Manuel de formation
Cyril Beaussier

JE DÉBUTE SOUS

Windev

Version 1.0 - septembre 2002

COPYRIGHT ET DROIT DE REPRODUCTION

Ce manuel vous est gentiment offert pour une utilisation dans un cadre privé. Cependant si vous l'utilisez au sein d'une entreprise ou dans un but lucratif, je vous saurai gré de me faire parvenir un chèque de 8,00 € libellé à l'ordre de :

Cyril Beaussier
4, rue de Paris
77200 TORCY – FRANCE

Une facture vous sera envoyée en retour sur simple demande écrite.

Si vous souhaitez des améliorations, je suis évidemment ouvert à toute suggestion. Il en est de même si vous constatez une erreur (nul n'est parfait ☺). Pour cela, il suffit de m'écrire un courriel avec pour sujet « Je débute sous Windev » dans la rubrique « Contact » de mon site :

www.beaussier.com

Avertissement complémentaire :

Les éléments (données ou formulaires) éventuellement inclus dans ce support vous sont fournis à titre d'exemple uniquement. Leur utilisation peut avoir, dans certains cas, des conséquences matériels et juridiques importantes qui peuvent varier selon le sujet dont ils traitent. Il est recommandé d'être assisté par une personne compétente en informatique ou de consulter un conseiller juridique ou financier avant de les utiliser ou de les adapter à votre activité.

SOMMAIRE

1. Objectif

Ce manuel est né d'une demande régulière sur les *newsgroups* et autres forums sur *comment bien débuter avec Windev*. Les réponses sont alors toujours les mêmes : le guide d'auto-formation fourni avec le produit.

Or si j'ai moi-même débuter avec la version 1.5 du produit et ce fameux guide, je me suis aperçu dernièrement qu'avec la version 7, le guide ne répondait plus du tout au besoin d'un débutant. D'une part parce que PC SOFT a fait évoluer son guide en le tournant beaucoup plus vers des utilisateurs aguerris, qui maîtrisent déjà un outil similaire ou une version antérieure. D'autre part parce que Windev est devenu terriblement complexe.

Avec ce manuel, on part à nouveau sur un esprit de découverte du produit à travers une réelle progression dans des exemples amusants et ludiques.

Ce manuel est cependant une version allégée d'un futur support beaucoup plus complet en préparation (date non planifiée). Si vous êtes intéressé par un achat anticipé, écrivez moi. Un prix spécial de lancement vous sera accordé.

2. Convention

Ce manuel respecte les règles générales de présentation en matière informatique et telles qu'elles sont exposées au sein de l'AGL Windev. Le code sera identifié sous la police `courrier` comme dans l'Editeur de code.

Dans les listings, les commentaires sont introduits, soit en début de ligne lorsqu'ils sont trop longs, soit à la suite d'une instruction lorsque c'est possible. Un commentaire se caractérise par une double barre.

Exemple :

```
// Voici mes premières lignes de code avec
// l'éditeur de Windev
Info("Test en cours") // Affiche un message
```

Par convention également, les commentaires commencent toujours par une majuscule. S'ils commencent par une minuscule, ils constituent alors la suite d'une première ligne de commentaires.

3. Introduction

Windev est une application Windows qui permet de fabriquer d'autres applications Windows. Cette fabrication se fait à partir d'un éditeur qui dessine l'interface et d'un langage simple et intuitif (le *W-Langage*).

Sans exagérer, l'interface de programmation est extrêmement simple avec cet outil : on crée un bouton et on y attache le code susceptible de répondre à un évènement comme le clic de la souris sur un bouton. Windev masque en effet la réelle complexité à concevoir une application graphique Windows comme avec des langages comme le `C++`.

3.1. L'orientation

Windev est particulièrement interactif, ce qui se traduit par le fait que vous pouvez passer directement de votre imagination à sa mise en œuvre sur l'ordinateur. Vous pouvez remanier votre application à volonté et la corriger ou la compléter sans difficulté.

Cette facilité pose d'ailleurs le problème de ne pas se lancer trop vite dans le développement pur et de réfléchir à toute la logique de conception d'un programme. Cet aspect n'est pas abordé dans ce manuel, mais je vous encourage à lire d'autres supports comme *la Conduite de projets* ou *l'Etude préalable* (disponible sur www.beaussier.com).

3.2. Limitations de Windev

Windev s'appuie d'ailleurs sur une méthodologie de type Merise et depuis la version 7, est supporté la méthode UML. Ce manuel traitant principalement de l'approche d'un débutant face à l'AGL, nous n'aborderons pas ces méthodes d'analyse dans le détail.

Rappelons enfin que Windev a des limites. Que l'AGL ne peut pas servir à créer n'importe quel type d'application. Si Windev permet la création de programmes de gestion relativement puissants et rapides, il pêchera dans d'autres domaines (temps réel, multimédia...).

On ne peut donc pas tout faire, ce qui ne veut pas dire que le produit soit limité. L'éditeur a prévu trois ouvertures pour donner à Windev plus de puissance :

- La possibilité d'intégrer des modules externes du type *ActiveX* qui sont des contrôles supplémentaires apportant d'autres possibilités immédiatement utilisables. Par exemple, vous pouvez intégrer à votre programme un navigateur HTML ou un lecteur de PDF.
- La gestion des objets OLE qui est un protocole d'échange de données et de commandes dont le principe consiste à imbriquer et lier des objets. Par exemple, vous pouvez intégrer à votre programme le traitement de texte MS-Word ou le tableur MS-Excel.
- L'appel toujours possible aux bibliothèques dynamiques (DLL) et en particulier celles de Windows avec les fonctions API (*Application Programming Interface*) pour contourner les limites du langage.

3.3. Limitations du support

Ce support dans un soucis de simplification, n'aborde pas certains aspects jugés complexes ou trop pointus pour démarrer avec Windev. Il s'agit de :

- Le module d'analyse `WDAna` et les méthodologies associés comme Merise ou l'UML.
- La base de données propriétaire Hyper File et les fonctions de gestion de cette base.
- Le module d'état `WDEtat` et la fabrication des états à imprimer.
- La programmation orientée objet (POO) et la fabrication des classes.

3.4. Le langage

Par rapport à d'autres langages, Windev apporte une simplification considérable avec un langage intuitif : le fameux *W-Langage*. D'abord parce qu'il est en français et qu'il est composé d'ordre facilement mémorisable : `Ouvre` ouvrira tout simplement une fenêtre.

Remarque :
Pour les anglophiles, le *W-Langage* deviendra le *W-Language*. Les ordres sont bilingues avec par exemple, `Open` pour `Ouvre` ou `Close` pour `Ferme`.

Ensuite parce que Windev est enrichi de nombreux assistants à la saisie du code qui facilite la création de procédures complexes.

4. Philosophie

Windev est fondé sur la notion d'objet, ou plus précisément de programmation événementielle. Un programme sous Windows est donc d'abord et avant tout, une interface munie d'objets (des contrôles le plus souvent). Chaque composant est doté de propriétés et réagit à des événements.

Un programme Windev, comme n'importe quel programme informatique, implique deux étapes distinctes :

- Une étape d'écriture et de mise au point (débogage) qui s'appelle plus simplement la conception. On dira donc « à la conception » ou « en mode conception » pour signifier une action faite ou initiée par le développeur.
- Une étape d'utilisation ou de test. On dira donc « à l'exécution » ou « en mode exécution » pour signifier que l'on fait fonctionner le programme en mode utilisateur et non plus en développement.

