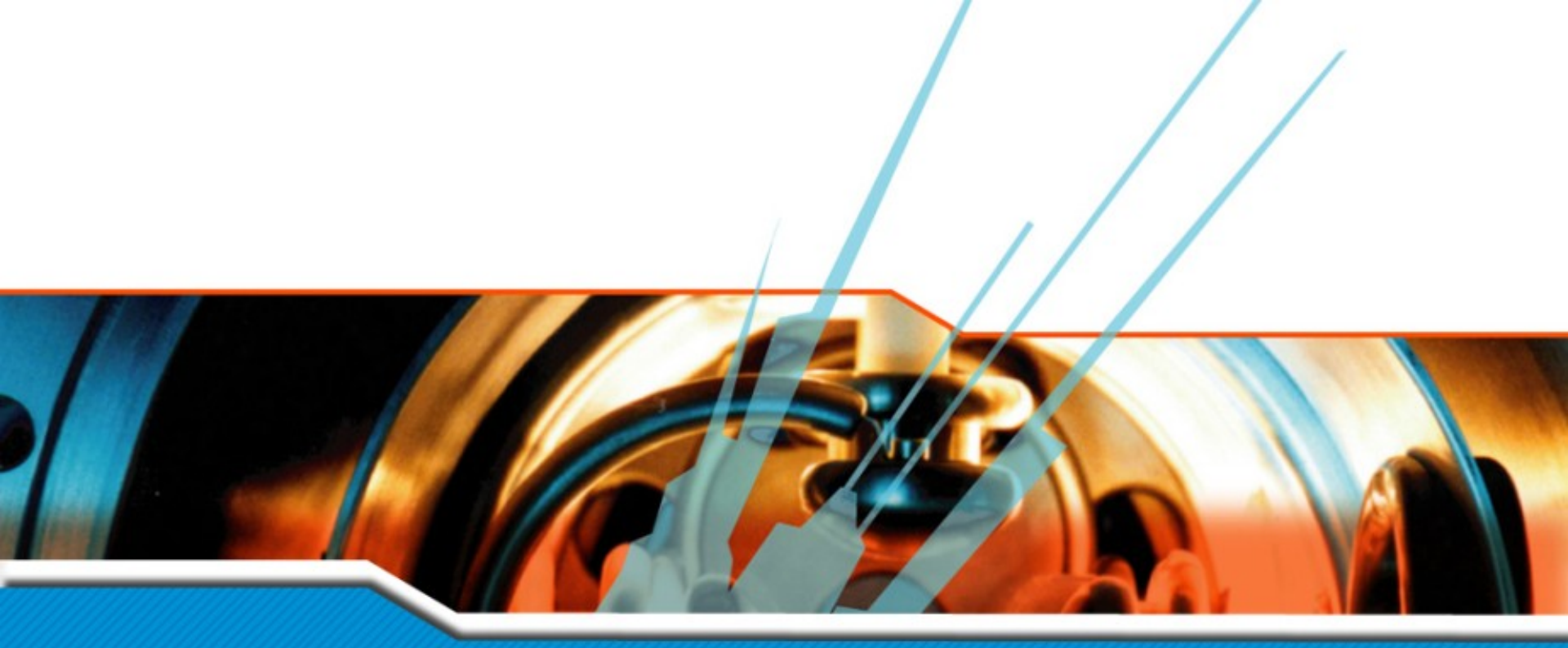


ENVOL 2016

Nouvelles technologies et méthodes pour gérer le cycle de vie d'un logiciel

28 novembre au 2 décembre à la Rochelle

Pierre-Yves Jallud et Sylvain Boschetto

Programme

- Licences, diffusion et collaboration : propriété intellectuelle
  - Quelle diffusion pour les développements ?
  - Pour la fonction publique
- Modélisation
  - Modélisation comportementale avec UML
  - Papyrus
  - UMLet
- Construction de logiciels
  - TDD (Test Driven Development)
  - Packaging avec CMake
  - Intégration continue
  - Contrôle de versions et collaboration – Git
- Ouverture nouveaux outils
  - Go
  - Docker

Licences, diffusion et collaboration

- Quelle diffusion pour les développements ?

