Projet de développement

Philippe Collet

Licence 3 MIAGE – S6

2012-2013

http://miageprojet2.unice.fr/index.php?title=User:PhilippeCollet/Projet_de_développement_2012-2013

Université Nice SOPHIA ANTIPOLIS

Objectifs

- Réalisation, en équipe,
  - d'un développement OO de taille conséquente
  - à partir d'un cahier des charges et d'une architecture préétablis en Java
- Donc : du développement !
  - Sans (gros) problème de conception
  - Avec des problèmes de
    - Communication (équipe de 4)
    - Techniques de programmation (API, etc.)
    - Fiabilité (tests unitaires indispensables)
  - En quasi-autonomie
  - Avec professionnalisme
    - Bonne réponse au cahier des charges
    - Efficacité, rapidité, qualité

Objectifs

- Travail demandé
  - **Beaucoup, beaucoup (beaucoup !) de travail personnel**
  - Surtout par rapport au faible volume des TD
  - Pour fournir du code et de la gestion de projet (expliciter ce que vous faites et ce que vous allez faire)

- Problématique
  - Comment vous organiser en équipe ? Développer en équipe ? Coder/tester ? Etre efficace ? Communiquer ?
  - Passer du cahier des charges à une définition et en suivi :
    - Des objectifs généraux et du livrable principal (l'application finale)
    - Des jalons pour y arriver, de comment évaluer qu'on arrive bien à ces jalons
    - Des contraintes

Calendrier

- Calendrier
  - Mardi 29 janvier : 1er et 2ème cours
  - Mardi 5 février : cours 3 et 4
  - Jeudi 7 février : publication des sujets
  - Vendredi 15 février : date limite de retour par mail des choix par équipe
  - Mardi 19 février : cours 5 et 6
  - Lundi 4 mars : publication des affectations
  - Mardi 5 mars : cours 7 et 8
  - Mardi 12 mars : premier TD de suivi
  - Mardi 30 avril : dernier TD de suivi
  - Dimanche 5 mai minuit : arrêt du développement (site de gestion du projet et des sources)
  - Mardi 7 mai : soutenance

Evaluation

- Rendu (soutenance et code) : 50 %
  - Soutenance : 20 %
  - Code livré : architecture, qualité et test : 30 %
- Gestion de Projet (méthode et suivi) : 50 %
  - Avancement en TD : 20 %
  - Utilisation des outils (wiki, ticket, svn) : 30 %

Principe de suivi : 7 semaines de TD de suivi

- Mardi après-midi :
  - 13h15 – 15h15 : 2h en présence de l'enseignant tuteur
    - Questions sur les fonctionnalités à réaliser
    - Propositions / validation sur le découpage du travail
    - Validation sur la conception de l'application
    - Surveillance de l'avancement du projet
    - Suivi et évaluation de chaque membre de l'équipe individuellement
    - Aide technique sur le langage utilisé
  - Jusqu'à 16h15 : salle réservée pour continuer à travailler en équipe
    - En profiter pour continuer d'avancer sur les points durs
    - Bien valider la répartition du travail, la charge de chacun
    - Mettre en place des tests
    - Finaliser un test d'intégration avec toute l'équipe

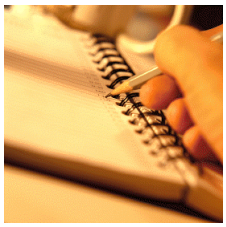
Le TD de suivi ne fait que le... suivi

- Le développement dure 9 semaines et non pas 2 ou 3 heures
  - Il faut organiser l'activité de ces 9 semaines avant et pendant
  - Chaque heure "perdue" compte
  - Il faut gérer l'information (documents, codes, tests…) en continu, particulièrement lorsqu'on travaille à 4…
- Avant le démarrage des TD de suivi, il vous est demandé :
  - De former des équipes
  - De vous auto-former (un minimum) aux outils (redmine, eclipse)
- Pendant les premiers TD, il vous sera demandé :
  - De formaliser le cahier des charges de ce que vous avez à réaliser sous forme de jalons (objectifs intermédiaires)