4.1. Le projet

L'ensemble d'une application Windev, lorsqu'elle n'est pas compilée (en mode conception), porte le nom de projet. Le projet rassemble ainsi un ensemble logique d'objets et de traitements dont le but est de réaliser un objectif donné. Le projet gère les relations des objets entre eux.

Techniquement un projet est un assemblage de fichiers. Ce super fichier porte l'extension `.WDP` (abréviation de *Windev Développement de Projet*). Une fois ouvert dans l'AGL, ce fichier de projet référence tous les fichiers dont Windev a besoin pour travailler.

Les fichiers composants un projet peuvent être de plusieurs natures :

- Les fenêtres bien sûr (`.WDW`)
- La feuille de style (`.WDS`) qui contient le style de chaque composant (couleur, police, forme)
- Les états à imprimer (`.WDE`) et réalisés avec `WDEtat`
- Les classes (`.WDC`) et les collections de procédures (`.WDG`)
- Tout autre fichier extérieur à Windev comme les images et icônes qu'utilise l'application

Astuce :
Pour afficher la liste des composants d'un projet, sélectionnez le menu **Projet** / **Liste des composants du projet**.

Enfin, lors de la création d'un projet, Windev crée une structure spécifique de sous-répertoires décrite comme suit :

- `NomProjet.WD7` pour l'analyse s'il y en a une et ses différentes versions.
- `EXE` qui contiendra le programme et les fichiers nécessaires à une version cliente.
- `NomProjet.CPL` pour le stockage du code compilé.
- `Sauvegarde` pour les copies de sauvegarde de tous les fichiers fabriqués.
- `Tâches` pour le stockage des tâches à exécuter dans le cadre d'un projet développé à plusieurs.

4.2. Les composants

Le concept d'objet en programmation a un sens particulier. Pour ne pas trop corrompre le terme, on parle plutôt sous Windev de composants. Il s'agit ici d'une belle abstraction qui demandera un petit effort au novice.

Un composant est une interface graphique pour une fonction précise. Derrière cette interface, il existe un certain nombre de propriétés et de traitements. Chaque traitement peut gérer un évènement à travers du code.

Remarque :
Sous Windev, on parle plus de champ que de composant. Pour des raisons de clarté, on utilisera indifféremment les deux termes.

Prenons par exemple, le composant bouton qui est largement utilisé dans les interfaces sous Windows.

Les propriétés sont codés par le nom du composant et du nom de la propriété séparés par deux points de suite. Ainsi le changement du libellé d'un bouton se codera

```
monBouton..Libellé = "Test"
```

Le détail des propriétés se résume comme suit (liste non exhaustive) :

- Le positionnement du composant avec `Largeur`, `Hauteur`, `Ligne`, `Colonne`
- L'aspect avec `Libellé`, `Couleur`, `Etat`

Il possède deux évènements dont le traitement sera différent :

- L'initialisation qui permet d'exécuter du code avant l'apparition du bouton. Ce traitement s'exécute sans intervention de l'utilisateur.
- Le clic qui permet d'exécuter du code lorsque l'utilisateur appuie sur le bouton. Ce traitement ne s'exécute qu'avec l'accord de l'utilisateur.

Il existe également d'autres évènements qui sont liés cette fois à l'interaction des périphériques souris et clavier. On peut ainsi exécuter du code si le bouton est survolé par la souris. Dans la majorité des cas, ils ne sont pas utilisés sauf à ajouter des fonctionnalités plus élaborés.

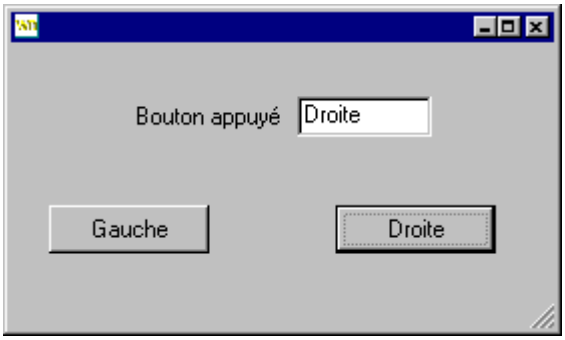
Enfin, on notera également qu'un composant peut contenir d'autres composants. Une fenêtre contiendra par exemple un ensemble de champs texte et de boutons.

5. Windev par l'exemple

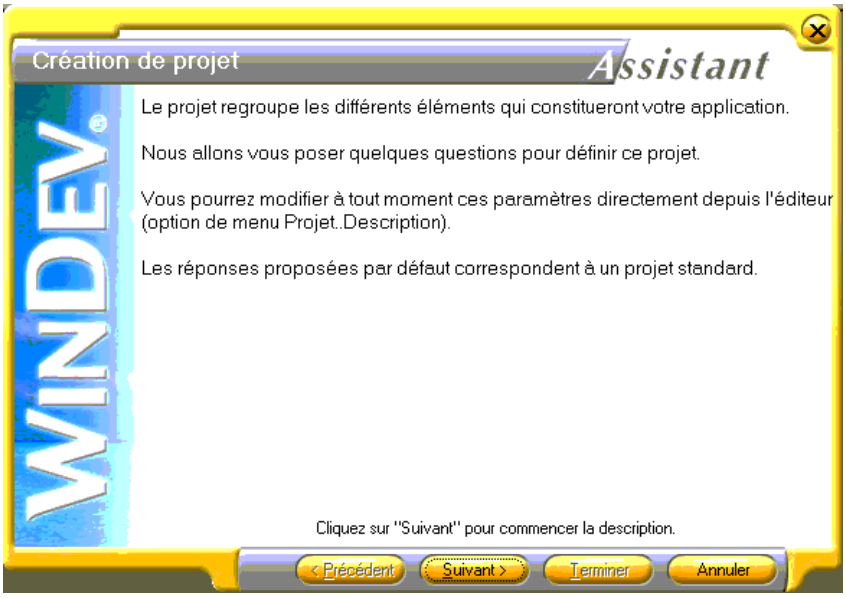
5.1. Notions élémentaires

Pour notre première application, nous allons créer une fenêtre avec trois composants (deux boutons et un champ texte).

Dans ce programme, le texte change au-dessus en fonction du clic de l'utilisateur sur le bouton *Gauche* ou *Droite*.



Avant toute chose, il vous faut créer un projet. Pour cela, procédez comme suit :
Sélectionnez le menu **Fichier** / **Nouveau** / **Projet.** L'assistant de création de projet s'ouvre.

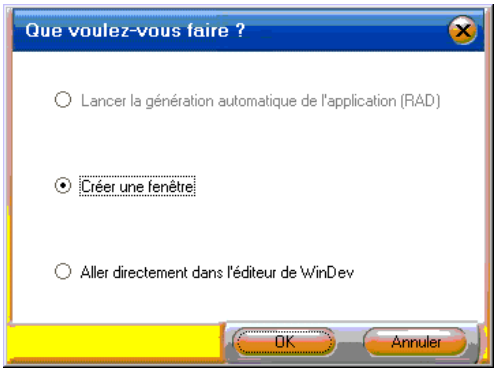


Spécifiez alors les informations voulues dans les différentes étapes du projet.