  - ATTENTION : pas de licence = pas de droit
  - Toujours mettre une licence ainsi qu'un ©
  - choisir sa licence (équipe du projet) avant de faire les développements

  - l'auteur du logiciel est le codeur
  - l'employeur possède les droits moraux
  - ATTENTION : pour les personnes non rémunérées, le code leur appartient

  - Si utilisation de librairies, faire attention à leur licence d'utilisation

  - ATTENTION : ne jamais utiliser de librairies/images sous licence « CC sans usage commercial »
  - Scanner de de sources : antelink / antepedia / plume.org / fossology

Licences, diffusion et collaboration

- Pour la fonction publique

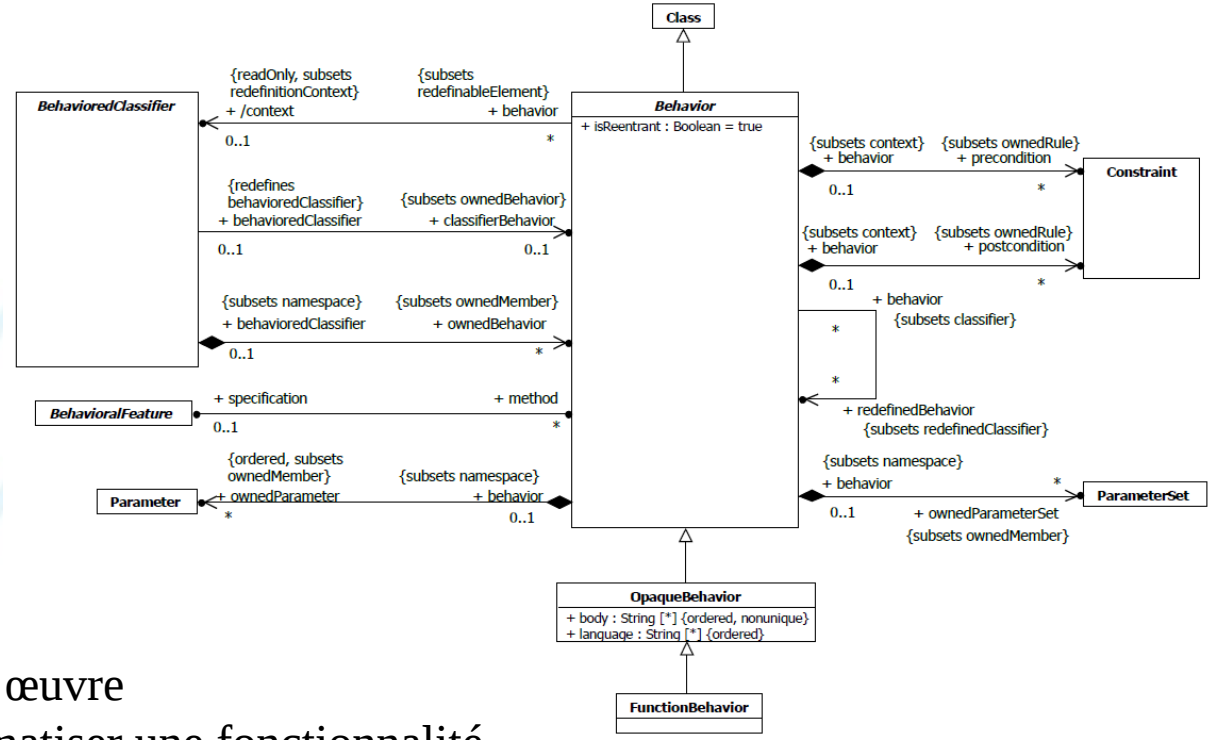
  - Loi CADA de 1978 : « ***si le document est achevé, tout doit être communicable*** » (donc accès libre et gratuit). Possibilité d'attendre la publication scientifique pour publier les données.
  - Aujourd'hui, loi Valter et Lemaire : « ***gratuité et réutilisation des données publiques*** »

  - … mais il y a des exceptions, principalement concernant la sécurité défense et la sécurité publique

Modélisation

- Modélisation comportementale avec UML

  - Éléments de base :
    - Classifier : objets
    - Event : événements
    - Behavior : actions



