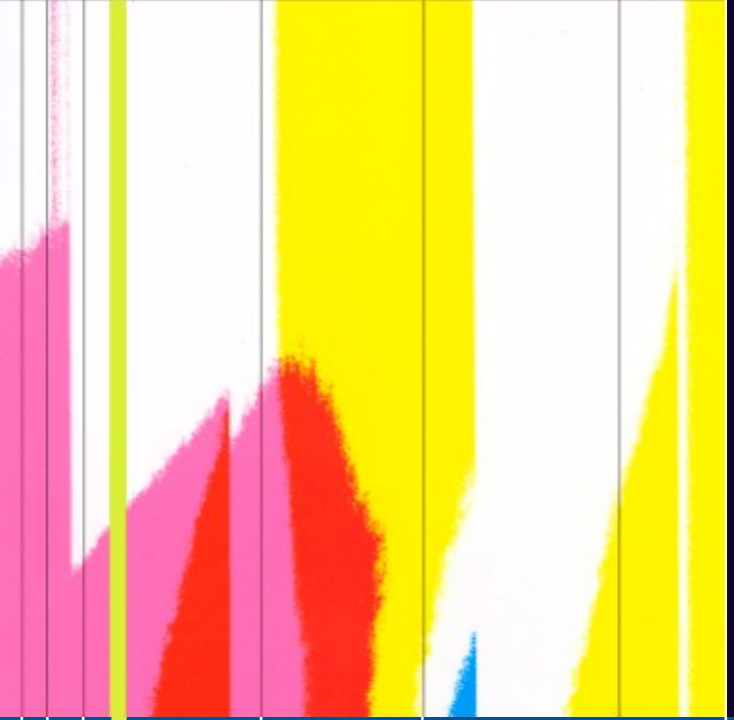


Le processus de développement logiciel : les (bonnes) questions à se poser

2 février 2010

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
BORDEAUX - SUD-OUEST

Le plan

- Un peu d'institutionnel … le SED
- Le développement logiciel
- Qu'est ce qu'un bon logiciel ?
- Bonnes pratiques organisationnelles
- Bonnes pratiques de codage
- Conclusion

Le SED : Service d'expérimentation et de développement (1/2)

- Diffuser les bonnes pratiques de développement logiciel.
- Participer au développement technologique des EPI.
- Mettre en place les plateformes expérimentales avec les EPI

Organisation INRIA du développement technologique (2/2)

Les acronymes ...

- Structures présentes dans les 8 centres :
  - SED : Service d'Expérimentation et de Développement
  - CDT : Commission de Développement Technologique
- D2T : Direction du Développement Technologique
- Réseau-Dev : coordination des SED
- ADT : Action de Développement Technologique
- PFE : Plate-Forme Expérimentale

Le développement logiciel : les phases

- Définition du projet (spécifications, conception)
- Développement (gestion de versions, suivi de bogues, tests, debug et profiling, compilation)
- Gestion de projet
- Distribution (paquetage, installation)
- Diffusion/Documentation
- Support

Le développement logiciel : type de projet

- Développer un prototype de recherche
  - Une personne (chercheur, ingénieur, doctorant, stagiaire) démarre un logiciel comme expérimentation, démonstration ou objet de recherche.
- Travailler à plusieurs sur un logiciel
  - Problèmes spécifiques de travail en groupe.
- Développer un logiciel diffusable
  - L'installation et l'utilisation du logiciel par d'autres personnes que leurs auteurs nécessitent des aménagements.
  - S'organiser pour répondre aux utilisateurs : il faut se préparer à l'existence d'une communauté d'utilisateurs, avec ses besoins et ses exigences

Le développement logiciel : diffusion

- Diffuser du logiciel hors de l'INRIA
  - L'activité de diffusion est plus efficace lorsqu'elle est préparée. Il y a également des contraintes d'ordre légal et administratif à respecter.
- Diffuser pour un public académique
  - Expérimenter avec le logiciel diffusé, souvent pour l'évaluer ou le comparer.
- Diffuser pour un public industriel
  - Environnement de travail souvent différent du nôtre et des exigences plus strictes pour les logiciels utilisés.
- Diffuser pour tout (autre) public
  - Le public non-informaticien (non-technicien) a des exigences différentes sur la présentation des logiciels.

