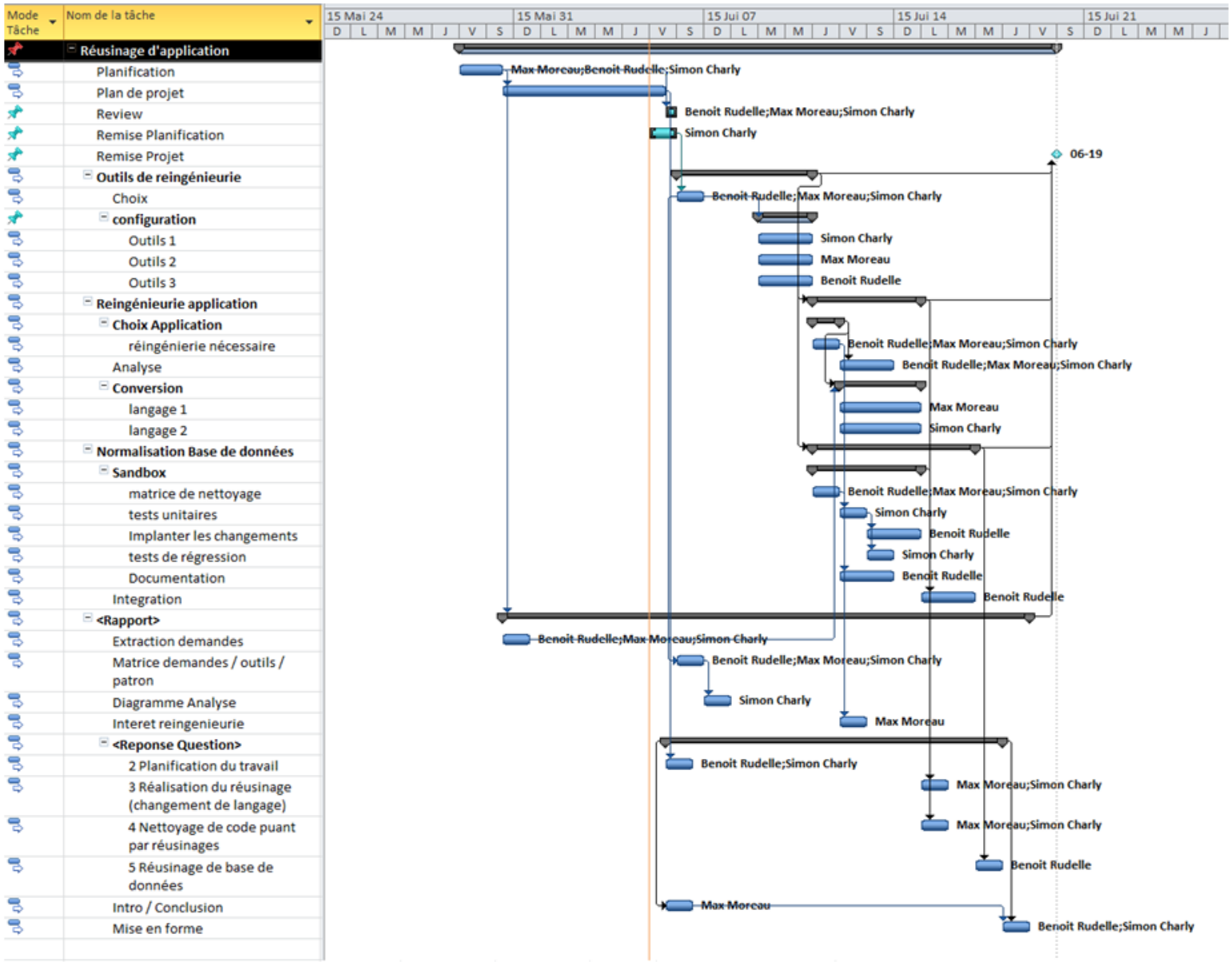
2. Planification du travail

Nous avons décidé de planifier notre travail en respectant les deux notions suivantes

- Une charge de travail équilibré pour tous les membres de l'équipe
- Garder l'ensemble de taches cohésives

Pour faciliter le développement du projet, nous avons suivi une approche itérative dans les différentes taches ce qui laisse la possibilité de faire des revues.