Soutenance et évaluation

- Soutenance (en mai) : 15 minutes
  - Présentation (technique) de la réalisation
  - Fonctionnalités réalisées (ou pas)
  - Choix de conception (et justification)
  - Mini-démo (attention, vraiment mini)
  - Travail de chacun bien identifié
  - Discussion sur les problèmes (de tout type) rencontrés et les solutions apportées
- Evaluation
  - La forme (soutenance, documents,…)
  - Le code (qualité, test, performance)
  - Mise en œuvre des outils (redmine)
  - Gestion du projet : livrables réguliers

Organisation



- Introduction
- Subversion
- Redmine
- Eclipse
- V&V
- Tests unitaires en Java - Junit
- TDD – Principes et application
- Objets Mock – Mockito
- Documentation

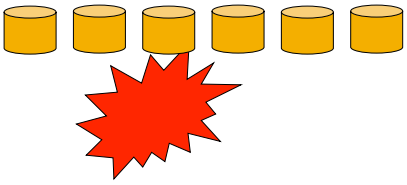
Gestion de version et de configuration

- Introduction à SVN

Université Nice SOPHIA ANTIPOLIS

Motivations

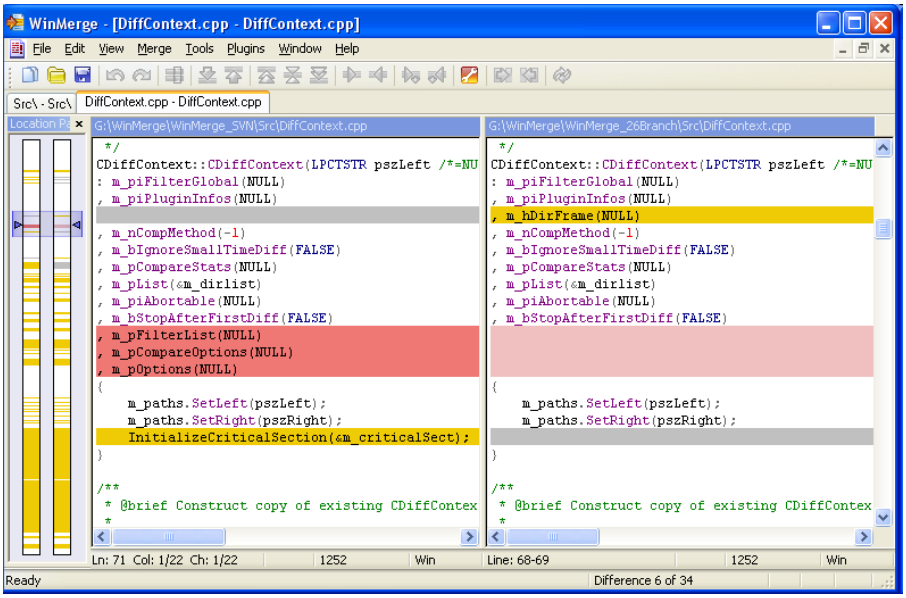
- Quand on modifie des sources :
  - Des bugs apparaissent parfois (souvent !)
- On pourrait sauver chaque version de chaque fichier modifié…
  - Ou ne stocker que les différences !
- Et quand on est plusieurs à modifier
  - Savoir qui modifie quoi
  - Ne rien écraser
  - Fusionner si on modifie à plusieurs



Principe de la différenciation

- Outil diff
- Différences entre 2 fichiers d'après
  - Ligne de début/de fin
  - Insertion/Suppression d'une ou plusieurs lignes
- Facilité de détection et de construction d'un patch
- Pas de détection des lignes modifiées
  - Traitées comme suppression + insertion

Différenciation : illustration



Historique

- SCCS (livré avec Unix dès Vx, Bell labs programmer workbench, fusionné en 1983)
  - Delta descendant
- RCS (W. Tichy 1985)
  - Delta ascendant
- CVS (B. Berliner 1989)
  - WinCVS
  - Support dans beaucoup d'environnements…

Historique (suite)

- Subversion (subversion.tigris.org)
  - La relève de cvs
  - Bonne gestion des modifications de l'arborescence des répertoires
  - Installation et maintenance simplifiée
  - Nombreuses interfaces graphiques ou intégrations dans les IDE
- Visual Source Safe
  - The Microsoft Way
- ClearCase (Rational)
  - L'usine de gestion de traçabilité

Ce que SVN n'est pas...