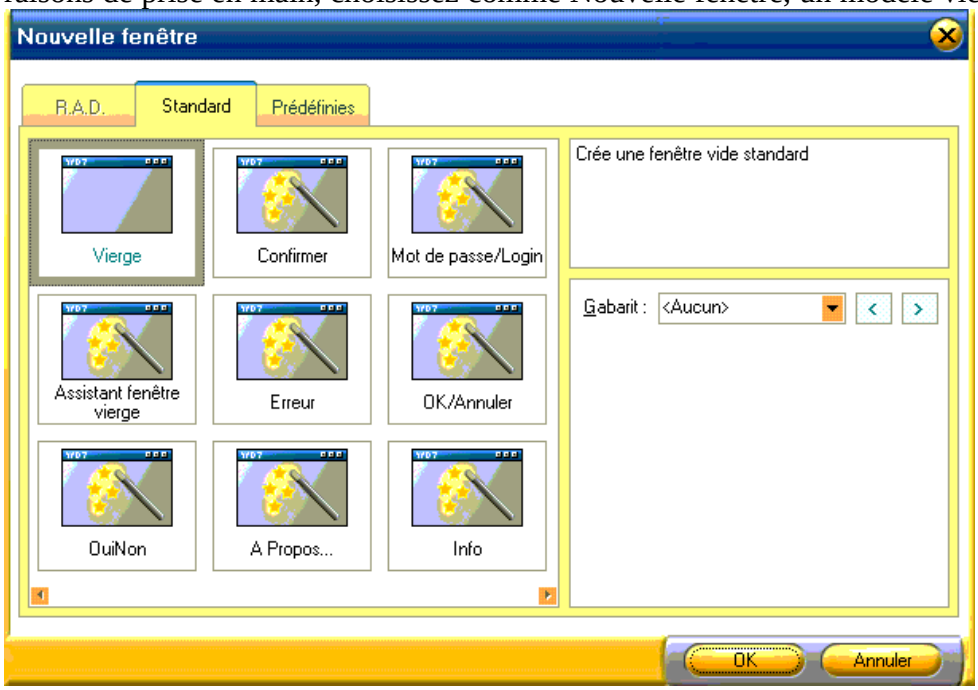
- Le nom et le répertoire du projet. Ces options ne seront pas modifiables. Tous les composants associés au projet seront créés dans le répertoire spécifié. Pour notre exemple, nous donnerons comme nom et comme répertoire : *TestWd*.
- Cochez l'option « Aucune analyse » car nous n'avons aucun besoin de fichiers de données pour cet exemple.

- Laissez « Aucun » pour le gabarit. Un gabarit permet de personnaliser l'interface de vos applications en leur donnant une ergonomie professionnelle grâce à différentes modèles prédéfinies.
- Laissez le Français comme seul langue gérée et comme langue principale.

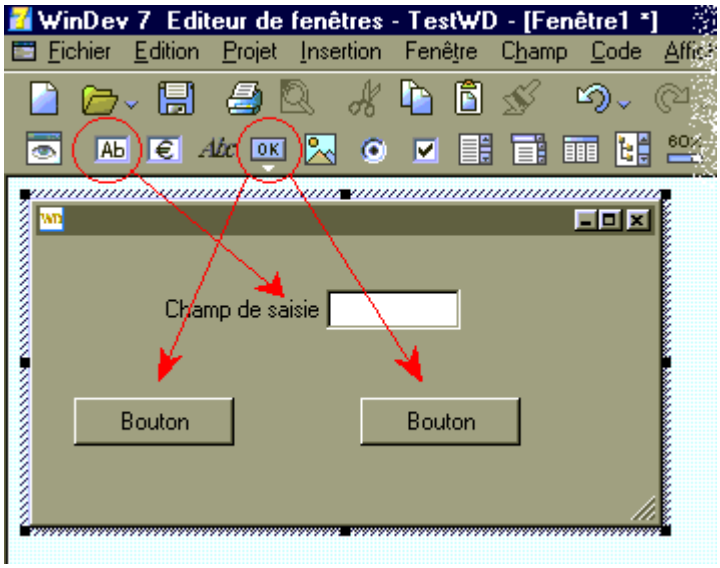
Le projet créé devient le projet en cours et s'affiche dans la barre de titre de Windev. Une boîte s'affiche pour vous demander la création d'une fenêtre.



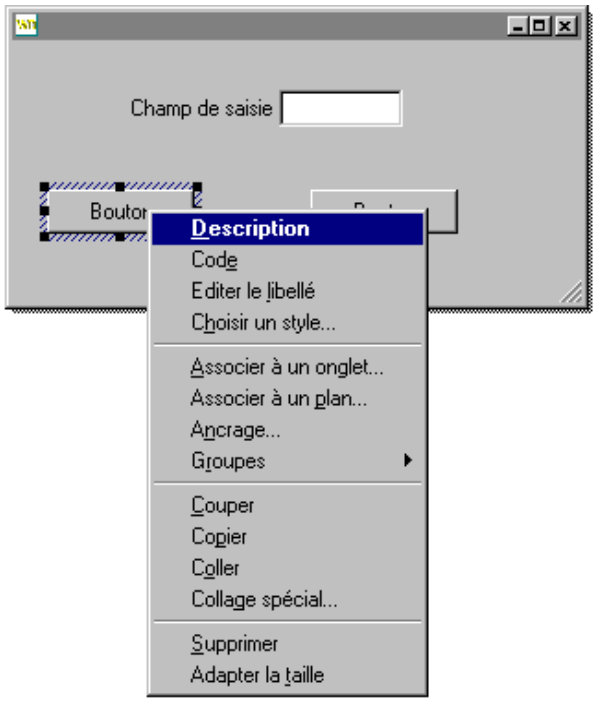
Pour des raisons de prise en main, choisissez comme Nouvelle fenêtre, un modèle vierge.



Une fenêtre vide de type standard apparaît dans l'éditeur de fenêtre de Windev. Modifiez la taille de la fenêtre en agissant sur les poignées et placez sur celle-ci un composant « champ de saisie » et deux « boutons ».



A chaque création, les composants portent des noms par défaut, la fenêtre s'appelle *Fenêtre1*, les boutons *Bouton1* et *Bouton2*, etc. C'est ce nom que vous devrez utiliser lorsque vous allez coder vos évènements.



Il est évident que de garder de tels noms ne sera pas aisé pour la compréhension du code par d'autres développeurs. Il est donc indispensable de les changer.

Appelez le volet des propriétés d'un composant par un clic droit sur celui-ci puis de choisir **Description** dans le menu contextuel.

Si vous êtes libre de nommer vos composants comme bon vous semble, je vous encourage à les préfixer. Ainsi pour les boutons, vous mettrez un *b* minuscule pour signifier qu'il s'agit d'un Bouton. Même chose pour les champs de saisie où vous ferez commencer chaque nom par un *s* minuscule.

Dans la propriété *Nom*, changez *Bouton1* pour *bGauche* et la propriété *Libellé* avec « Gauche ». Faîtes la même chose pour *Bouton2* avec *bDroite* comme *Nom* et « Droite » comme *Libellé*. Comme *Nom* pour le champ *Saisie1*, donnez *sResultat* et comme *Libellé* « Bouton appuyé ».

Enregistrez maintenant votre projet en cliquant sur l'icône « **Disquette** » ou en appuyant sur *Ctrl + S*. Confirmez l'enregistrement de la fenêtre en validant ou en modifiant son nom.