	Mode Tâche	Nom de la tâche	Durée	Début	Fin	Prédéces	Noms ressources	Ajouter un colo
1		**Réusinage d'application**	**22 jours**	**Ven 15-05-29**	**Ven 15-06-19**			
2		Planification	1,5 jours	Ven 15-05-29	Sam 15-05-30		Max Moreau;Benoit Rudelle	
3		Plan de projet	6 jours	Sam 15-05-30	Ven 15-06-05	2		
4		Review	0,5 jour	Ven 15-06-05	Ven 15-06-05	2	Benoit Rudelle;Max Moreau	
5		Remise Planification	1 jour	Ven 15-06-05	Ven 15-06-05		Simon Charly	
6		Remise Projet	1 jour	Ven 15-06-19	Ven 15-06-19	7;13;20;28	Max Moreau	
7		**Outils de reingénieurie**	**5 jours**	**Sam 15-06-06**	**Mer 15-06-10**			
8		Choix	1 jour	Sam 15-06-06	Sam 15-06-06	5	Benoit Rudelle;Max Moreau	
9		**configuration**	**2 jours**	**Mar 15-06-09**	**Mer 15-06-10**	**8**		
10		Outils 1	2 jours	Mar 15-06-09	Mer 15-06-10		Simon Charly	
11		Outils 2	2 jours	Mar 15-06-09	Mer 15-06-10		Max Moreau	
12		Outils 3	2 jours	Mar 15-06-09	Mer 15-06-10		Benoit Rudelle	
13		**Reingénieurie application**	**4 jours?**	**Jeu 15-06-11**	**Dim 15-06-14**	**7**		
14		**Choix Application**	**1 jour?**	**Jeu 15-06-11**	**Jeu 15-06-11**		**Max Moreau**	
15		réingénierie nécessaire	1 jour?	Jeu 15-06-11	Jeu 15-06-11		Benoit Rudelle;Max Moreau	
16		Analyse	2 jours	Ven 15-06-12	Sam 15-06-13	14	Benoit Rudelle;Max Moreau	
17		**Conversion**	**3 jours**	**Ven 15-06-12**	**Dim 15-06-14**	**14;29**		
18		langage 1	3 jours	Ven 15-06-12	Dim 15-06-14		Max Moreau	
19		langage 2	3 jours	Ven 15-06-12	Dim 15-06-14		Simon Charly	
20		**Normalisation Base de données**	**6 jours?**	**Jeu 15-06-11**	**Mar 15-06-16**	**7**		
21		**Sandbox**	**4 jours?**	**Jeu 15-06-11**	**Dim 15-06-14**			
22		matrice de nettoyage	1 jour?	Jeu 15-06-11	Jeu 15-06-11		Benoit Rudelle;Max Moreau	
23		tests unitaires	1 jour	Ven 15-06-12	Ven 15-06-12	22	Simon Charly	
24		Implanter les changements	2 jours	Sam 15-06-13	Dim 15-06-14	23	Benoit Rudelle	
25		tests de régression	1 jour	Sam 15-06-13	Sam 15-06-13	23	Simon Charly	
26		Documentation	2 jours	Ven 15-06-12	Sam 15-06-13	22	Benoit Rudelle	
27		Integration	2 jours	Lun 15-06-15	Mar 15-06-16	21	Benoit Rudelle	
28		**<Rapport>**	**19,5 jours?**	**Sam 15-05-30**	**Jeu 15-06-18**	**2**		
29		Extraction demandes	1 jour	Sam 15-05-30	Dim 15-05-31		Benoit Rudelle;Max Moreau	
30		Matrice demandes / outils / patron	1 jour?	Sam 15-06-06	Sam 15-06-06	8DD	Benoit Rudelle;Max Moreau;Simon Charly	
31		Diagramme Analyse	1 jour	Dim 15-06-07	Dim 15-06-07	30	Simon Charly	
32		Interet reingenieurie	1 jour?	Ven 15-06-12	Ven 15-06-12	15	Max Moreau	
33		**<Reponse Question>**	**12,5 jours**	**Ven 15-06-05**	**Mer 15-06-17**			
34		2 Planification du travail	1 jour	Ven 15-06-05	Sam 15-06-06	3	Benoit Rudelle;Simon Charly	
35		3 Réalisation du réusinage (changement de langage)	1 jour	Lun 15-06-15	Lun 15-06-15	13	Max Moreau;Simon Charly	
36		4 Nettoyage de code puant par réusinages	1 jour	Lun 15-06-15	Lun 15-06-15	13	Max Moreau;Simon Charly	
37		5 Réusinage de base de données	1 jour	Mer 15-06-17	Mer 15-06-17	20	Benoit Rudelle	
38		Intro / Conclusion	1 jour	Ven 15-06-05	Sam 15-06-06	33DD	Max Moreau	
39		Mise en forme	1 jour?	Jeu 15-06-18	Jeu 15-06-18	38;33	Benoit Rudelle;Simon Charly	


Mode Tâche
Nom de la tâche
15 Mai 24
15 Mai 31
15 Jui 07
15 Jui 14
15 Jui 21
D L M M J V S D L M M J V S D L M M J V S D L M M J V S D L M M J
Réusinage d'application
Planification
Plan de projet
Review
Remise Planification
Remise Projet
Outils de reingénieurie
Choix
configuration
Outils 1
Outils 2
Outils 3
Reingénieurie application
Choix Application
réingénierie nécessaire
Analyse
Conversion
langage 1
langage 2
Normalisation Base de données
Sandbox
matrice de nettoyage
tests unitaires
Implanter les changements
tests de régression
Documentation
Integration
<Rapport>
Extraction demandes
Matrice demandes / outils / patron
Diagramme Analyse
Interet reingenieurie
<Reponse Question>
2 Planification du travail
3 Réalisation du réusinage (changement de langage)
4 Nettoyage de code puant par réusinages
5 Réusinage de base de données
Intro / Conclusion
Mise en forme
Max Moreau;Benoit Rudelle;Simon Charly
Benoit Rudelle;Max Moreau;Simon Charly
Simon Charly
06-19
Benoit Rudelle;Max Moreau;Simon Charly
Simon Charly
Max Moreau
Benoit Rudelle
Benoit Rudelle;Max Moreau;Simon Charly
Benoit Rudelle;Max Moreau;Simon Charly
Max Moreau
Simon Charly
Benoit Rudelle;Max Moreau;Simon Charly
Simon Charly
Benoit Rudelle
Simon Charly
Benoit Rudelle
Benoit Rudelle
Benoit Rudelle;Max Moreau;Simon Charly
Benoit Rudelle;Max Moreau;Simon Charly
Simon Charly
Max Moreau
Benoit Rudelle;Simon Charly
Max Moreau;Simon Charly
Max Moreau;Simon Charly
Benoit Rudelle
Max Moreau
Benoit Rudelle;Simon Charly

3. Réalisation du Réusinage (changement de langage)

Dans cette section, nous allons effectuer 3 conversions de code à l'aide de 3 outils différents puis nous comparerons entre eux les différents codes obtenus.
Ceci nous permet d'expérimenter différents outils de conversion et aussi d'identifier les parties délicates à traduire entre les différents langages.
Nous expérimenterons le tout sur un code assez simple afin d'expérimenter les outils dans un cas simple ceci afin d'avoir des comparaisons honnêtes.
Enfin, nous identifierons les différents codes puants obtenus avec les différentes versions.