  - Devient très vite complexe à mettre en œuvre
  - Reste utile pour schématiser une fonctionnalité
  - À utiliser pour la communication

Modélisation

- Papyrus
  - https://www.eclipse.org/papyrus/
  - Outil intégré à Éclipse … donc assez gourmand en ressources
  - Mais langage graphique assez simple à utiliser

- UMLet
  - http://www.umlet.com/
  - Un peu plus léger, plus simple, plus adapté à des usages simples
  - Attention il n’est pas aligné sur le standard UML

Construction de logiciels

- TDD (Test Driven Development)
  - Principe : écriture des tests puis développements pour que les tests soient valident
  - Origines AGILE (2001) / Artisanat (2009)
  - 4 notions :
    - fonctionne
    - bien conçu
    - réactif au changement
    - de mieux en mieux au niveau qualité

Construction de logiciels

- Packaging avec CMake
  - Il permet de créer l'environnement nécessaire au bon fonctionnement du logiciel
  - Le packaging est utile même si le logiciel n'est utilisé qu'en équipe et qu'il n'a pas l'objectif d'être un logiciel distribué
  - Plus ce sera automatisé et plus ce sera facile de reproduire l'installation dans la vie du logiciel
  - Permet de prendre en compte les différents O.S., les mises à jours et les désinstallations

  - Inclure des fichiers README.txt, LICENCES.txt pour rappeler les possibilités d'utilisation du logiciel
  - SÉCURITÉ : sur le site de distribution, penser à mettre la signature des paquets (MD5 / SHA...)

Construction de logiciels

- Intégration continue
  - À utiliser surtout pour les gros systèmes… mais pas que
  - Permet d'avoir une chaîne de traitement automatisée garantissant le bon fonctionnement du code
  - Peut inclure :
    - Dépôt Git
    - Compilateur
    - Testeur de code (dépendances, bonne syntaxe du code, licences, etc...)
    - Testeur d'IHM (cf. SeleniumHQ : http://www.seleniumhq.org/)

Construction de logiciels

- Contrôle de versions et collaboration – Git
  - Ancêtres : CVS / SVN : Système de versionning centralisé

  - ***Git est un système décentralisé***
  - Pour commencer :
    - 1. ***Cloner*** un dépôt et travailler dessus (développements et commits locaux).
    - 2. Le ***commit*** sur le dépôt de référence se fait une fois que le développement est stable
  - Quelques astuces :
    - ***TAG*** : permet de mettre une étiquette à un commit
    - ***Branche*** : permet de développer sur plusieurs versions du logiciel en parallèle
    - ***Merge*** : permet de fusionner deux branches
    - ***Push*** : met à jour le repository de référence avec les développements locaux
    - ***Pull (fetch + merge)*** : récupère le répository de référence en local
    - ***Stash*** : permet de mettre de côté les développements et bascule le répository local sur la version précédente

Construction de logiciels

- Contrôle de versions et collaboration – Git
  - ***ATTENTION*** : ne pas mettre de fichier de configuration dans les dépôts (cf. mots de passe)
  - Outil pour nettoyer les dépôts des données sensibles : https://rtyley.github.io/bfg-repo-cleaner/
  - Préférer SourceSup (RENATER) à GitHub (plateforme commerciale)… mais à tester quand même
  - GitHub : rend le logiciel libre
  - Formation en ligne : https://ccwiki.in2p3.fr/developpements:formation:git
  - Documentation en français : http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/fr/
  - Anti-sèche Git : https://services.github.com/on-demand/downloads/fr/github-git-cheat-sheet.pdf

Ouverture nouveaux outils

- Go
  - Langage très facile d'accès, très simple… et aussi performant qu'un langage plus lourd comme C++
  - Pas de compilation, comparable à Python
- Docker
  - Gestionnaire de container
  - Optimise le déploiement d'applications
  - Englobe dans un seul paquet toutes les dépendances d'un logiciel sauf le système d'exploitation
  - Permet de séparer les développements de l'administration système
  - Fonctionne sous LINUX. En cours pour Mac et Windows
  - ***ATTENTION*** : à suivre pour la sécurité car tout est root dans un docker donc failles probables…
  - Autre outil similaire à Docker: ***Rocket*** (CoreOs)