

Étapes d'analyse d'un logiciel

Présenté par
Jonathan Lemay
Programmeur-analyste

Mise à jour : 2010-11-24

Création : 2010-11-23

Introduction

Dans ce document, vous retrouverez qu'elles sont les procédures pour développer un logiciel informatique. Il est important de formaliser les étapes et avoir une grande organisation dans la conception.

Dans les pages suivantes vous trouverez un graphique expliquant le fonctionnement de la conception d'un logiciel informatique de l'analyse à l'installation finale en passant par les étapes de tests.

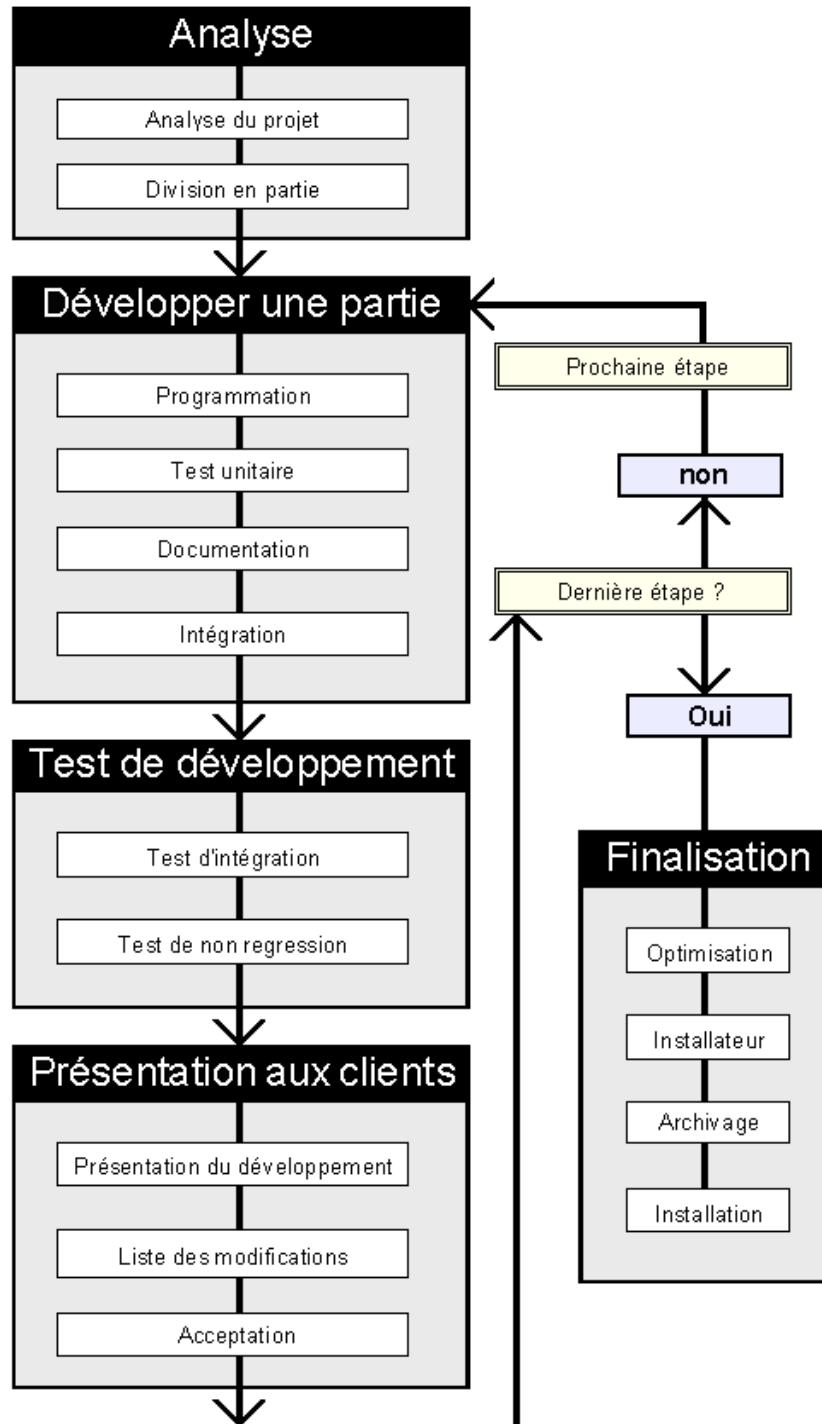
Durant les tests, il faut faire plusieurs validations en ce fiant au plan d'analyse et aux recommandations des clients, car ce sont eux qui vont accepter le produit final. S'il est mal conçu à cause de nombreux bogues ou ne se fient pas à ce que le client veut, le logiciel risque d'être laissé de côté.