3.1 Choix du système à convertir

Système : Mastermind
URL : https://github.com/kirbyfan64/shedskin/tree/master/examples/mm
Description :

Le but du jeu est de découvrir un code secret composé de 4 pions ayant 6 possibilités de couleur. Afin de découvrir le code, nous disposons d'un certain nombre d'essais (par défaut 8). À chaque essai l'on a en résultat le nombre de pions correct, mais mal placé (1 pion blanc) et le nombre de pions correctement placé (1 pion noir).
Selon ses combinaisons obtenues, nous devrions trouver le code avec le minimum d'essais.

Code source d'origine : Python
Code source de destination : Java, C++

3.2 Choix des outils pour la conversion de langage

Outil 1 : ShedSkin

URL : http://shedskin.github.io/

Description : Programme standalone open source en lignes de commandes de licence mix (BSD, Expat, GPL3)

Tâches visées : Conversion de code Python en C++ optimise.

Justification/Explications : ShedSkin est un programme assez reconnu dans le monde de la conversion pour python.
Bien que ShedSkin ne couvre pas toutes fonctionnalités de Python il semble être le programme le plus avance et adapte pour le travail.

Captures d'écran :

```
[lighta@localhost mm] $ shedskin mastermind.py
*** SHED SKIN Python-to-C++ Compiler 0.9.4 ***
Copyright 2005-2011 Mark Dufour; License GNU GPL version 3 (See LICENSE)

[analyzing types..]
**************************************100%
[generating c++ code..]
*WARNING* code.py:18: Function (class Code, 'setPegs') not called!
*WARNING* colour.py:26: Function (class Colours, 'getNumberOfColours') not called!
*WARNING* peg.py:11: Function (class Peg, 'setColour') not called!
*WARNING* peg.py:20: Function (class Peg, 'display') not called!
*WARNING* row.py:12: Function (class Row, 'setGuess') not called!
*WARNING* row.py:15: Function (class Row, 'setResult') not called!
[elapsed time: 7.80 seconds]
```

Outil 2 : Tangible Software

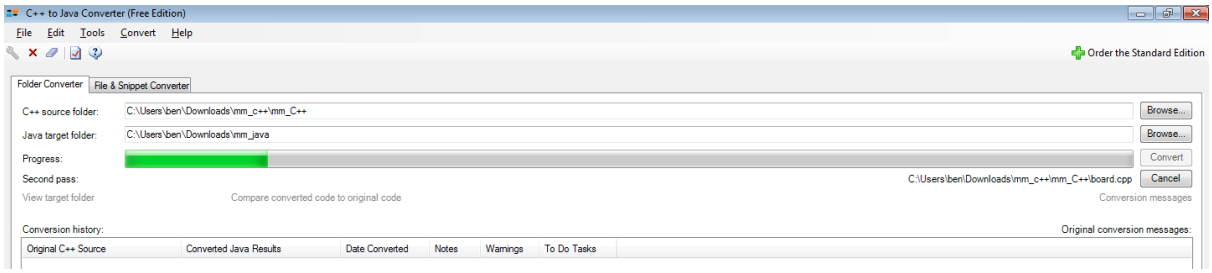
URL : http://www.tangiblesoftwaresolutions.com/

Description : Programme propriétaire standalone pour Windows

Tâches visées : Conversion de programme entre C#,C++,Java et VB

Justification/Explications :
Tangible Software semble être une compagnie canadienne (Vancouver) spécialisée dans la conversion de programme entre différents langages. (C#,C++,Java et VB).
Leur solution devrait donc effectuer un travail correct, car c'est notre premier contact avec eux si jamais l'on voulait par la suite utiliser leur service.

Captures d'écran :



Outil 3 : p2j

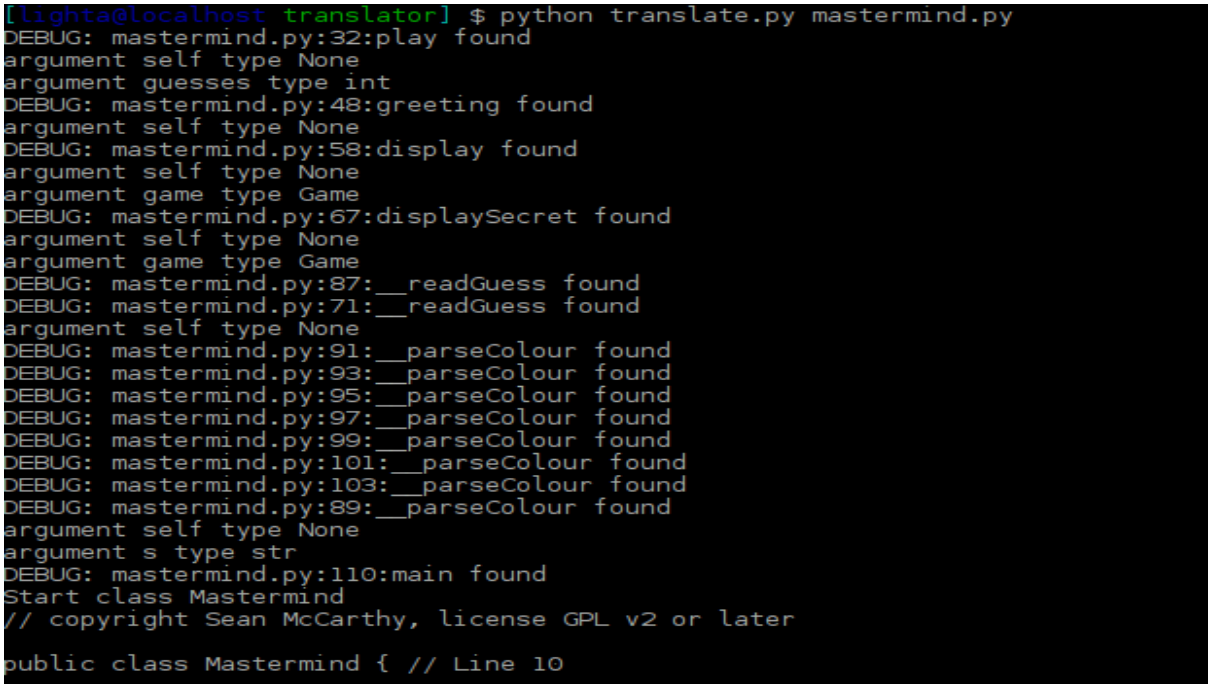
URL : https://github.com/chrishumphreys/p2j