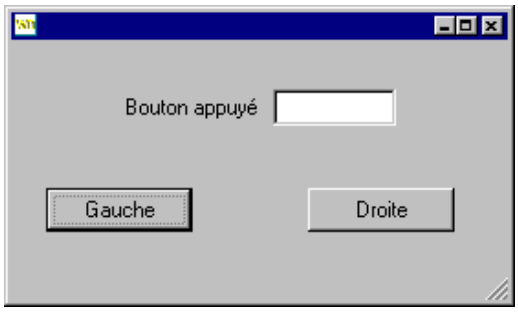
Remarque :
Comme toujours et c'est valable pour toutes applications informatique, il est recommandé de sauvegarder régulièrement.

5.2. Premier test

Nous allons maintenant tester notre application en lançant le mode exécution. Pour cela, choisissez le menu **Projet** / **Tester le projet** ou appuyez sur *Ctrl + F9*.

La première fois que vous lancez un test, Windev vous demande quel sera la première fenêtre à faire apparaître. Choisissez l'unique fenêtre de votre projet.

Une fois validé, Windev se met en icône et lance votre projet. La fenêtre de votre application apparaît alors.



Bien sûr le fait de cliquer sur les boutons ne déclenchent rien. C'est normal car jusque là, nous n'avons pas créer le code évènementiel des boutons.

En revanche, tout ce qui concerne l'interface graphique fonctionne. Vous pouvez ainsi déplacer la fenêtre, la dimensionner, la mettre en icône ou la maximiser. Vous pouvez également la fermer (bouton `[x]`).

Cette dernière action va mettre fin au test et rendre la main à Windev.

5.3. Coder les évènements

Comme on l'a préciser plus haut, chaque composant doit être codé pour réagir et interagir avec l'utilisateur. Il s'agit donc de construire des couples composant/évènement.

L'accès au code d'un composant se fait en sélectionnant celui-ci (un seul clic) et en appuyant sur la touche F2. L'éditeur de fenêtre Windev se transforme alors en éditeur de code.

Dans notre projet, seuls les boutons auront leur évènement « Clic » codifié. Commençons par le code du bouton *bGauche*, cliquez une fois et appuyez sur F2. Dans l'éditeur de code, saisissez le texte :

```
Clic sur bGauche
// Affichage dans le champ
SResultat = "Gauche"
```

Histoire de voir qu'il n'y a pas qu'un seul moyen d'arriver au même résultat, nous allons coder différemment le bouton *bDroite*.

```
Clic sur bDroite
SResultat = bDroite..Libellé
```

L'éditeur de code de Windev 7 est devenu assez sophistiqué. Il vous suggère au fur et à mesure de la saisie, les fonctions du *W-Langage*, les noms des composants ou leurs propriétés associées.

Enregistrez votre projet et relancez un test par *Ctrl + F9* pour vérifier que tout fonctionne correctement.

5.4. Procédure ou fonction

Afin de ne pas enfermer du code dans un couple composant/évènement, il est possible de définir des procédures ou des fonctions. Celles-ci permettent d'économiser du code qui serait exécuter dans plusieurs traitements avec des modifications minimes.

Quelle est la différence entre une procédure et une fonction :

- Une **procédure** ne retourne pas de résultat.
- Une **fonction** retourne un résultat.

Pourtant en *W-Langage*, il n'existe pas de distinction entre les deux termes. Les procédures et les fonctions sont gérées de la même façon. Une procédure comme une fonction peut retourner ou non un résultat. Le compilateur de Windev ne posera pas de problème.

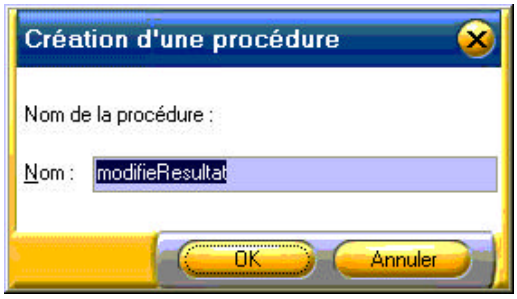
Ensuite une procédure ou une fonction peut être de niveau global ou local. Si elle est déclarée comme globale, son code sera accessible partout dans le projet. Si elle est déclarée comme locale, son code ne sera accessible que dans la fenêtre où elle se trouve.

Important :
Il ne faut pas déclarer deux procédures ou fonction avec le même nom (notamment une globale et une locale).

Dans notre projet, nous allons réaliser une procédure locale que nous allons placer dans les deux évènements clic des boutons. Placez vous dans le traitement clic du bouton *bGauche* :

Clic sur **bGauche**
```
// Appel d'une petite procédure
modifieResultat("G")
```

Saisissez à la place du code existant le nom de la procédure et appuyez sur la touche *F4*.



Une boîte de dialogue s'affiche avec le nom de la procédure que vous avez tapé.

Validez pour ouvrir l'éditeur et en saisir son code.

Procédure locale **modifieResultat**
```
PROCEDURE modifieResultat(param)
SI param = "G" ALORS
      sResultat = "Gauche"
SINON
      sResultat = "Droite"
FIN
```

N'oubliez pas le traitement clic du bouton *bDroite* :

Clic sur **bDroite**
```
// Appel d'une petite procédure
modifieResultat("D")
```

Détaillons ensemble le code. La procédure est appelée dans le traitement clic. On lui passe un paramètre, un caractère qui identifie le bouton. Puis dans la procédure, on récupère ce paramètre et on en teste la valeur pour afficher le bon résultat.

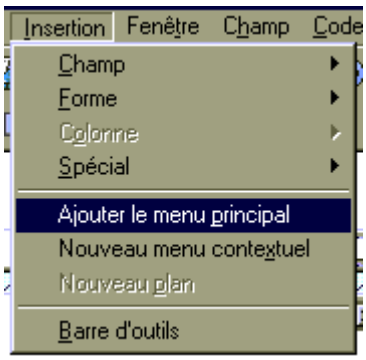
Bien sûr, l'exemple a été volontairement complexifié car on aurait pu directement passé le libellé complet comme paramètre : `modifieResultat("Droite")`

Mais dans ce cas, le test dans la procédure aurait été inutile et la procédure elle-même n'aurait plus de justification.

5.5. Amélioration de l'interface

En règle général, une application possède une barre de menu. Celle-ci permet à l'utilisateur de retrouver toutes les fonctions du programme.

Pour notre exemple, nous allons créer un menu ultra simple. Une barre de menu principal qui contiendra deux éléments : **Fichier** et **Aide**. Dans le menu **Fichier**, deux éléments **Exécuter** et **Quitter**. L'élément **Exécuter** aura deux sous-éléments **Gauche** et **Droite** qui feront la même chose que les boutons du même nom. Dans le menu **Aide**, un élément **A propos**.



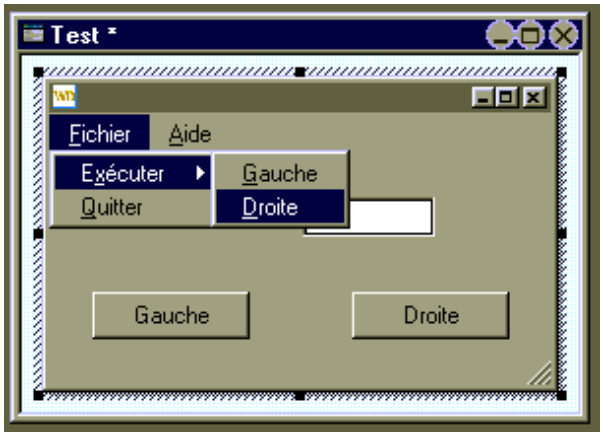
Placez vous sur votre fenêtre et appelez le menu *Insertion* / *Ajouter le menu principal*

Sur votre fenêtre apparaît par défaut une barre de menu avec le mot **Menu**. A partir de là, vous allez gérer votre menu directement sur votre fenêtre.

