Outils de développement

Véronique Baudin

Pascal Dayre

Frédéric Camps

3ième Conférence COMPIL – 5 Février 2009

Source principale

ANGD ENVOL Octobre 2008 Annecy co-organisé par UREC, PLUME, LAAS, ICJ, Groupe Calcul, …..

Tous les supports ici:

http://www.projet-plume.org/fr/ressource/presentations-envol-2008

PLAN

- Contexte de développement
- Pourquoi utiliser des outils
  - Contraintes et types d'applications
  - Technologies et pratiques
  - Etapes de développement et outils associés
- Environnements de développement intégrés
  - Introduction
  - IDE NetBeans
  - Cas d'usage: démonstration avec l'IDE Eclipse

Contexte

En tant que développeur, nous avons certainement tous été confrontés à ce type de situation:

- dans le cadre d'un logiciel qui doit répondre aux besoins de 2 ou 3 chercheurs, ou de 2 ou 3 équipes d'un même laboratoire ou de laboratoire différents

- dans le cadre de projets contractuels du type PCRD, ANR ou autre, dans lesquels nous avons à prendre en compte la présence d'industriels qui n'ont pas forcément les mêmes habitudes de développement

- dans le cadre de l'amélioration ou de la modification de logiciels libres développés par d'autre et sur lesquels on nous a demandé de travailler

Dans tous ces cas de figure, on nous impose d'utiliser ou de choisir un outil ou un ensemble d'outils pour réaliser notre logiciel: mais pourquoi ?

I. Contraintes et types d'applications

- Contraintes de réalisation
  - Délais
  - Qualité
  - Ressources humaines et matérielles
- Types d'applications
  - Accessibles totalement par le web
  - Application « standalone »
  - Réalisation de composants à intégrer
  - Réalisation de services
  - .....

Cycle de vie

- Qu'est-ce ?
  - Un processus
    - Phases: création, distribution, disparition
- Pourquoi ?
  - 2 buts
    - Maîtriser des risques, des délais et des coûts
    - Contrôler la qualité / aux exigences

Phases du cycle de vie

- **Définition des objectifs: que doit faire ce logiciel ?**
- **Analyse des besoins et faisabilité: recueil et formalisation des besoins du demandeur et de l'ensemble des contraintes.**
- **Conception générale: élaboration des spécifications de l'architecture globale du logiciel.**
- **Conception détaillée: définition précise de chaque sous-ensemble du logiciel.**
- **Codage : implémentation des fonctionnalités définies lors des phases de conception**
- **Tests unitaires: permettent de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.**
- **Intégration: pour s'assurer de l'interfaçage correct des différents éléments du logiciel.**

Documentation

- **Toutes les informations nécessaires pour utiliser et ajouter des développements au logiciel (manuel d'installation, jeu de tests, manuel(s) utilisateur, spécification, dossier d'architecture, dossier de conception)**

Phases du cycle de vie

- **Recette: vérification de la conformité du logiciel aux spécifications initiales.**
- **Installation et déploiement**
- **Valorisation**
- **Maintenance corrective et évolutive, support.**

Phases du cycle de vie

- **Migration vers une nouvelle application**
- **Arrêt progressif du service ou de l'application**