Description : Programme standalone open source en lignes de commandes sous GPL3.

Tâches visées : Conversion de programme python en langage Java.

Justification/Explications : Ce programme semble être assez simple d'utilisation et est recommandé par différent utilisateur satisfait de la conversion obtenue.

Captures d'écran :



3.3 Analyse des outils

Outils 1: ShedSkin

Taux de conversion: 80%
Points forts:
-Vitesse et taux de conversion très grands.
-Facilité d'utilisation (1 simple ligne de commande)
-Interopérabilité (codé en Python donc utilisable sous un grand nombre d'OS)
-libre d'accès sous licence mixte. Permets de garder son programme 'fermé' tous en utilisant ShedSkin du point de vue légal.
-Génère un Makefile pour compiler le résultat obtenu.
Points faibles:
-ne supporte pas tous les aspects du langage Python. (Les fonctions introspectives ne sont pas supportées (e.g hasattr))
-convertis uniquement en C++.
-Ne trouve pas toutes les associations pour les librairies.
-Génère une abstraction de C++ en utilisant leurs headers. (Ne génère pas du C++ standard, mais des sous-objets)

Outils 2: Tangible Software

Taux de conversion: 50%
Points forts:
-effectue une conversion correcte même sans toutes les sources disponibles.
-Facilité d'utilisation. (Possède un GUI intuitif).
Points faibles:
-utilisable uniquement sous Windows et nécessite une installation. (Pas de version portable donc impossible utilisé en lab)
-Taux de conversion assez faible du manque de sources. (librairies référentes manquantes)
-faible interpolation, génération d'un grand nombre de forfaits.

Pour cette conversion nous avons obtenu trop de classe par rapport au système initial, ceci est dû au fait que le C++ dispose de header ainsi que de classe, ceci a donné lieu a des références croisées.
De plus, le nom des objets utilisés est générique donnant lieu à du code assez incompréhensible.

Outils 3: Pj2

Taux de conversion: 35%

Points forts:

-Interopérabilité (codé en Python donc utilisable sous un grand nombre d'OS)

-Portabilité (copie de fichier et 1 ou 2 lignes de commande).

-Peut convertir tous les projets python simplement en analysant les traces de ceux-ci.

Points faibles:

-semble mal maintenue. La dernière version nécessitait beaucoup de 'Hacks' pour effectuer la conversion :

--Soucis de caractères non Unicode dans des commentaires (Python ne gère que Unicode de base...)

--Soucis triviaux d'indentation. (Python est fortement dépendant de l'indentation)

L'ensemble des changements effectué est disponible sous 'doc/p2j_fixtools.diff'

-le code Java obtenu est incompilable et nécessite une intervention humaine.

-Ne convertit pas tous les fichiers / classes, mais uniquement celle disposant d'un Main.

Pour cette conversion nous avons obtenu uniquement la classe Mastermind avec des références sur les autres classes ainsi que leurs fonctions utilisées. Nous pouvons ainsi par la suite en utilisant un IDE crée les classes manquantes ainsi que leurs fonctions. Nous pouvons voir cet outil comme un créateur de squelette.

Recommandation

Si nous devions choisir qu'un seul programme, nous utiliserons ShedSkin compte tenu du résultat obtenu (fort taux de conversion). Néanmoins, cela s'applique uniquement pour Python avec un codage restreint.

Si nous voulions être plus larges, nous utiliserons Tangile qui a effectué du bon travail malgré le non-accès aux sources originales ni à toutes les librairies.

3.4 Identification du code puant

Pour cette partie, nous analyserons uniquement le système Java obtenu par les conversions de P2J.

Analyse de P2J_cleaned

Code puant 1 : Long Méthode
Description :
La méthode est longue avec du code redondant ce qui rend plus dur la compréhension.
Copie du code :
public Code compare(Code code){} : Code.java:92
Refactorings à appliquer :
Séparer la fonction en multiples fonctions
Justification/Explications :
La méthode fait 40 lignes avec 3 boucles internes pour une simple comparaison de 2 objets différents.

Code puant 2 : Duplicated Code
Description :
Code identique.
Copie du code :
public void display(Game game) : Mastermind.java:80-83;85-88;100-103
Refactorings à appliquer :
Créer une fonction regroupant la fonctionnalité.
Justification/Explications :
Les 3 morceaux de code choisi font la même fonction d'affichage, mais sur différentes listes de pions.

Code puant 3 : Problème de nom
Description :
"Type embedded in Name".
Copie du code :
public void addRow(Row row) : Board.java:36
public void setPegs(ArrayList<Peg> __pegList) : Code.java:38
Refactorings à appliquer :
Changer le nom des méthodes.
Justification/Explications :
Les méthodes citées mentionnent le nom de leurs paramètres dans le nom de leurs méthodes.