Vous pouvez utiliser le menu contextuel sur le composant menu mais il est plus simple de créer ses éléments avec le clavier.

Voici quelques raccourcis :

- La barre d'espace pour changer un libellé d'élément
- La touche de tabulation pour créer un élément à droite de l'élément courant
- La touche Entrée pour créer un élément sous l'élément courant
- Les flèches pour se déplacer dans les éléments de menu créés



Rappel :
N'oubliez jamais qu'un élément de menu doit avoir une lettre d'appel (souligné). Vous devez faire précéder cette lettre du symbole &.

Nous allons maintenant coder chaque élément. Pour cela, cliquez sur l'élément désiré et appuyez sur la touche *F2*.

Sélection du menu de **_Menu._Fichier._Quitter**
```
Ferme() // Quitte l'application
```

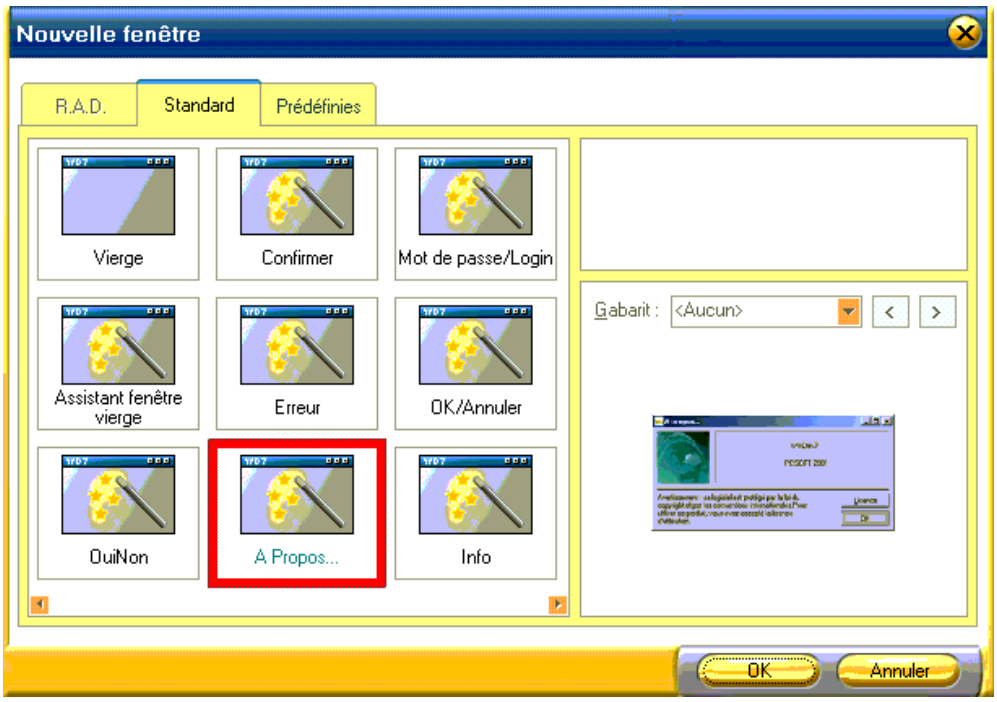
Sans commentaire, reportez vous à l'aide en ligne pour cette fonction.

Sélection du menu de **_Menu._Fichier.E_xécuter._Gauche**
```
ExecuteTraitement(bGauche,trtClic)
```

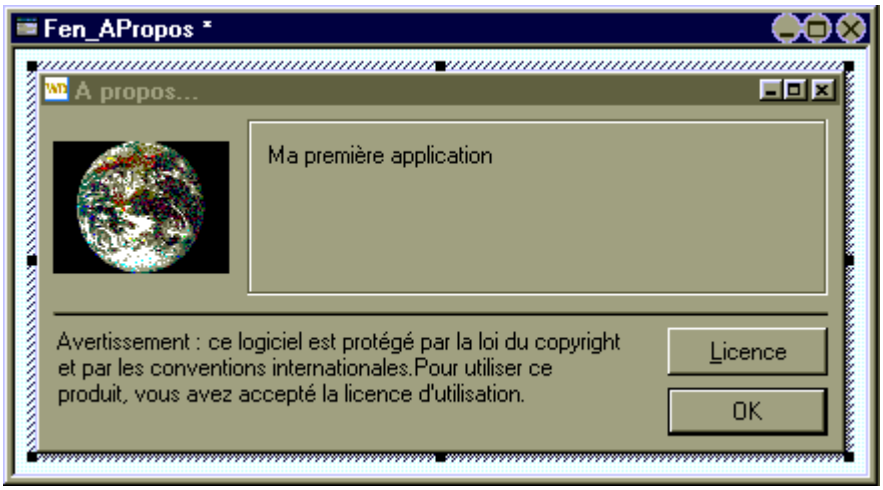
Sélection du menu de **_Menu._Fichier.E_xécuter._Droite**
```
ExecuteTraitement(bDroite,trtClic)
```

Il est plus simple d'appeler l'évènement d'un autre composant que de faire une copie de son code. En effet si vous devez modifier les lignes du traitement, vous n'aurez à le faire qu'à un seul endroit.

Pour l'option de menu « A propos », nous allons d'abord créer une nouvelle fenêtre. Appuyez sur *Ctrl + N* et choisissez l'option *Fenêtre* et sélectionnez la fenêtre **A propos**.



Renseignez les question que vous pose l'Assistant. Une fois la fenêtre créée, enregistrez la.



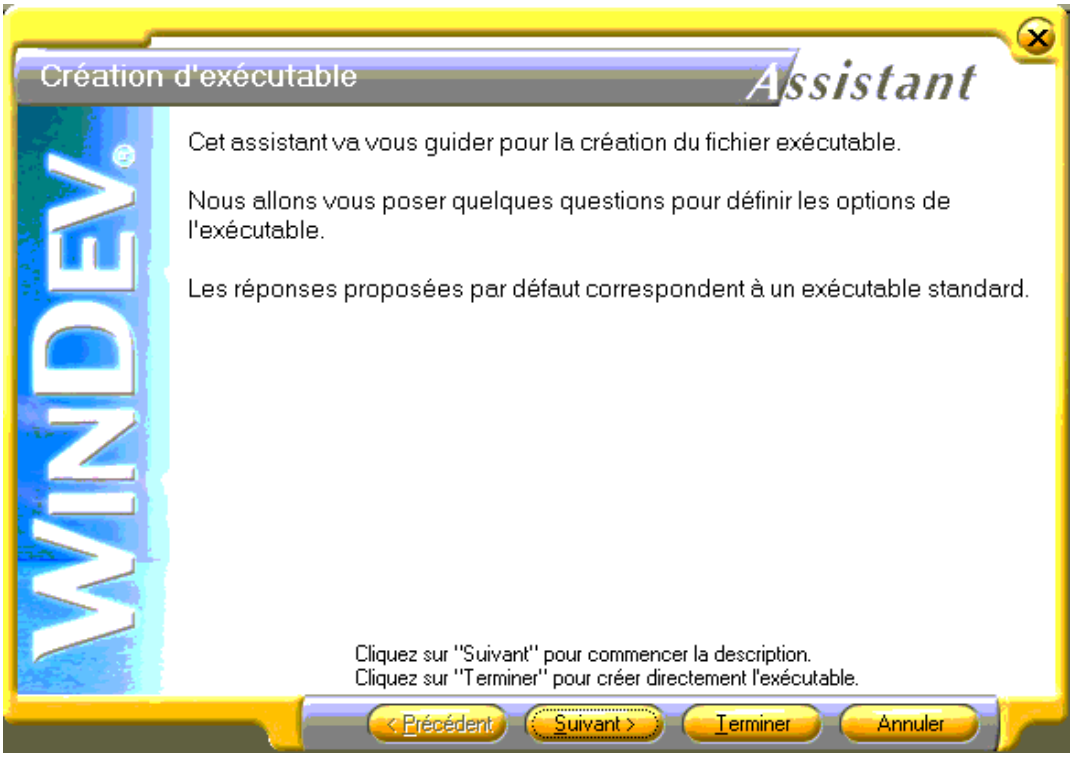
Revenez enfin sur votre fenêtre principale et appelez le code de l'élément de menu **A propos.**