- Un système de construction (makefile, ant…)
- Un système de gestion de projet (Ms-project)
- Un substitut à la communication entre développeurs (ex: conflit *sémantique)*
- Un système de contrôle du changement (bug-tracking, ChangeLog)
- Un système de tests automatisés
- Un système fondé sur un processus particulier

Commande(s) SVN

- `svn subcommand [switches] [cmd_args]`
  - Commande de base coté client
  - **subcommand : obligatoire**
  - **switches : options spécifiques à la sous-commande**
  - **cmd_args : arguments de la sous-commande**

```
svn checkout http://svn.c.net/rep/svn/trunk subv
```

- `svnadmin subcommand [switches] [cmd_args]`
  - Administration de la base

Base (ou dépôt) svn

- **Locale (accéder directement par le client) :**
  - `file://`
- **Accédée à travers Apache 2 (WebDAV)**
  - `http://`
  - `https://` (SSL encryption)
- **Accédée par le protocole spécifique « svn » (possibilité de passer par ssh)**
  - `svn://` (nécessitée d'avoir un serveur svnserve)
  - `svn+ssh://` identique à `svn://`, mais *tunneling ssh* (et pas de serveur)

Administrer une base SVN

- **Créer une base SVN**
  - `svnadmin create /chemin/vers/referentiel`
  - **Par défaut format de stockage FSFS (autre format Berkeley-DB moins performant, conservé pour compatibilité)**

administrateur

create

- conf : répertoire des fichiers de config
- dav : répertoire spécifique à mod_dav_svn
- db : les données (pas directement « lisibles »)
- format : un fichier avec un seul entier donnant le numéro de version des hooks de traitement
- hooks : répertoire des scripts de hook
- locks : répertoire des verrous de subversion
- README.txt : des infos sur les autres répertoires

Administrer une base SVN

- **Au sein d'une base se trouvent un ou plusieurs projets.**
- **À chaque projet correspond en général un répertoire situé à la racine du dépôt et qui contient lui-même les fichiers et dossiers du projet.**
  - **Organiser les répertoires :**

svn copy trunk branches/my-branch

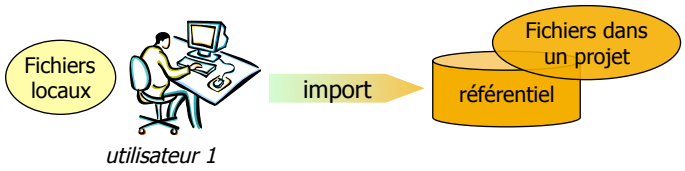
administrateur

create

```
--+--/batchxsl--+--/trunk      Tronc de l'arborescence
                |
                +--/branches   Branches de travail
                |
                +--/tags       Versions taggées
```

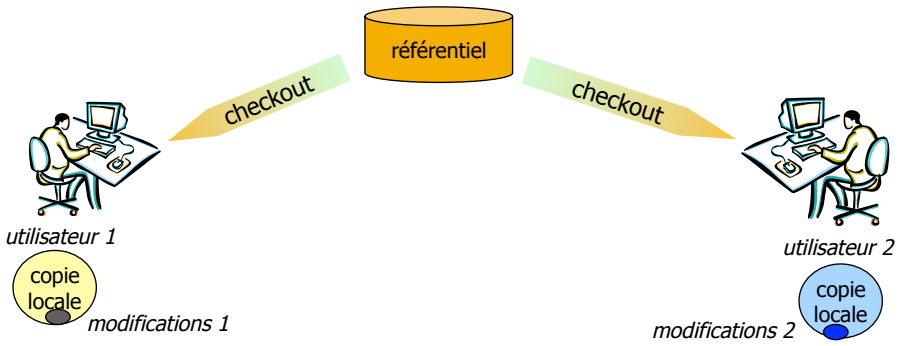
Importer des sources

- **Importer des sources**
  - `svn import rep_local /chemin/vers/referentiel [options]`
  - `svn import myTree file:///usr/local/svn/newrepos/batchxsl/trunk -m "Initial import"`