Code puant 4 : Switch statements
Description :
Action différente selon le type.
Copie du code :
public String getColourName() : Colour.java:26
Refactorings à appliquer :
Utiliser le polymorphisme avec les différents types possibles.
Justification/Explications :
La méthode getColour retourne un nom différent selon le type de couleurs, on pourrait éviter ce switch en utilisant des sous-classes.

3.5 Documentation / Re-documentation du code

Le fichier Readme a la base de l'archive explique la hiérarchie générales des fichiers afin de repère les informations. Néanmoins, les documentations des différentes versions du code sont disponibles dans leur dossier 'doc' respectif.
Ceux-ci sont disponibles sous la forme de doc autogénéré par les commentaires dans les codes.
Nous utilisons javadoc et doxygen pour effectuer ceci.

NB : Seulement les versions (cleaned) ainsi que l'original disposent d'une documentation, les autres versions étant les versions intermédiaires obtenues directement après conversions par les outils.

4. Nettoyage de code puant par réusinages

Dans cette partie nous analyserons un système existant fonctionnel, mais ayant des tests de validations incorrectes.

Nous effectuerons différents refactoring selon les codes puants identifie ainsi que réparerons et étendrons les tests unitaires afin de valider que nous n'avons rien brise.

L'ensemble des changements est disponible dans le diff : 'doc/partie2.diff' auquel nous ferons référence pour mentionner les changements.
Le code de base est disponible sous Partie2/base et le code refactorer sous Partie2/refact, certaines modifications proviennent de notre identification personnelle de code puant. (Par exemple la non-attribution de scope pour certains attributs).

4.1 Identification du code puant

Code puant 1 : Comments
Description : Expliquer le pourquoi pas simplement décrire le code
Copie du code : Trip.java:13;45;48
Refactorings à appliquer :
Retirer les commentaires et encapsuler les codes associes dans des fonctions
Justification/Explications :
Les commentaires ne sont pas absolument nécessaires, il donne un peu plus d'informations sur les opérations effectues, mais celles-ci pourraient être déplacé dans des fonctions propres pour être réutilisé.

Code puant 2 : Long Methode
Description : Dur de partager la logique, redondance de code, difficulté de compréhension.
Copie du code :
public double getExpectedTripDuration() : Trip.java:12
Refactorings à appliquer :
Séparer la méthode en plusieurs sous méthodes
Justification/Explications :
La méthode cite fait une 40ene de ligne ce qui est beaucoup pour du code Java.

Code puant 3 : Long Parameter List
Description : La liste des paramètres pour que la méthode puisse s'effectuer est longue.
Copie du code :
double calculateSpeed(int engineSpeed, double gearRatio, double differentialRatio, double tireDiameter) : Trip.java:53
Refactorings à appliquer :
Utiliser un objet pour regrouper les paramètres nécessaires a cette fonction.
Justification/Explications :
La fonction nécessite 4 paramètres ce qui n'est pas si grand, mais néanmoins conséquent pour la taille du programme. De plus tous les paramètres étant dépendant du véhicule il serait d'utiliser cet objet a la place de cette liste.

Code puant 4 : Switch Statement
Description : Actions différentes selon le type.
Copie du code :
public double getExpectedTripDuration() : Trip.java:12
Refactorings à appliquer :
Remplacement de la condition par du polymorphisme
Justification/Explications :
Pour les différents types de gear on définit un gearRatio que l'on utilisera par la suite pour calculer la vitesse. Il serait préférable d'avoir c'est différents gearRatio directement par l'objet Gear ou Transmission plutôt que d'utiliser ce switch.

Code puant 5 : Primitive Obsession
Description : Les variables primitives sont bien, mais utiliser des objets peut permettre de simplifier le code.
Copie du code :
protected int currentGear : Transmission:11
protected double tireSizeDiameter = 0.5; : Vehicule.java:6
Refactorings à appliquer :
Remplacement du type primitif par une classe.
Justification/Explications :
Le Gear et le Tire peuvent être des objets vu que le programme semble les utilise assez fréquemment, ceci permet de découpler leur représentation de leur usage et permet ainsi facilement d'étendre leurs fonctionnalités.

Code puant 6 : Data Class
Description : Classe uniquement compose d'assesseurs.
Copie du code :
Engine.java
Transmission.java
Vehicule.java
Refactorings à appliquer :
Ajouter des responsabilités au besoin.
Justification/Explications :
Chacune des classes cite ne comprend uniquement que des assesseurs. On pourrait leur assigne plus de responsabilités afin que celle-ci est plus de sens d'exister.

Code puant 7 : Feature Envy
Description : Une méthode demande trop à d'autres classes des données ou fonctions
Copie du code :
public double getExpectedTripDuration() : Trip.java:12
Refactorings à appliquer :
Deplacer la méthode dans Vehicules.java
Justification/Explications :

La méthode getExpectedTripDuration demande l'ensemble des attributs de la classe Véhicule afin d'invoquer sa méthode calculateSpeed(). Il serait plus intéressant de déplacer la méthode calculateSpeed dans véhicule.