Il faut préparer des jeux d'essais c'est la validation que les données entrées donnent la bonne valeur de sortie. Il faut faire un rapport de test pour s'assurer que tout fonctionne comme il se doit.

Le facteur de risque ce sont les chances qu'une personne tombe sur une erreur et l'impact sur la machine. Les problèmes majeurs qui causent un crash important seront considéré en premier et les petits corrections mineur seront développé à la fin ou durant les mises à jour de logiciel après la publication.

Une fois ces rapports fait il faut qu'un utilisateur, un programme ou un programmeur vérifie chacun de ces erreurs potentiels pour que les données soit corrigé par la suite. Ce qu'on appel une séance de test. Parfois certain bogues connus peut être corrigé dans une version beaucoup plus avancé.

1) Conception d'un logiciel



1.1) Analyse

L'analyse est l'étape préliminaire au développement d'un projet, car ceci permet de mieux sous-diviser les tâches. Au fond, cela permet de développer des conceptions plus fidèles aux besoins des clients.

Pour développer cela, il faut un modèle basé sur un plan de conception en utilisant l'approche structure. En d'autres mots, il faut créer un plan détaillé du projet en se basant sur les étapes suivantes : Les tests qui seront exécutés et les demandes des clients.

Une fois l'analyse faite, il faut sous-diviser en différentes parties. Pourquoi faire cela, car plus un programme est complexe plus les personnes ont de la chance de ce perdre durant le développement.

1.2) Développer une partie

Il faut maintenant programmer une partie en tenant compte que ceci est une section du programme principal et qu'il doit s'intégrer dans celui-ci. Il faut s'arranger aussi pour que le code source soit réutilisable dans d'autres applications. N'oubliez pas qu'il faut toujours laisser des commentaires sur le fonctionnement du code et non sur la programmation de celui-ci.

Une fois programmé, il est primordial de s'assurer que la procédure soit fonctionnelle en faisant un test unitaire indépendant du logiciel principal. Il est important de toujours se fier au plan durant l'analyse et de faire des tests structurels (boîte blanche) et fonctionnels (boîte noire) (page 7 séance de test). Une fois dans le code principal, il sera plus difficile de corriger les erreurs ce qui cause une régression.

Il est recommandé selon la grosseur du travail de laisser une explication dans une quelconque base dans le but de suivi ultérieur. Vous indiquez les informations suivantes (dans l'entête de la procédure en cours) l'utilité du module, son fonctionnement, les valeurs d'entrées et de sorties et selon votre bon vouloir qui l'a développé.

Une fois faite, Il ne reste qu'à intégrer tout cela dans le code principal et s'assurer que tout fonctionne bien avant de passer à l'étape de test.

1.3) Test de développement

Les tests sont une importante étape dans le développement d'un logiciel, car c'est l'étape qui précède la présentation au client. Si vous présenter un logiciel qui présente un programme non fonctionnel cela risque de couler un prix sur votre réputation ou celle de votre groupe.

La première étape est le test d'intégration qui est de vérifier si l'ajout est conforme aux directives de l'analyse et qu'il devra faire ce qu'il doit. Le deuxième test est la non-régression il faut s'assurer que l'ajout n'entrave pas le fonctionnement des anciennes parties déjà développé. La méthode c'est de refaire les anciens tests déjà fait auparavant. Pour de plus amples informations voir la page 7 intitulé séance de test.