Sélection du menu de **_Menu._Aide._A_propos**
```
Ouvre(Fen_Apropos)
```

5.6. Finition

Il s'agit maintenant de transformer votre projet pour en faire un fichier exécutable autonome. Pendant cette opération, l'application recevra quelques finitions qui lui donneront un aspect un peu plus professionnel.

⇦ Cliquez sur ce bouton pour lancer l'Assistant de création de l'exécutable.



Cliquez sur le bouton « Suivant » et appelez le catalogue pour ajouter une icône à votre application.

Saisissez un mot clé pour afficher des icônes.

A l'étape du *fonctionnement de l'exécutable*, sélectionnez un fonctionnement mono-instance et une intégration de la bibliothèque dans l'exécutable.

Cette bibliothèque est un fichier d'extension `.WDL`. Il ne faut pas la confondre avec les bibliothèques de Windows `.DLL`. La bibliothèque regroupe tous les objets utilisés par votre projet :

- Le projet lui-même.
- Toutes les fenêtres, requêtes, états intégrés au projet.
- Les fichiers images utilisés initialement pour les composants images, boutons ou fenêtres.
- Le fichier de description des données, si une analyse Hyper File est gérée.
- D'autres fichiers dont vous fournissez la liste.

Une fois la dernière étape franchie, l'Assistant va créer les fichiers nécessaires pour la version cliente de votre application. Ces fichiers se trouvent dans le sous répertoire `EXE` de votre projet.

Cyril Beaussier
Gestion de projet informatique

Les méthodes d'estimation

Version 1.1 - Juin 2001

Relecture et corrections : Evelyne Henry

Sommaire

Introduction

Ce support est avant tout écrit pour la gestion de projet développé avec Windev, néanmoins il peut s'adapter à d'autres AGL graphiques du marché. Vous risquez cependant de ne pas trouver la même terminologie et les coefficients risquent également de varier d'un outil à un autre.

L'estimation du coût d'un projet informatique est un art périlleux. Il coûte toujours plus cher que prévu, dure plus longtemps qu'on ne l'imaginait à l'origine.

Il est impossible de sortir une estimation financière d'un chapeau. Celle-ci doit se faire le plus sérieusement possible suivant les critères suivants :

- ✓ On doit disposer d'une définition précise des fonctions à développer en se basant sur le **cahier des charges**.
- ✓ On doit examiner des éléments significatifs comme des écrans, des états ou le modèle de la base de données.
- ✓ On doit connaître les contraintes comme le niveau technique des développeurs, les exigences des utilisateurs ou l'environnement matériel.

C'est avec ces éléments que l'on pourra appliquer une méthode d'estimation.

Les principaux écueils

La mauvaise estimation des charges est la principale raison. Les délais qui sont bâtis dessus souffrent en conséquence d'une mauvaise planification des tâches à réaliser.

La tendance à la sous-estimation est fréquente dans une gestion de projet. Elle trouve son origine parmi les causes suivantes :

- ✓ Le désir de plaire
- ✓ Le besoin de gagner
- ✓ L'optimisme
- ✓ L'expérience limitée des évaluateurs
- ✓ L'oubli des utilitaires (sauvegarde ou restauration)
- ✓ L'oubli de la documentation
- ✓ La mauvaise estimation des efforts de mise au point
- ✓ L'absence de méthode ou de standard

Il arrive aussi de faire des erreurs, pas seulement dans le calcul de la charge mais aussi dans le calcul du délai et ceci à cause de :

- ✓ Les ressources ne peuvent pas être affectées également à chaque phase.
- ✓ Il faut respecter les contraintes d'enchaînements.
- ✓ Certains temps sont incompressibles.
- ✓ La communication interne prend du temps.

Cela signifie qu'il faut se méfier de la notion de "mois/homme". Une charge de 100 mois/homme faisable en 10 mois avec 10 personnes ne le sera plus en 1 mois avec 100 personnes, ni en 100 mois avec 1 personne.

L'erreur la plus répandue combine les deux tendances : mauvaise estimation des charges PUIS mauvaise planification. En effet, lorsque le projet dérape, des renforts massifs ne peuvent que le perturber un peu plus. Ainsi il faudra que :

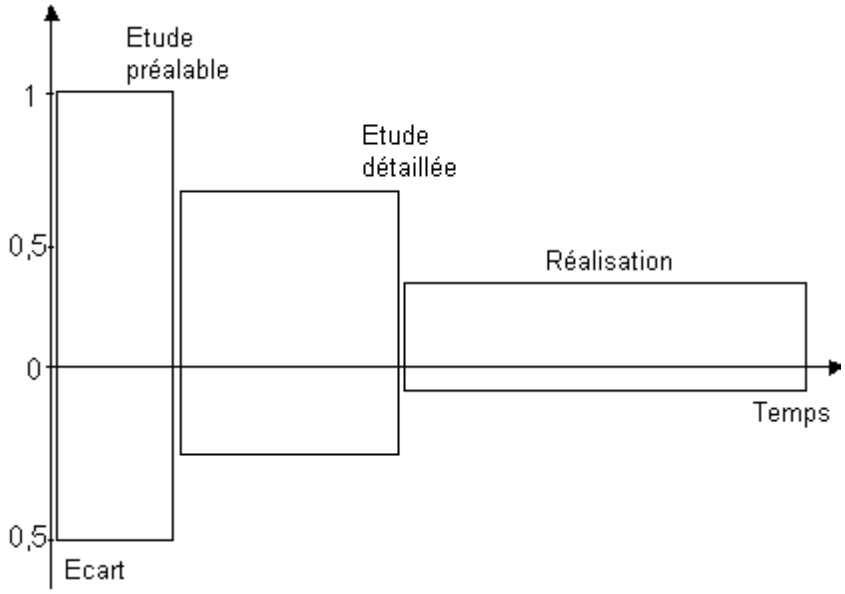
- ✓ Le planning soit retaillé.
- ✓ Une partie du travail soit refaite.
- ✓ Les temps de communication, d'encadrement et de formation augmentent.

Rappel de la loi de Fred Brooks :

"ajouter du monde dans un projet ne fait que le retarder un peu plus".

La procédure d'estimation

La précision des estimations s'améliore avec le temps.



Comment procéder efficacement à une estimation ? Plusieurs éléments permettent d'atteindre cet objectif :

1. La sélection d'une méthode.
2. L'expression correcte et complète du besoin à partir d'un cahier des charges détaillé.
3. La traduction de ce besoin en tâches quantifiables, ce qui revient à choisir une unité.
4. Tenir compte de l'environnement technique.
5. Connaître la productivité de l'équipe en s'appuyant si possible sur le référentiel des projets passés.
6. Exprimer les charges par phase.