4.2 Application des refactorings

Dépendant de quel refactoring est applique avant, certain code puant ne sont plus identifiables.
L'ensemble des changements effectue est disponible sous un diff dans 'doc/partie2.diff'

Refactoring 1 : Feature Envy
Description : Une méthode demande trop à d'autres classes des données ou fonctions
Copie du code modifié après le refactoring : (cf doc/partie2.diff:270-283)
Justification/Explications :
L'ensemble des paramètres pour effectuer cette fonction étant dépendants de la classe véhicules nous l'avons déplacé dans cette classe-ci. Permettant ainsi de régler ce code puant et donnant en même temps a la classe Vehicule plus de responsabilités. (Enlevant ainsi l'aspect Data Class)

Refactoring 2 :Primitive Obsession
Description : Les variables primitives sont bien, mais utiliser des objets peut permettre de simplifier le code
Copie du code modifié après le refactoring : (cf doc/partie2.diff:209-267;34-131) Gear.java, Tire.java
Justification/Explications :
Les objets Tire et Gear ont étés crées afin d'éliminer les types primitives pour chacun de ces attributs. Permettant ainsi d'étendre les fonctionnalités de celles-ci si nécessaire. Nous avons néanmoins gardé les anciennes méthodes en marquant celles-ci "deprecated".

Refactoring 3 :Switch Statement
Description : Actions différentes selon le type
Copie du code modifié après le refactoring : (cf doc/partie2.diff:275;59) Gear.java
Justification/Explications :
Les différents types de Gear ayant été créé de façon statique, le switch n'est plus nécessaire pour obtenir le ratio associe vu que celui-ci est contenu dans l'objet Gear. Lorsqu'on récupère le Gear on récupère ainsi son GearRatio et ne nécessite plus de condition.

Une fois ces 3 refactoring effectués, les autres codes puants identifiés ont soit déjà été réglés par les refactoring précèdent soit juge non particulièrement important pour nécessité un refactoring. (Les codes puants que nous avons laissés sans refactoring sont les commentaires et les data class).

5. Réusinage de base de données

Lors de cette partie, nous allons travailler sur la réingénierie d'une base de données pour une compagnie de distribution d'électricité. Nous ne communiquerons pas le nom pour des raisons de confidentialités. Cette compagnie a fait appel à nos services, car leurs services TI s'est aperçus que plusieurs problèmes récurrents ont due à la sémantique et plus particulièrement sur les 3 tables suivant : 'Compteurs', 'Résidences' et 'Clients'.

Pour remédier a ce problème nous allons utiliser plusieurs patrons de réingénierie qui nous on permit d'identifier plus précisément ce qui posait problème. Néanmoins, il faudra faire attention a ce que la réingénierie ait 'un impact limiter sur les différentes applications qui utilise celle-ci.

5.1 Identification de problèmes dans la base de données

Dans cette section nous commencerons par identifier les différents problèmes lier aux trois tables schématiser si dessous.

Compteurs		
PK	ID	
char(12)	NoCompteur	« 321-6823-142 » « 9824769255 »
char(30)	Sceau	Numéro du sceau / Années du scellage « 5764 / 2007 » « 9876 / 2009 » « 98333 / 2010 »
Date	DatePC	Production ou calibration
integer	TypeDate	1- production; 2-calibration
String	Puissance	« 240v » « .24kv » « 240volt » « 240V » « 240 »
...		

Résidences		
PK	Res_ID	
String	IDCompteur_FK	
String	IDClient_FK	
Date	DInstallation	Date d'installation du compteur
char(200)	PositionCompteur	« Côté gauche, derrière la cheminée. **Attention au chien** »
...		

Client		
PK	NAS	
char(15)	Nom	« Smith »
char(15)	Prenom	« John »
integer	nbResidences	Nombre de Résidences du client
...		

Problème 1 : Colonne Sceau
Description : La colonne Sceau contient deux informations distinctes le numéro du sceau et l'année du scellage
Identification dans la table : Conteurs.Sceau
Changements à apporter :
Pour corriger ce problème, il faudrait séparer cette colonne en deux colonnes du type intérêt avec pour nom “NoSceau” et “AnneeSceau”
Justification/Explications :
Chaque colonne ne doit avoir qu'un seul objectif de plus cela est en application directe avec le patron ‘Split Column’.

Problème 2 : Colonne DatePC et TypeDate
Description : L’objectif d’une colonne est déterminé par la valeur d’une autre colonne
Identification dans la table :Compteurs.DatePC et Compteurs.TypeDate
Changements à apporter :
Il y a deux actions possibles pour cette colonne.

- Séparer cette colonne en deux colonnes une pour la date de calibration “DateProduc” et une pour la date de calibration “DateCalibre”
- Ne pas changer la colonne de date et transformer la colonne TypeDate en IDType_FK qui ferais référence a une table contenant les diffère Type de date

Justification/Explications :
Pour simplifier les jointures et être sûr de garder toute les donné nous prendrons la première solution en supprime la colonne TypeDate et en transforment la colonne DatePC en “DateProduc” et “DateCalibre” pour une application des patrons Remove Unused Column et Split Column

Problème 3 : Donnée redondante
Description : La colonne nbRésidence est une donnée qui peut être trouvée en faisant une requête du type “groupe By Residences.IDCilent”
Identification dans la table : Client.nbResidences
Changements à apporter : Supprimer la colonne
Justification/Explications : Cette donnée est redondante du faite quelle peux être retrouvé à l’aide d’une requête. Mais surtout, cela génère un champ qui doit être mis à jour dès qu’un client est lié à une nouvelle résidence. Si le champ n’est pas mis à jour cela peut créer une incohérence dans les données.
Application du patron Remove Redondent Column.

Problème 4 : Cohérence des données pour la puissance
Description : Les puissances sont enregistrées de différentes manières pour la même valeur.
Identification dans la table : Compteurs.Puisance
Changements à apporter :
Pour corriger ce problème, il existe deux possibilités envisageables

- Ajouter une validation sur la colonne pour être sûr que l'application enregistre toujours al valeur au même format
- Modifier le type pour un entier et uniquement enregistrer la valeur de la puissance

Justification/Explications :
La solution1 permet de maintenir une meilleure consistance dans les données de la colonne, car elle ne modifie pas le type de la colonne.
Néanmoins, faciliter le traitement des données par les applications, nous prendrons la solution deux.