Le développement logiciel : outils

- Développer un prototype de recherche
  - Définition des objectifs, développement, gestion de versions, compilation et build, Tests.
- Travailler à plusieurs sur un logiciel
  - Idem + Suivi de bugs + Liste de diffusion
- Développer un logiciel diffusable
  - Idem + Spécifications + Conceptions + Paquetage et installation

Le développement logiciel : métrique

Mesurer la taille d'un logiciel pour comparer, évaluer, estimer...

- Compter le nombre de lignes : « The LOC (Lines Of Code) measure is a terrible way to measure software size, except that all the other ways to measure size are worse. »
- Le nombre cyclomatique (v(G)) de Mc Cabe mesure le nombre de chemins possibles.
- ...

Le développement logiciel : estimation (1/2)

COCOMO = COnstructive COst Model

- http://fr.wikipedia.org/wiki/Constructive_Cost_Model
- Modèle basé sur des statistiques
- Apparu en 1981.
- Utiliser pour évaluer l'effort (homme mois), et la durée de développement en fonction de la complexité et de la taille et de plein d'autres paramètres !
- Trois complexités : S (faible),P(moyenne),E(forte)
- Donner la taille en amont nécessite d'avoir bien avancé la phase de conception et d'avoir une assez bonne expérience !
- Depuis : COCOMO II (1998), Fonction de point (1979, 1994)

Le développement logiciel : estimation (2/2)

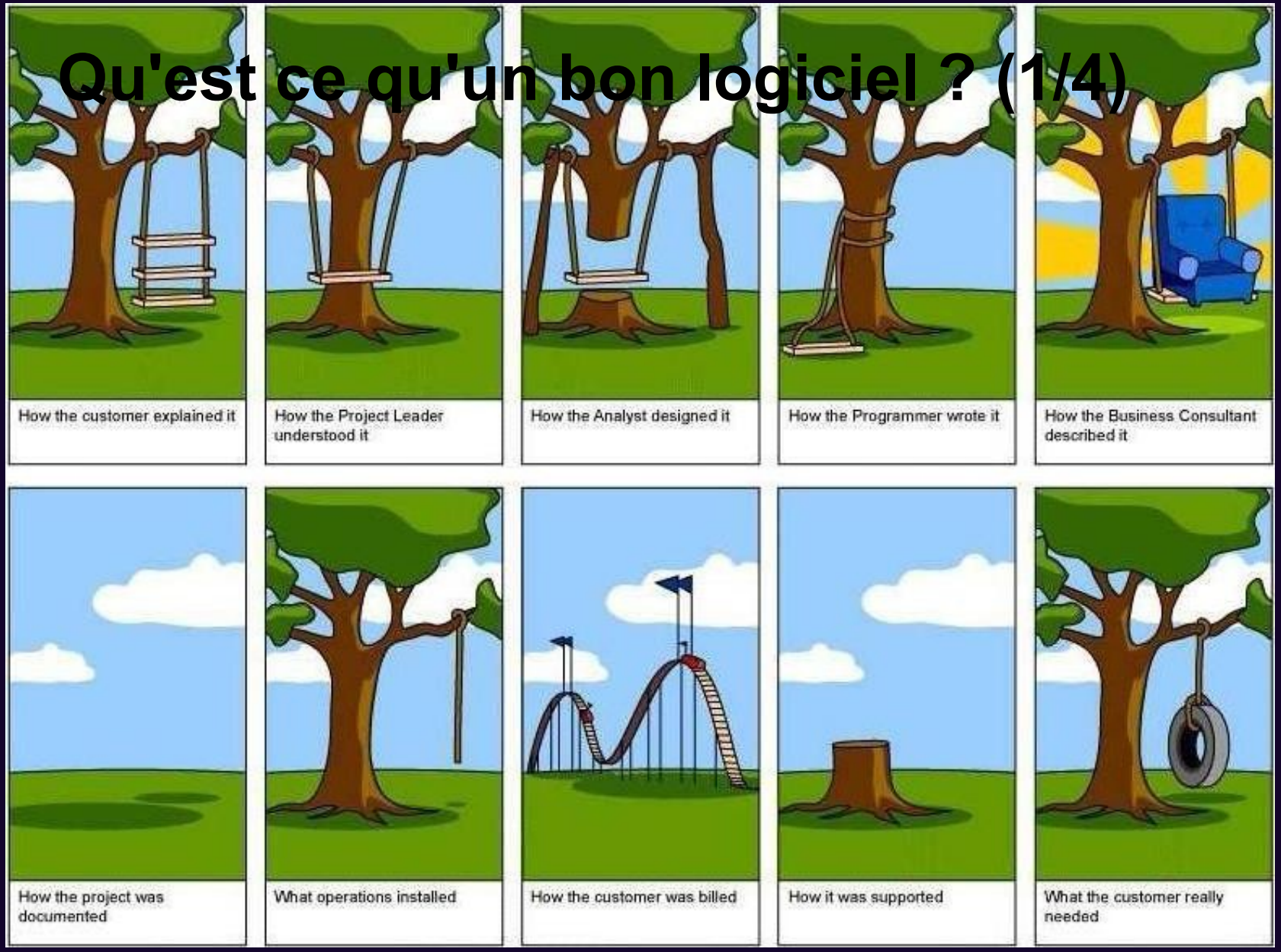
Un exemple...