1.4) Présentation aux clients

Dans cette étape il faut présenter le programme et ces fonctionnalités à des personnes qui sont intéressées à l'application. Tout ce qu'il faut c'est un intervenant externe qui pourra donner leurs points de vue. Quand un programmeur s'occupe du projet il conserve sa vision des choses. C'est toujours bon de recevoir un aperçu du monde extérieur, ce n'est pas à la fin complète qu'il faut présenter le produit. Il faut que les gens que vous connaissez aillent une idée de ce que vous faites.

Une fois présenté, il faut s'assurer que l'attente des personnes est vraiment ce que vous avez fait et écouter bien les commentaires ils sont constructif prenez tout cela en note. Indiquer les points forts et points faibles de votre projet les différentes modifications à faire et une fois faite, résumer tout cela en vulgarisant les termes pour les non-connaissant.

Une fois que les personnes ressources sont satisfaites et qu'ils y a acceptation retourner sur votre projet et développé une nouvelle partie à moins que vous avez terminé le produit et qu'il est temps de finaliser

1.5) Finalisation

Si le programme est terminé il serait temps de le finaliser. Ce dernier terme ne veut pas dire que c'est totalement terminé il sera toujours des petites mises à jour avec le temps. Cela veut dire que c'est une version stable.

En premier, il faut optimiser votre programme. Faites le ménage en enlevant le superflue comme les codes inutilisés, les fichiers temporaires, etc. Faites des tests pour diminuer la vitesse d'exécution. Vérifier la consommation d'énergie et mesurer la vitesse du code. Trouver les boucles critique qui ralentit le programme en général. Savez-vous que 10% de votre travail ralenti 80% de votre projet.

Une fois fait développé un installateur pour permettre de mettre votre projet facilement sur ordinateur. Une fois votre l'étape présente terminé copier tout dans un stockage permanent (CD). Tout ce qui reste à faire c'est de remettre votre produit final à vos clients.

2) Structure d'un plan de test

Lors de l'analyse il faut développer un plan de test qui permet au programmeur de savoir ou s'en aller. (À développer)

Il faut créer une introduction générale. Dans ce document il faut faire une présentation de l'application, le contexte de la réalisation, le choix technologique et les coordonnées du responsable. Il faut préciser quel projet il faudra tester et le préciser en mettant la liste des modules et/ou son lots et la date de livraison.

Plusieurs documents sont joints au projet qui est de la documentation général et technique expliquant le fonctionnement du projet. En indiquant la plateforme de test (souvent des ordinateurs) on y retrouve sa configuration matérielle, les outils de test, les bases de test, les données de test, les coordonnées de l'administrateur et du responsable.

Dans les tests à réalises on retrouve la liste des modules à tester, les objectifs des tests, les exigences, l'analyse du risque, la matrice et exigences vs risques, les jeux d'essais (voir page suivante) et l'estimation de la charge. Les stratégies de test qui sont les démarches donnent la description de l'approche générale, la phase de tests, les campagnes de test et l'ordre d'exécution des tests. Les conditions d'arrêt qui sont les critères retenues et pourquoi.

La gestion des fichiers d'anomalies : Actions et états, gestion des flux et liste des intervenants. N'oubliez pas le nom du responsable en ressource et diverses informations utiles. Il ne manque que le planning des tests et l'analyse du risque.

3) Jeux d'essai

Les jeux d'essai est le résultat attendu attendus, l'objectif est la validation d'une fonctionnalité, d'y décrire les situations possible, plausible ou probable.

Tout d'abord, dans le plan du jeu d'essai il faut rentrer le nom de l'application, la référence de scénario de test. Après cette saisie, il faut analyser les résultats grâce aux cycles de test. Ce cycle correspond à la période durant laquelle un ensemble de test est exécuté. À la fin du cycle un examen permet de définir si l'on peut passe au cycle suivant. Il est recommandé de s'organiser un cycle en créant un tableau pour simplifier l'analyse. En cas que l'application est une version n+1, d'une application existante, il faut vérifier la non régression et rejouer les scénarios de test de la version précédente.