Problème 5 : Le type des ID des clefs étrangères
Description : Si l'on considère le cas le plus probable, les ID sont des nombres incrémentés automatiquement. Donc les clefs étrangères qui les utilise devraient être du mémé type que la clef primaire auxquelles elles sont liées.
Identification dans la table : Residence.IDConteur_FK et Residence.IDClient_FK
Changements à apporter : Modifier le type pour un Integer
Justification/Explications : Vu que les données présentes dans les colonnes de clef étrangères sont des valeurs qui doivent obligatoirement être présentes dans la colonne qui servent de clef primaire.
Attention cette modification ne s'applique pas si l'hypothèse de base n'est pas valide (les clefs primaires sont des Integer) cela permet le respect du patron Apply a standard type

Problème 6 : Type des NoConteurs
Description : Les numéros de compteur ne sont pas normalisés ce qui peut compliquer la lecture.
Identification dans la table : Compteurs.NoCompteurs
Changements à apporter : Pour corriger ce problème, nous avons deux possibilités qui s'offrent à nous

- La première est de transformer les champs en Integer
- La deuxième serait d'ajouter un trigger sur le format des données entrées dans la colonne pour uniformiser les données

Justification/Explications : Dans les deux solutions, l'objectif de la modification est de respecter le patron "Introduce Common Format". La méthode que nous implémenterons est la première, afin de faciliter le maintien de la base de données et la lisibilité des données. Dans le cas où les 3 paquets de chiffres seraient pertinents, il serait intéressent de mettre un trigger pour les garder distincts. On pourrait même envisager de les séparer en trois colonnes si les trois champs ont une signification distincte en plus de composer le numéro du compteur.

Problème 7 : Inconsistance des noms
Description : Les noms des colonnes ne respectent pas tous la même nomenclature.
Identification dans la table : Residences.IDCompteurs_FK Residence.IDClient_FK Resicence.DInstalation Compteurs.ID
Changements à apporter : Transformer les colonnes citées ci-dessus en:

- IDCompteurs_FK devient Compteurs_ID_FK
- IDClient_FK devient Client_ID_FK
- DInstalation devienst DateInstalation
- ID devient Compteurs_ID

Justification/Explications :
Pour respecter une cohérence dans le nom des colonnes dans la BD.
Pour garder la consistance dans le langage le préfixe/suffixe “ID” doit se situer toujours au même endroit.
Pour harmoniser avec Res_ID avec IDCompteurs_FK et IDClient_FK ainsi que la colonne DatePC avec DInstalation .

Le problème suivant est un problème potentiel déjà rencontré dans le passé et est mis ici à titre indicatif.
Problème 8 : La dimension des champs noms
Description : Dans certains pays les noms sont composés de 3 noms ce qui peut déborder des 15 caractères qui sont autorisés.
Identification dans la table : Client.nom et client.Prenom
Changements à apporter : Attribuer une taille plus grande aux deux champs cités plus haut comme 30 caractères.
Justification/Explications : Ce problème ne nous a pas été signalé et bon nombre d’application a un nombre de caractères limiter pour les champs de ce type afin d’économiser de la place.

5.2 Présentation de la nouvelle sémantique de la base de données

Pour résoudre les problèmes cités plus haut nous proposons la solution suivante. Cette solution se base sur les informations disponibles. Si nous avions connaissance des applications utilisant les données nous aurions pu optimiser et les adapter ainsi d'avantages les tables.

Compteurs	
Integer PK	Compteur_ID
Integer	NoCompteur
Integer	NoSceau
integer	AnneeSceau
Date	DateProduc
Date	DateCalibre
Integer	Puisance
...	

Residence	
Integer PK	Res_ID
Integer FK	Compteur_ID_FK
Integer FK	Client_ID_FK
Date	DateInstalation
VARCHAR2 (200)	PositionCompteur
...	

Clients	
Integer PK	NAS
Varchar2(30)	Nom
Varchar2(30)	Prenom
...	

Nous recommandons qu'avant de déployer la solution propose qu'elles soient déployées dans un nouvel environnement avant d'être mises en production pour garantir des tests d'intégrations fiables. De plus il serait intéressent de faire une sauvegarde des tables concerner pour envisager un rollback en cas de problèmes après la migration en production.

6. Conclusion

En conclusion, les outils utilisés pour la conversion ne furent pas particulièrement performants.
Ils peuvent être utiles lors de la migration de très grand programme afin d'automatiser certaine action, mais dans notre cas présent il aurait été plus rapide de la migrer manuellement.
Le temps pris pour débugger certains outils aurait été plus judicieusement pris pour effectuer la conversion. Nous pourrions toutefois écrire où amélioré un programme existant pour la prochaine fois.

Nous nous sommes encore une fois entrainés à la détection de code puant comme dans l'exercice 1 et avons remarqué que nous les identifions plus vite. Néanmoins, certains cas que nous jugeons puants ne sont toujours pas présents dans la liste vue en cours.

Enfin, nous avons expérimenté le refactoring de base de données qui est intéressant et permet de régler des problèmes potentiels que les applications pourraient rencontrer avant que ceux-ci deviennent problématiques.
Les patrons ont été appliqués rigoureusement afin d'avoir un fil conducteur dans notre réingénierie. Il faut noté que nous n'avions pas toute les donné ni les application pour voir la signification des informations, pour palier a cela nous avons dû émettres des hypothèse pour considérer l'utilisation des informations.
Néanmoins, le refactoring théorique est souvent plus aisé que le refactoring pratique qui lui engendre des casse-tête de versionnement sur les différentes applications.