Calcul d'efforts pour une projet de 8 KLOC et de complexité E :

- (8%)Expression des besoins et planification
- (18%)Conception générale
- (57%)Programmation
  - (27%)Conception détaillée
  - (30%)Programmation et tests unitaires
- (25%)Tests et intégration

=> Faire un logiciel ne se résume pas à programmer !

Qu'est ce qu'un bon logiciel ? (1/4)



Qu'est ce qu'un bon logiciel ? (2/4)

- Utile
  - Le logiciel répond au besoin (justesse)
  - Le logiciel couvre au mieux le besoin (complétude)
- Ergonomique
  - Intuitif
  - Minimise le nombre de manipulations à effectuer lors d'une opération
- Maintenable
  - Incidents facilement diagnostiqués
  - Anomalies rapidement corrigées

Qu'est ce qu'un bon logiciel ? (3/4)

- Exploitable
  - Installation, mise à jour
  - Supervision
  - Sauvegarde, récupération des données
- Fiable
  - Robuste
  - Résistant aux perturbations externes
- Performant
  - Temps de réponses adaptés
  - Consommation maîtrisée de ressources (CPU, mémoire, disque, réseau)

Qu'est ce qu'un bon logiciel ? (4/4)

- Scalable
  - Évolutif
  - Prise en compte rapide et robuste des évolutions des besoins
  - Ajouts de nouvelles fonctionnalités
- Ouvert
  - Repose sur des technologies standard(s)
  - S'intègre facilement dans un environnement

Bonnes pratiques organisationnelle (1/2)

- Avoir les compétences adaptées :
  - Spectre nécessaire couvert
    - Prévoir les formations en amont
  - Équipe stable, disponible
    - Peu ou pas de rotations
    - Pas à 50 % sur autre chose
    - Suffisamment expérimentée
- Séparer les rôles si possible...
  - Organisation (chef de projet) - ex: Chercheur, Aide SED avec suivi IJD
  - Technique (architecte, expert) - ex: Doctorants, stagiaires, IJD, Ingénieurs Experts,...
  - Fonctionnel (utilisateur, expert) - ex: Chercheurs, Doctorants, Partenaires extérieurs, ...

Bonnes pratiques organisationnelle (2/2)

- Favoriser le travail en équipe
  - Communication
  - Appropriation
  - Implication
- Formaliser la mise en commun
  - Réunion "debout" 1 fois par jour (pas plus de 15mns)
  - Réunion d'avancement 1 fois par semaine (1h maximum)
  - Réunion "d'itération" 1 fois par mois

Bonnes pratiques de codage

Mais avant de coder !

Écrire des spécifications !