Choisir une méthode

Si des méthodes que l'on nommera méthode "pifométrique" (approche par analogie, jugement d'expert) existent, l'estimation se fait surtout avec l'expérience des projets passés. Pourtant des méthodes dites algorithmiques permettent d'établir l'estimation d'un projet. Trois méthodes plutôt anciennes (datant des années 70-80) et remises au goût du jour, se dégagent : les 10 paramètres, la MCP et la méthode COCOMO.

Les 10 paramètres

Cette méthode implique de disposer du cahier des charges détaillé, du schéma de la base de données, de l'enchaînement des écrans et des modèles d'états.

Elle se décompose en trois phases :

1. L'évaluation quantitative par les 10 paramètres.
2. Le calcul de la valeur du coefficient lié au contexte de réalisation et au contexte d'analyse
3. Le calcul de la charge nette.

Evaluation quantitative

Cette phase consiste à comptabiliser les dix natures d'éléments en les pondérant à l'aide des coefficients figurant dans le tableau ci-dessous et à obtenir ainsi, la mesure de la complexité de l'application (ou **CPX**). En clair, vous devez compter le nombre de chaque paramètre de votre projet et le multiplier par le coefficient correspondant.

Paramètre	Coef.
1. Nombre de tables	3
2. Relation entre deux tables	4
3. Nombre d'index	2
4. Nombre de fenêtres de l'application	3
5. Nombre d'états en sortie	2
6. Formules et règles de gestion	2
7. Procédures ou fonctions	2
8. Classes créées	4
9. Classes importées	2
10. Modules de liaison	4

Ainsi, si votre projet compte dix fenêtres, vous aurez un total pour le paramètre 4 de (10 x 3) 30.

Explication des différents paramètres :

1. Nombre des tables de la base de données ou du SGBD ou encore nombre des fichiers de données Hyper File (.FIC) ou autre.
2. Pour les SGBD, nombre des relations entre deux tables ; pour l'HF, nombre des relations entre fichiers.
3. Nombre de données indexées. Attention si une même donnée apparaît plusieurs fois, elle ne compte que pour une.
4. Nombre des fenêtres Windows mère ou fille de l'application ainsi que les boîtes de dialogue. Mais ne doivent pas être comptées les boîtes élémentaires du système de type information (Info), erreur ou confirmation (OuiNon).
5. Nombre d'états en sortie écrit ou graphique.
6. Formules et règles de gestion établies telles que calcul d'intérêt ou barème d'amortissement.
7. Procédure ou fonction (globale ou interne) présente dans l'application.
8. Nombre de méthodes de classes créées.
9. Nombre de méthodes utilisées par des classes importées.
10. Nombre de modules de liaison utilisés par l'application tels que DLL, ActiveX, etc. Ne pas compter bien sûr les DLL de Windev.

Calcul des contextes de réalisation et d'analyse

On détermine la valeur du coefficient du contexte de réalisation (**CTR**) en additionnant les valeurs retenues pour les éléments du tableau ci-dessous.

	Coef. Mini	**Coef. Maxi**
1. Puissance de l'outil	-0,3	+0,4
2. Fiabilité du système	0	+0,2
3. Puissance du système	0	+0,4
4. Support système	-0,1	+0,1
5. Disponibilité pour test	0	+0,1
6. Langage utilisé	-0,2	+0,3

Moins le contexte est favorable et plus vous devez utiliser un coefficient élevé.

Explication des paramètres :

1. Vous devez calibrer les outils de développement avec la future application. Si l'AGL peut concevoir l'ensemble de l'application sans difficulté, le coefficient sera maximum.
2. Vous devez évaluer la fiabilité du système par rapport à votre projet. Par exemple, si votre application doit tourner sur un réseau, il est bon de savoir si la nouvelle charge sera absorbée correctement sans risque de panne.
3. La puissance du serveur est elle suffisante, de même les stations clientes pourront-elles faire tourner votre application dans de bonnes conditions ?
4. Pouvez vous compter sur un support technique en cas de difficulté ?
5. Lorsque le développement de l'application sera terminé, aurez vous les moyens et le temps nécessaires pour assurer les tests ?
6. Les développeurs maîtrisent-ils le langage utilisé ?

On détermine ensuite la valeur du coefficient lié au contexte d'analyse (**CAF**) en multipliant les valeurs retenues pour les trois éléments du tableau suivant.

	Coef. Mini	**Coef. Maxi**
1. Besoins connus	1	1,5
2. Utilisateurs multiples	1	2
3. Contacts directs	1	2

Explication des paramètres :

1. Si vous pensez que l'étude détaillée répond correctement aux besoins, le coefficient sera maximum.
2. L'application est monoposte, le coefficient sera au minimum puis en réseau vous augmenterez le coefficient de 0,1 par tranche de 10 utilisateurs.
3. Si lors du développement, vous êtes en contact direct avec l'utilisateur ou le responsable, appliquez le coefficient maximum.

Calcul de la charge nette

La charge nette de réalisation (**R**) et la charge nette d'analyse (**AF**) sont déterminées en appliquant les formules suivantes. Elles expriment un résultat en homme/jours.

Charge nette de réalisation : R = CPX x CTR

Charge nette d'analyse : AF = 0,1 x R x CAF

Exemple

Un petit exemple pour illustrer la méthode. Imaginons un projet devant développer une application de prise de commande client. Elle est en client/serveur[1] avec un SGBD de type SQL/Server tournant sur un serveur Windows NT. Elle sera utilisée par 12 utilisateurs au maximum. L'application sera entièrement développée sous Windev en utilisant l'accès ODBC. On notera que le serveur hébergera en plus de l'application, la messagerie de l'entreprise. Les ordinateurs des utilisateurs sont récents. Votre équipe (2 développeurs) maîtrise bien l'AGL et l'étude détaillée exprime bien les besoins.

8	tables	x 3	=	24
10	relations	x 4	=	40
40	index	x 2	=	80
40	fenêtres	x 3	=	10
5	états	x 2	=	10
4	formules	x 2	=	8
10	fonctions	x 2	=	20

En calculant la mesure de la complexité de l'application, on arrive donc à un **CPX de 192**. On calcule ensuite le coefficient lié au contexte de réalisation.

Puissance de l'outil	Excellente	0,4
Fiabilité du système	Moyenne	0,1
Puissance du système	Moyenne	0,2
Support système	Bon	0
Disponibilité pour test	Correct	0,1
Langage utilisé	Très bon	0,2

Le **CTR** sommé donne **1**.De la même façon, on calcule le coefficient lié au contexte d'analyse.

Besoins connus	1,5
Utilisateurs multiples	1,2
Contacts directs	2

Le **CAF** est multiplié pour donner **3,6**. Enfin on calcule la charge nette.

Réalisation : 192 x 1 = 192 hommes/jours
Analyse : 0,1 x 192 x 3,6 = 69 hommes/jours

L'application sera donc réalisée en un peu plus de 3 mois, analyse comprise.

[1] Il est conseillé de lire le support Client/Serveur en cas de non compréhension de certains termes.

La méthode M.C.P. de GEDIN

Cette méthode dont le sigle veut tout simplement dire Méthode de Conduite de Projets a été mise au point par M. GEDIN alors Directeur informatique de la RATP.