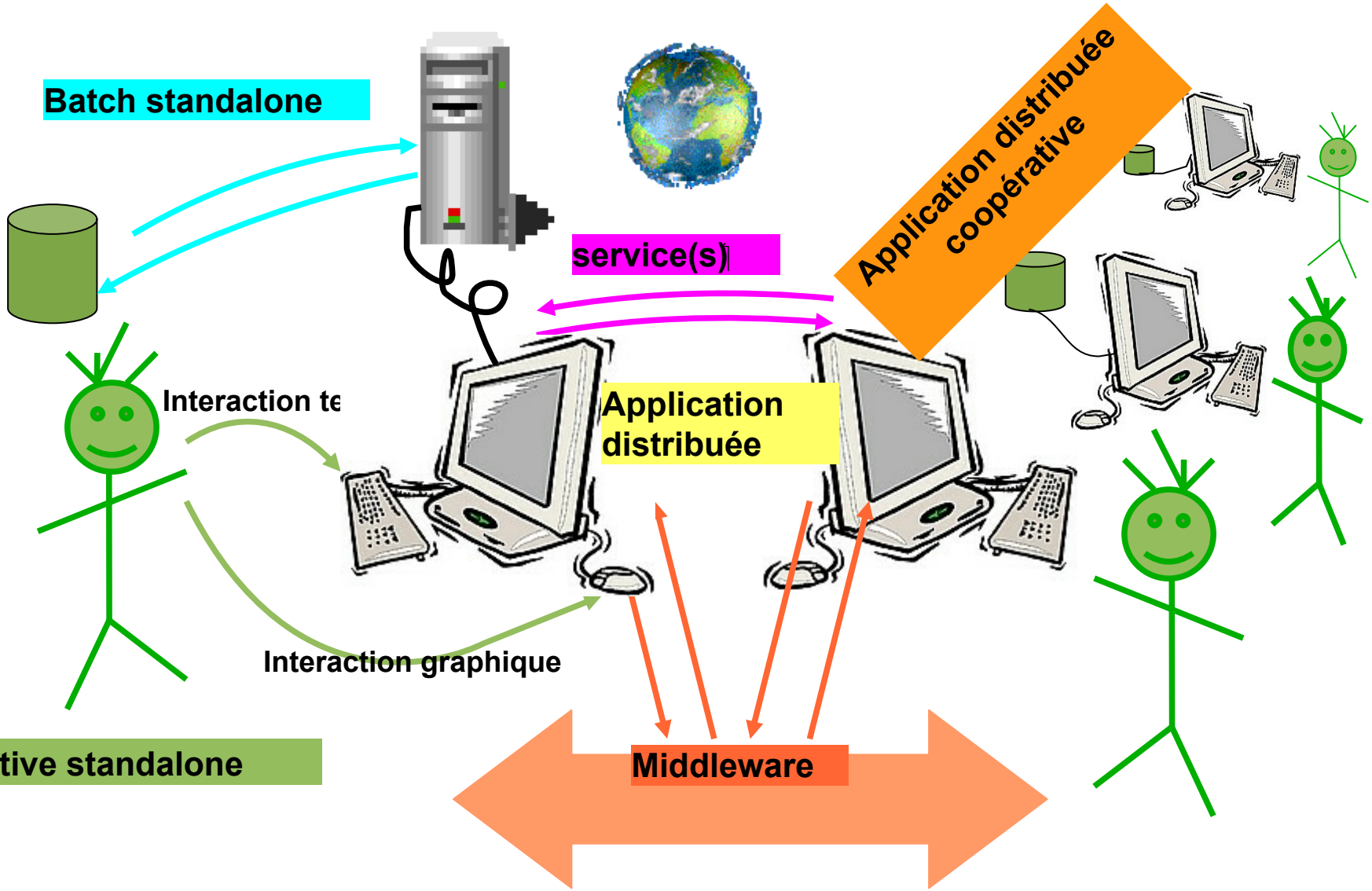
Quelques points clés pour le développement de logiciel

- Bien comprendre les demandes des utilisateurs finaux
- Tenir compte des modifications du cahier des charges
- Eviter de découvrir trop tard les défauts sérieux du projet
- Traiter au plus tôt tous les points critiques du projet
- Définir une architecture robuste et adaptée
- Bien maîtriser la complexité du système
- Bien communiquer avec l'utilisateur final
- Favoriser la réutilisation
- Faciliter le travail en équipe

Les 7 bonnes pratiques préconisées dans le Processus Unifié

- Développement itératif
- Développement à base de composants centré sur l'architecture
- Pilotage par les risques
- Gestion des exigences
- Maîtrise des modifications
- Evaluation continue de la qualité
- Modélisation visuelle

Typologie des applications



En résumé

Il existe plusieurs modèles de cycle de vie:

- **Pourquoi plusieurs modèles ?**
  - parce que aucun n'est parfait, ou meilleur que les autres
  - chacun présente des qualités et des défauts, dépendant du contexte dans lequel il est mis en oeuvre
- **Que faire ?**
  - identifier le modèle qui semble le mieux approprié à votre contexte, et suivre les grandes lignes de celui-ci, sans s'imposer des règles strictes
  - garder toujours un oeil critique sur la méthode / contexte

Quid des types de développement ?

Deux grandes familles de pratiques pour les développements:

- codage de l'ensemble des fonctionnalités définies à partir des besoins identifiés
- codage par le biais de maquettes ou prototypes successifs permettant d'inclure petit à petit les fonctionnalités identifiées initialement ou au fil des échanges entre concepteurs et client/demandeur