Bonnes pratiques de codage

- Décrire ce que le logiciel doit permettre de faire
  - Du point de vue de son utilisation, et non comment il doit le faire
  - Ensemble de cas d'utilisation
  - Ensemble de besoins (exigences) fonctionnels
  - Ensemble de besoins non fonctionnels
    - Environnement
    - Performance
    - Sécurité
    - Support
    - Maintenabilité....
- Identifier/Numéroter/Classer chaque exigence
  - Traçabilité

Bonnes pratiques de codage

- Développer par l'exemple
  - Le chercheur/expert construit un exemple complet
    - Code, test, documentation
    - Les développeurs reproduisent l'exemple
- Apporter un support de proximité
  - Le chercheur/expert doit être disponible et répondre aux questions
- Utiliser une convention de codage
  - Préconisée par la communauté des développeurs
    - Convention Sun^{tm} pour Java par exemple
  - Préconisée au sein du projet

Bonnes pratiques de codage

- Systématiser la relecture de code
  - Par exemple, lors du premier "commit" du fichier au moins, ou en utilisant un « notificateur » de commit.
  - Par un membre de l'équipe
- Faire de la revue de code
  - Régulièrement au cours des itérations
  - Par le responsable du projet
- Ne pas hésiter à demander un audit
  - Par une personne extérieure au projet

Bonnes pratiques de codage

- YAGNI : You Ain't Gonna Need It (Vous n'en aurez peut-être pas besoin)
  - Éviter de "prévoir" des fonctionnalités à l'avance
- KISS : Keep It Simple, Stupid
  - Faire le plus simple possible
  - Optimiser une fois que cela marche
- DRY : Don't repeat Yourself
  - Automatiser au maximum les tâches de fabrication du logiciel
    - Ant, Cmake,..
  - Pas de code qui se répète
    - Éviter le "copier/coller"
- Syndrome "NOT INVENTED HERE" : Réutiliser du code !

Bonnes pratiques de codage

- Des classes de taille raisonnable
  - Entre 100 et 1000 lignes (hors commentaires), 250 en moyenne
- Un nombre limité de méthodes par classe
  - Entre 3 et 20 (hors get/set)
- Des méthodes de taille réduite
  - Entièrement visible à l'écran
  - De 0 à 10 paramètres (2 en moyenne)
- Un arbre d'héritage de faible hauteur
  - 4 niveaux max, 1 en moyenne
- Couplage faible et cohésion forte
  - couplage = nb dépendances / nb classes (idéalement entre 1 et 2)
  - cohérence = méthodes agissant sur un même ensemble de données

Bonnes pratiques de codage

Documenter au plus près du code

- Documentation pérenne
  - Génération automatique de documentation
- Commenter les entêtes (classes, méthodes)
  - Peu de commentaire au sein des méthodes : Astuce, points obscurs, complexité particulière
  - Éviter les commentaires inutiles : Get/Set
- Rédiger des tutoriels

Bonnes pratiques de codage

Se protéger par le test automatisé :

- Il documente
  - Un test fonctionnel peut-être pris comme un exemple pertinent
- Exécuter un test ne coute rien
  - L'application est testée (validée) plus souvent
- Le développeur se sent protégé
  - Il accepte plus facilement de faire du refactoring

Conclusion (1/2)

- Dans un projet, la qualité n'est pas une étape, c'est un principe
  - On doit s'en préoccuper au début pas à la fin du projet
- La non-qualité engendre la non-qualité
  - Un mauvais code n'est pas respecté
  - Les corrections sont de plus en plus difficiles
- Faire simple
  - Faire ce qui est juste nécessaire et suffisant
  - Préférer les technologies les plus adaptées
  - Bâtir autour d'une architecture aussi simple que raisonnable

Conclusion (2/2)

- La qualité, c'est du pratique
  - Bonnes pratiques faciles à mettre en œuvre
  - Retour sur investissement rapide
  - C'est une façon rationnelle de faire les tâches existantes
- L'acceptation de la qualité par les équipes passe par :
  - Un processus structurant sans être étouffant
  - Par un outillage puissant, intégré et simple à utiliser
- Faire un Bilan en fin de projet
  - Pour reproduire ce qui a marché
  - Pour corriger ce qui n'a pas marché

La suite ...

Spécification

Méthodes Agiles

Forge

Gestionnaire de version

Environnement de développement intégré (IDE)

Intégration continue

Compilation

Build

Debug

Test

Profiling

Documentation

Paquetage

Dépôt APP

Licence

Intellectual property rights

Question ?

Question ?