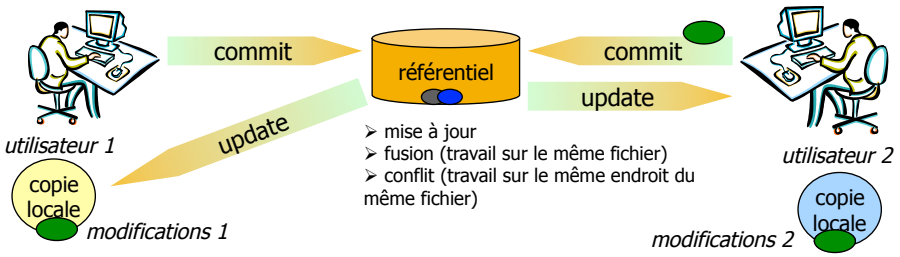
Récupérer une copie locale des sources

- `svn checkout chemin/vers/referentiel/et/projet [options]`
- `svn checkout http://svn.collab.net/repos/svn/trunk`



Propager ses changements
Mettre à jour par rapport à la base

- Propagation de vos changements
  - `svn commit`
- Récupération de nouvelles mises à jour
  - `svn update`



Signification des sorties SVN
pour *update* et *checkout*

- `U file` : **votre répertoire a été mis à jour**
- `A file` : **fichier ajouté à votre copie privée, sera propagé après commit**
- `D file` : **fichier effacé… définitivement après commit**
- `C file` : **conflit détecté lors de fusion**
- `G file` : **fusion effectuée (car pas de conflit)**

Quelques commandes et options

- **Ajouter un fichier/répertoire :** `svn add`
  - **+ commit**
- **Retirer un fichier/répertoire :** `svn delete`
  - **+ commit**
- **Copie des fichiers/répertoires :** `svn copy`
  - **+ commit**
- **Déplacer des fichiers/répertoires :** `svn move`
  - **+ commit**

Quelques commandes et options (suite)

- **Liste des répertoires dans le référentiel :**
  - `svn list`
- **Affichage des messages de commit :**
  - `svn log`
- **Mes modifications locales (pas de connection au référentiel) :**
  - `svn status`
- **Visualiser les différences :**
  - `svn diff`
- **Revenir en arrière (undo) :**
  - `svn revert`
- **Indiquer qu'un conflit est résolu sur un fichier :**
  - `svn resolved sandwich.txt`

Plein d'autres choses…

- Les outils graphiques aident à rendre tout cela *vivable*
  - *RapidSVN (multi-plateformes)*
  - *TortoiseSVN (très bonne intégration dans Windows)*
  - *Les supports des environnements de développement*
    - *Eclipse : très bon support cvs, subclipse ou subversive pour svn*
  - *…*