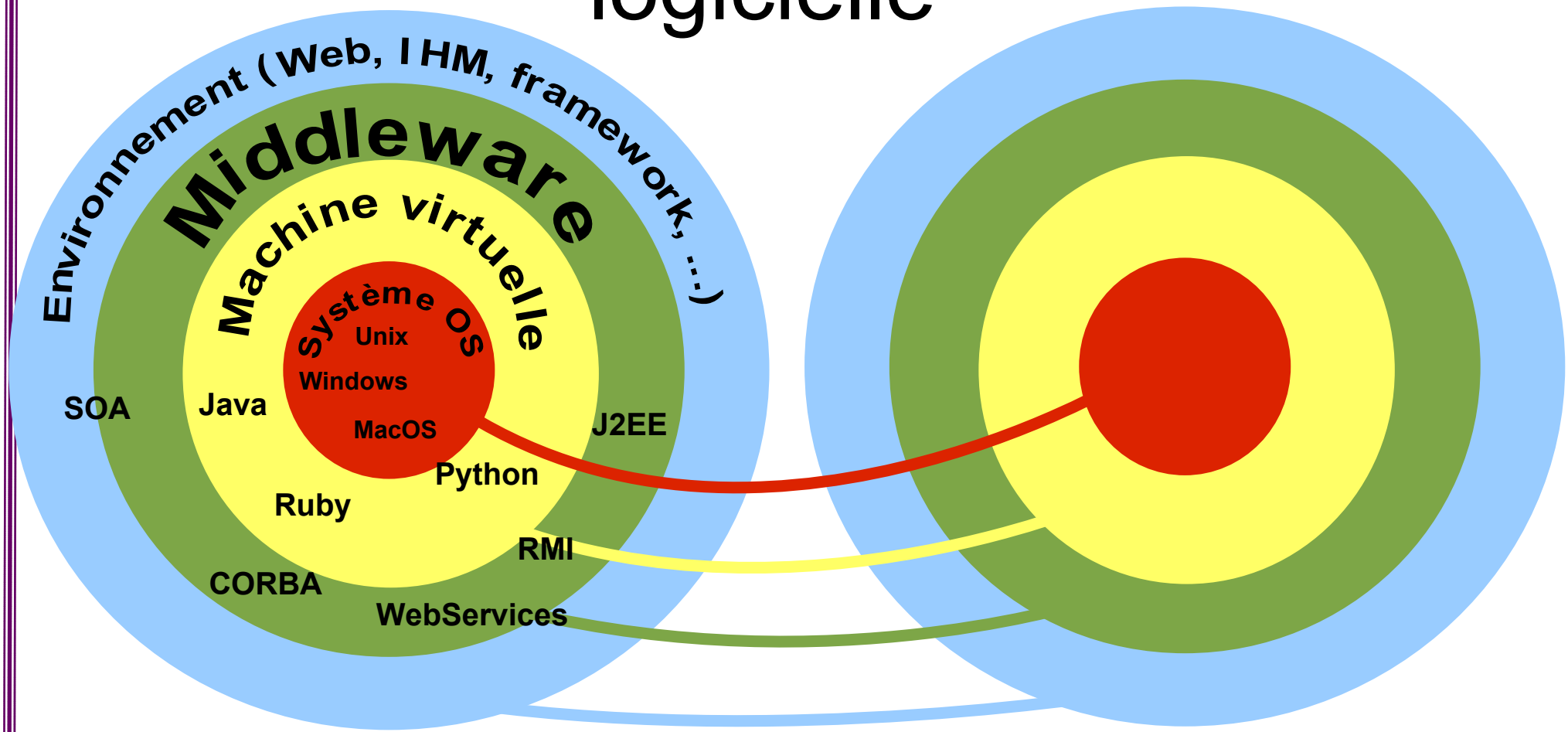
Pourquoi utiliser des méthodes de développement?

Pour guider les développeurs de la phase d'analyse à celle de maintenance

II. Technologies et pratiques

- **Organisation :**
  - Tenir compte d'un contexte existant ?
  - Développer seul ou en équipe ?
  - Application ou service ?
- **Contraintes ou choix techniques :**
  - Langage (s) ?
  - Machines cibles ?
  - Type d'architecture: à plat, client/serveur, n-tiers, ...
  - Interface graphique ?
  - Utilisation de bases de données ?
  - Utilisation de bibliothèques, API, composants ... ?

Mise en oeuvre de l'architecture logicielle



Première étape

- Comprendre ce que désire(nt) l'(es) utilisateur(s)
  - *Description de l'activité*
  - *Vocabulaire*
  - *Description du problème*
  - *Ebauche de solution*
    - UML : pour communiquer
    - Maquettage : pour valider/vérifier la demande

Le travail du développeur

- Quelque soit la méthode retenue: 5 étapes en 1 ou plusieurs cycles
  - Conception
  - Codage
  - Tests unitaires
  - Intégration
  - Recette

Etape: conception

- Outils de conception :

du papier/crayon aux outils sophistiqués pour la mise en oeuvre de différentes approches

Langages/Techniques de description	Outils de mise en oeuvre
UML	BoUML, Papyrus, eUML, ...
Réseaux de Pétri	CPN-AMI, CPNTools, TINA, HiQPN-Tool, JPetriNet, ...
Méthode formelle B	B4free , ComenC, ...
Notation Z	Community Z Tools