Annexes

Grilles de vérification pour code smell

Bad Code Smells	Description	Refactoring propose
Intra-class		
Comments	Expliquer le pourquoi, pas simplement décrire le code	-Extract Method -Rename Method -Introduce Assertion
Long Method	Dur de partager la logique, redondance de code, compréhension	-Extract Method -Replace Temp with Query -Introduce Parameter Object -Preserve Whole Object -Replace Method with Method Object
Long Parameter List	Paramètres requis pour que la méthode puisse récupérer ce dont elle a besoin, elle n'a pas tout ce qu'elle a besoin	-Replace Parameter with -Method -Preserve Whole Object -Introduce Parameter Object
Duplicated Code	Code identique, structures ou étapes très différentes mais semblables	-Extract Method -Pull Up Field -Pull Up Method -Form Template Method -Substitute Algorithm
Problèmes de noms	-Type Embedded in Name -Uncommunicative Name -Inconsistent Names	Rename Method
Code Mort	Variable/paramètre/méthode ou portion inutilisée ou utilisée seulement dans les tests	Retrait du code mort
Speculative Generality	Ne pas trop généraliser/abstraire le code inutilement	Classes abstraites ne faisant presque rien: Collapse Hierarchy Retirer des délégations inutiles: Inline Class Méthodes avec paramètres inutilisées: Remove Parameter Méthodes avec des noms abstraits: Rename Method
Switch Statements	Action différente selon le type, souvent dupliquées à divers endroits	Replace Conditional with Polymorphism Replace Type Code with Subclasses Replace Type Code with State/Strategy Replace Parameter with Explicit Methods Introduce Null Object

Temporary Field	Une variable d'instance est parfois utile, parfois inutilisée pour un objet	Extract Class (ex.: liste de paramètres) Introduce Null Object (lorsque variables invalides)
Extra-class		
Primitive Obsession	Les variables primitives sont bien, mais créer des classes spécifiques peut simplifier le code	Replace Data Value with Object Replace Type Code with Class Replace Type Code with Subclasses Replace Type Code with State/Strategy Plusieurs variables à regrouper => Extract Class Si les primitives sont dans des listes de paramètres => Introduce Parameter Object Si présencede tableaux => Replace Array With Object
Data Class	Classes avec seulement des getters/setters	Ajouter des responsabilités au besoin => Move Method
Data Clumps	Données toujours retrouvées ensemble, mais éparpillées un peu partout	Extract Class Introduce Parameter Object Preserve Whole Object
Refused Bequest	Les sous-classes ne veulent pas ou n'ont pas besoin de tout ce qui est hérité	Créer une classe voisine (au même niveau hiérarchique) -Push Down Method -Push Down Field Si une sous-classe réutilise le comportement, mais ne veut pas supporter les interfaces de la super classe: -Replace Inheritance with Delegation
Inappropriate Intimacy	Deux classes sont trop couplées	Move Method Move Field Change Bidirectional Association to Unidirectional Association Extract Class Hide Delegate Replace Inheritance with Delegation
Lazy Class	Ces classes ne font presque rien et devraient être réusinées	Collapse Hierarchy Inline Class
Parallel Inheritance Hierarchies	Créer une sous-classe d'une hiérarchie requiert de créer une	Combiner les hiérarchies -Move Method

	sous-classe d'une autre hiérarchie	-Move Field
Feature Envy	Une méthode demande trop à d'autres classes des données ou fonctions	Move Method Extract Method
Message Chains	Client->getA()->getB()->getC() alors que les objets ne sont pas vraiment reliés	Hide Delegate Extract Method Move Method
Middle Man	Une classe délègue trop à une autre classe ; il faut peut-être réusiner celle qui fait tout	Remove Middle Man Si seulement quelques méthodes ne font rien: -Inline Method Transformer la classe intermédiaire en sous-classe: -Replace Delegation with Inheritance
Divergent Change	Une classe est souvent modifiée pour diverses raisons	Regrouper des responsabilités reliées (cohésion) ou ce qui change en même temps dans une autre classe -Extract Class
Shotgun Surgery	Un changement requiert plusieurs petites modifications dans des classes diverses	Rassembler les changements dans une classe -Move Method -Move Field Rassembler les comportements communs -Inline Class
Incomplete Library Class	Des responsabilités devraient être placées dans une bibliothèque, mais on ne veut pas la modifier	Introduce Foreign Method Introduce Local Extension
Alternative Classes with Different Interfaces	Méthodes qui font la même chose, mais avec des signatures distinctes	Rename Method Move Method (ex.: déplacer des responsabilités dans les classes les plus pertinentes)

Références

Volumes

- Serge Demeyer Stéphane Ducasse Oscar Nierstrasz. Object-Oriented Reengineering Patterns (Version of 2013-11-27). Square Bracket Associates

Site Internet

- Francis Cardinal, Réusinages de Fowler, [En ligne]
  https://cours.etsmtl.ca/log530/private/
  (Consulte le 15 juin 2015)

- Francis Cardinal, Réusinages BD de Ambler, [En ligne]
  https://cours.etsmtl.ca/log530/private/
  (Consulte le 17 juin 2015)

- Tangiblesoft, FAQ_on_CPlusPlus_To_Java, [En ligne]
  http://www.tangiblesoftwaresolutions.com/
  (Consulte le 14 juin 2015)

- Shedskin, Documentaion, [En ligne]
  http://shedskin.github.io/
  (Consulte le 17 juin 2015)

- P2J, Readme, [En ligne]
  https://github.com/chrishumphreys/p2j
  (Consulte le 12 juin 2015)