La relation de couverture de test est le lien entre les scénarios de test, les objectifs et les règles de gestions. Il est difficile d'affirmer que toute l'application a été testée.

Il faut développer un rapport d'avancement qui évalue l'avancement des tests. Il faut générer des rapports qui visualisent: les scénarios enregistrés, ceux qui contiennent des bogues et peut liste les scénarios sans erreur. Il faut que ceci face référence aux objectifs de test.

Après cela, il faut créer un rapport (un graphique) des performances qui inclus : le temps de réponse observés, les tests de montés en charge, des tests en mode dégradé. Une fois faite un suivi des erreurs dans un rapport d'anomalie. Cela permet de déterminer les modules les plus fiables.

Dans le rapport d'interprétation des résultats, il faut se prononcer sur la signification du rapport, si ce sont des tests préliminaires on peut corriger les anomalies. Si on se retrouve dans une phase avancé, il faut prendre en compte l'impact de la correction. Veuillez à qualifier les anomalies en recherchant les modules qui en cause le plus.

4) Séance de test

Le but de cette séance est d'arriver à un produit « zero défaut », car il faut rechercher les anomalies dans le comportement de logiciel, les tests sont importants pour avoir un produit final de qualité.

Le test unitaire ou de module permet d'isolé le fichier ou le programme à tester. Ce processus permet de valider la qualité du code et les performances des modules. La méthode structurelle nommé boîtes blanches valide la structuration interne, comme les boucles, les instructions et les données internes. Comparé au test fonctionnel, la boîte noire c'est plus l'aspect fonctionnel.

Le test d'intégration se base sur l'architecture d'un module vérifié qui révèle les problèmes d'interface entre les différents programmes. En fait, il valide l'intégration des modules. Le test fonctionnel vérifie s'il n'y a pas d'anomalie dans les fonctions en validant les spécifications technique et les exigences fonctionnelles. Le test de non régression, vérifie si les modifications n'ont pas altéré l'application Le test d'installation test les procédures d'installation et vérifie la documentation et la vérification de la plateforme convient aux tests.

Le teste de validation vérifie que le système correspond aux besoins des clients et validé les notions mathématique. Le test IHM, vérifie que la charte graphique a été respectés en vérifiant que la présentation visuelle, les menus, les paramètres d'affichages, les propriétés des fenêtres, les barres d'icônes, la résolution des écrans, les effets de bord, la navigation (menu, raccourci, déplacement).

Le test de configuration doit s'adapter au renouvellement des ordinateurs (16 bits à 32 bits, DLL, formats de fichier, drives de périphériques). Il faut fixer certain paramètres. Le test de performance, valide la capacité pendant les charges d'accès importantes. Il faut que le temps de réponse reste raisonnable. Il faut trouver le temps normal et vérifier en cas de cas extrêmes et définir l'environnement matériel minimum.

5) Facteur de risque

Dans les facteurs de risque on retrouve l'ensemble des objectifs et le degré d'utilisateur. Il faut créer une table de risque, assigner des priorités en fonction de cette table.

En gros ce sont les chances que le problème soit rencontré par un utilisateur

- Inévitable : Tout les utilisateurs
- Fréquent : Utilisé par les utilisateurs, mais dans toutes les sessions
- Occasionnel : Un utilisateur moyen ne va pas utilisé (Pour les expérimentés)
- Rare : Apparaît que dans certains cas

Il faut aussi calculer l'impact de celui-ci.

- Grave : L'application peut continuer, mais au risque
 - Perdre des données
 - Utiliser des données corrompues
- Modéré : Pertube l'utilisateur, mais ce dernier peut le contourne
- Ennuyeux : On peut continuer à utiliser, mais elle peut causer des problèmes

Il y a en dernier la priorité du risque, qui est l'importance de l'objectif.

- Élevé : Doit être impérativement être résolu
- Moyenne : À résoudre ultérieur, mais qu'il ne faut pas négliger
- Basse : La solution peut attendre et qui seront traités en dernier