Etape: Codage

- Sélection des mécanismes/technologies à utiliser
  - en fonction des besoins identifiés et des contraintes d'utilisation définies
    - Quelles machines cibles ?
    - Quel type d'interface ?
    - Quel type d'utilisateur ?
    - Quel type de déploiement ?
    - Nécessité d'intégration dans un environnement existant ?
    - .....
- Choix d'un langage de programmation:
  - à partir des objectifs à atteindre pour l'application à développer, quelques question à se poser, très dépendantes de l'application

Etape : Codage vs langage

- Faisabilité :
  - Ressources pour satisfaire aux spécifications fonctionnelles de l'application
  - Ressources pour faire déployer et fonctionner les applications écrites sur la/les machine(s) cibles
  - Aspects techniques:
    - **connexion avec des bibliothèques graphiques, de calcul, de connexion réseau,.... des SBGD,**
    - **existence de codes réutilisables**
    - **utilisable pour des applications locales, client/serveur, n-tiers, web .....**
    - **présence d'un « garbage collector », ...**
- Fiabilité et performances de l'application :
  - Existence d'outils pour
    - **test,**
    - **logging, (traces d'exécution)**
    - **analyse de code**
  - Existence d'outils de
    - **profiling (mesures de performances)**
    - **pooling (gestion de groupe de serveurs, ...)**
    - **.....**

Etape : Codage vs langage

- Utilisabilité
  - Possibilité de modulariser en fichiers distincts
  - Portabilité des sources et/ou des binaires ou bytecodes sur différents OS
  - Documentations facilement accessibles (constructeur/fournisseur, cours EnsSup, ...)
  - Outils d'aide au développement
  - Environnement de développement intégré
  - Maintenabilité, extensibilité, flexibilité :
    - Langage orienté objet
    - Existence d'outils implémentant les principaux design patterns
    - Existence d' outils pour effectuer de la génération de code et du « reverse engineering » (UML <=>code source)
- Pérennité:
  - Utilisé en entreprise et en enseignement supérieur
  - Existence de communauté(s) d'utilisateurs actives
  - Normalisation ISO

Etape: codage

- Outils de codage :

des outils mono-fonction aux outils intégrés pour le développement des différente types de logiciels

Fonctionnalité	**Outils de mise en oeuvre**
langage	C/C++, Fortran, Java, HTML, XML, PHP, Python, Ruby, ...
Editeur simple ou syntaxique	vim, emacs, NotePad++,jedit,gedit, Komposer,geany, ...
Débogage	Gdb , debugger NetBeans et Eclipse, pdb (Python debugger), jdb (Java debugger), Jswat (debugger java graphique)
Automatisation des tâches de compilation, génération de code, ...	ANT, MAVEN, Make, autotools, ...
Versionning	CVS, SVN, Git, ...
Tests unitaires et mesures de performances	Junit, Cobertura, HeapAnalyser (Eclipse), VisualVM(profiler pour java), Jprobe, Gprof, python-profiler, ...
Analyse de code	Checkstyle, findbugs, ...
Création d'interfaces graphiques	Plugin VE (Eclipse)
Documentation du code	Javadoc , doxygen
Bases de données	
Déploiement d'applications	JavaWebStart, plateformes OSGi (Equinox, OSCAR, ...)

En fonction du type de logiciel à développer, des technologies utilisées, des choix d' outils vont se faire de façon naturelle

Etape: Tests unitaires

- Objectif: vérifier le « bon fonctionnement » d'une classe, d'une méthode, d'une fonction, d'une procédure, ...
  - Lors de la première mise en oeuvre
    - En utilisant les use cases définissant le fonctionnement attendu de l'application
  - Après des modifications
    - En effectuant des tests de non-régression
    - En re-utilisant les use-cases pour vérifier le fonctnnement des nouvelles fonctionnalités
- Types de tests
  - Manuels: définis à partir des uses cases, et en relation avec les utilisateurs finaux (si possible): tests significatifs pour l'application
  - Automatique: sur une spécification en logique temporelle des propriétés impérative de l'application, on utilise un Model Checker (fourniture de contre-exemple lorsqu'une propriété n'et pas vérifiée)
- Exemples d'outils
  - junit, Dart,QMTest,JFunc, GUItest

Etape: Intégration

- Objectif:
  - s'assurer que les briques logicielles développées fonctionnent correctement entre elles.
  - Vérifier que les fonctionnalités de l'applications correspondent aux spécifications
- Types de tests
  - Définis sur la base des use cases et des spécifications initiales
  - Difficilement automatisables
  - Mais il existe des outils d'évaluation de la couverture du code par les tests
    - Cobertura
    - clover

Etape: Recette

- Point de vue du développeur
  - Eléments logiciels:
    - Logiciel
    - Script d'installation ou procédure de déploiement
    - Système de suivi et de maintenance : trac, bugzilla, ...
  - Documentation
    - Manuel d'installation
    - Mode d'emploi
    - Jeux de tests (au moins de l'étape intégration)