Illustration : TortoiseSVN

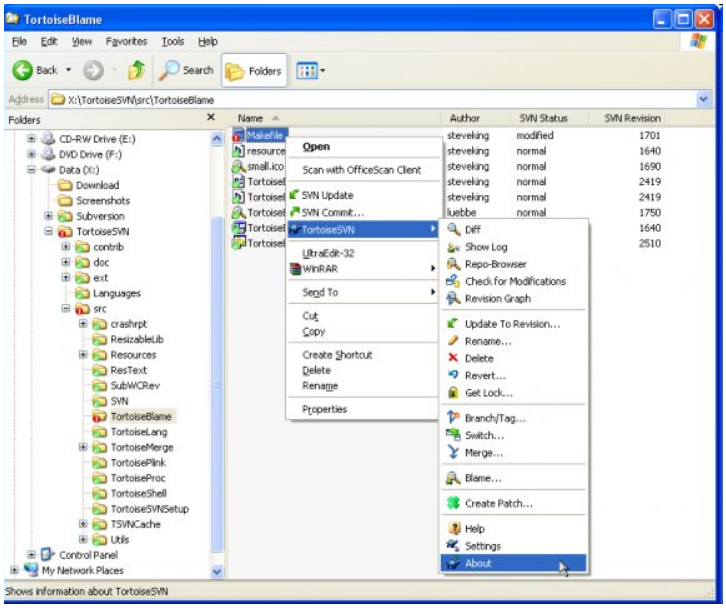
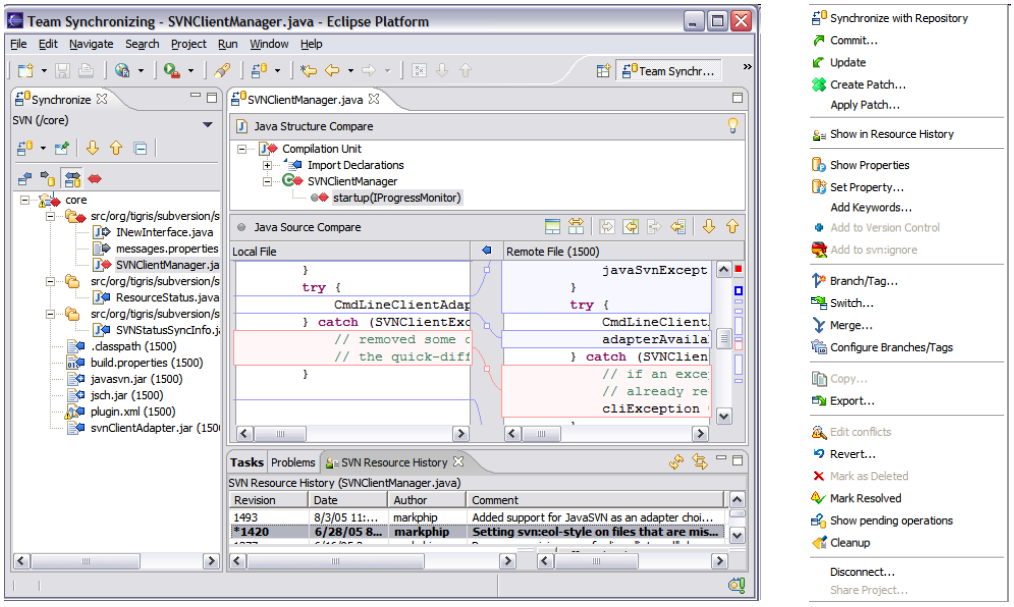


Illustration : plugin subclipse dans Eclipse



Références

- The SVN Book (en téléchargement, licence GPL)
  - http://svnbook.red-bean.com/
- Source d'information
  - http://subversion.tigris.org/

Mise en application

- Une base svn sera mise à disposition par projet
  - Connexion prévue grâce aux authentifications LDAP
  - Accessible depuis l'extérieur
- Le tuteur a accès à la base (et au site de suivi Redmine)

Et après subversion ?

- Git
  - Beaucoup plus robuste que subversion
  - « Distributed » Revision control
    - Plus de serveur central (si le serveur tombe, le service est indisponible)
    - Chacun à une base locale, il existe une ou plusieurs bases distribuées sur des serveurs (une base est la base maître)
  - On peut donc :
    - commiter en local, sans que les autres développeurs soient tenus informés
    - puis « pousser » quand le moment est opportun vers la base maître
    - Updater depuis la base maître
  - Avantages / inconvénients :
    - Si la base maître tombe, on clone sa base locale sur une autre base…
    - Complexe à utiliser : parfois on oublie de « pousser » vers la base…
- Mercurial
  - Distribué, facile d'utilisation mais moins complet et moins utilisé

Questions