Elle propose un ensemble de critères pour estimer d'une manière très globale, l'importance des charges de réalisation d'une opération d'automatisation, dès l'étude d'opportunité. Elle permet d'évaluer avec une bonne approximation, compte tenu des imprécisions d'une étude d'opportunité, les temps à partir desquels sont obtenus les coûts et le calendrier de développement. La méthode peut également être appliquée au stade du cahier des charges. La conception étant alors presque achevée, la précision de l'estimation est meilleure.

Elle comprend six étapes :

1. Estimation de la complexité logique.
2. Evaluation de la difficulté pratique.
3. Détermination de la catégorie du projet.
4. Evaluation de la charge et du délai bruts.
5. Influence de la nouveauté.
6. Calcul final de la charge nette.

Estimation de la complexité logique

Elle se détermine à partir de six éléments auxquels on applique une note variant de A (simple) à D (complexe).

1. Nombre de fichiers permanents	
Inférieur ou égal à 2 fichiers	A
De 3 à 4 fichiers	B
De 5 à 6 fichiers	C
Supérieur à 6 fichiers	D
2. Architecture et complexité des fichiers	
Fichiers sans index	A
Fichiers avec index	B
Fichiers avec index composé	C
Fichiers intégrés (SGBD)	D
3. Relation avec d'autres applications	
Aucune, indépendance totale	A
Interface simple	B
Systèmes couplés	C
Systèmes complexes	D
4. Perspectives d'évolutions prévues	
Le système évoluera peu	A
Développement connu	B
Flexibilité totale	C
5. Ampleur de l'automatisation	
Inférieur ou égal à 5	A
Entre 6 et 9	B
Entre 10 et 40	C
Supérieur à 40	D
6. Connaissance de l'environnement	
Bon	A
Connu mais étude à prévoir	B
Entièrement nouveau	C

Explication des paramètres :

1. Combien de fichiers (.FIC) seront présents sur le serveur ou la station. Si un SGBD est utilisé, compter le nombre de tables.
2. Complexité des fichiers sans clé (A), avec clé simple (B) ou avec clé composée (C) ou la base sera un SGBD (D).
3. L'application sera monoposte (A), monoposte lié à d'autres applications (B), en réseau (C), en réseau lié à d'autres applications (D).
4. Quels sont les risques d'évolution dans l'année suivant la livraison de la version finale de l'application ?
5. Combien de fonctions, de procédures, de classes ou de traitements longs (édition, calcul…) figureront dans l'application ?
6. Quel est le niveau de compétence du Chef de projet dans le domaine où l'application évoluera ?

Estimation de la difficulté pratique

Elle se détermine à partir de quatre éléments auxquels on appliquera la même notation, de A (simple) à D (complexe).

1. Type de traitement	
Traitements en local	A
Quelques traitements déportés	B
Mode client/serveur	C
Entièrement déporté (mode internet)	D
2. Nombre de fonctions de l'entreprise	
Une seule fonction	A
Deux ou trois fonctions	B
Plus de trois fonctions	C
3. Connaissance des besoins	
Excellente	A
Très bonne	B
Assez bonne	C
Plutôt mauvaise	D
4. Variation des besoins	
Aucune ou presque	A
Sur des points limités et connus	B
Sur de nouveaux points	C

Explication des paramètres :

1. Vous devez déterminer l'emplacement des traitements.
2. Quel sera le nombre de fonctions touché dans l'entreprise par l'application.
3. Quel est le niveau de connaissance des besoins des utilisateurs.
4. Dans quelles proportions les besoins des utilisateurs vont-elles évoluer ?

Détermination de la catégorie du projet

On pondère ensuite les résultats précédents avec les coefficients suivants :

A = 1 ; B = 3 ; C = 6 ; D = 12

On totalise ensuite chacun des deux résultats avec :

Total de la complexité logique = X
Total de la difficulté pratique = Y

On détermine enfin la catégorie du projet à l'aide du tableau ci-dessous.

↓X - Y→	> à 30	24 à 29	20 à 23	16 à 19	13 à 15	10 à 12	6 à 9	< à 8
> à 45	13	12	11	10	9			
36 à 44	12	11	10	9	8	7		
28 à 35	11	10	9	8	7	6	5	
21 à 27	10	9	8	7	6	5	5	4
16 à 20	9	8	7	6	5	5	4	3
13 à 15	8	7	6	5	5	4	3	3
10 à 12		6	5	5	4	4	3	3
8 à 9			5	4	4	3	3	2
< à 8				3	3	3	2	1

Evaluation de la charge et du délai bruts

En fonction de la catégorie du projet, on détermine la charge brute moyenne (en mois/homme) et le délai brut moyen (en mois) à l'aide du tableau suivant :

Catégorie de projet	Charge brut moyenne	Délai brut moyen
13	450	20
12	300	18
11	200	15
10	120	13
9	90	11
8	40	10
7	25	9
6	20	8
5	14	7
4	9	6
3	3	4
2	2	2
1	1	2

Influence de la nouveauté

Il convient ensuite de pondérer les résultats obtenus à l'étape précédente par un coefficient exprimant la nouveauté du sujet. Celui-ci est déterminé par cinq éléments que l'on note de A à D.

1. Introduction de nouvelles techniques	
Non	A
Partiellement avec expérience limitée	B
Beaucoup	C
2. Utilisation de nouveaux équipements	
Non	A
Quelques uns, importance secondaire	B
Oui pour la plupart	C
3. Nouveau projet	
Non, simple transcription	A
Oui en partie	B
Oui en totalité	C
4. Première automatisation	
Non, utilisateurs habitués	A
Oui mais certains utilisateurs novices	B
Oui et les utilisateurs sont tous vierges	C
5. Nouveautés des concepts	
Non très peu	A
Quelques uns, importance moyenne	B
Oui, beaucoup de nouveaux concepts	C

On pondère les résultats avec les coefficients suivants :

A = 1 ; B = 3 ; C = 6 ; D = 12

On totalise les cinq éléments et on détermine la valeur du coefficient de nouveauté à l'aide du tableau suivant.

Total	**Coefficient de nouveauté**
< ou = à 5	0,6
6 à 9	0,8
10	1
11 à 15	1,2
16 à 20	1,4
21 à 24	1,6
25 à 30	1,8
> à 30	2

Calcul final de la charge nette

On multiplie enfin l'estimation brute par le facteur de nouveauté pour obtenir l'estimation nette.

On reprend l'exemple de la méthode précédente pour illustrer celle-ci.

1.	Estimation de la complexité logique		
	8 fichiers permanents	D	12
	Architecture SGBD	D	12
	Application indépendante	A	1
	Peu d'évolution	A	1
	10 fonctions et 4 formules	C	6
	Bonne connaissance des traitements	A	1
			33
2.	Evaluation de la difficulté pratique		
	Traitement C/S	C	6
	Un seul service impliqué	A	1
	Assez bonne connaissance	C	6
	Peu de variation des besoins	A	1
			14
3.	Type de projet		**7**
4.	Charge brute (en mois/homme)		**25**
	Délai brut (en mois)		**9**
5.	Influence de la nouveauté		
	Pas de nouvelles techniques	A	1
	Pas de nouveaux équipements	A	1
	En partie c'est un nouveau projet	B	3
	Les utilisateurs sont habitués	A	1
	Très peu de nouveaux concepts	A	1
			7
	Facteur de nouveauté		**0,8**
6.	Charge nette (en mois/homme)		**20**
	Délai net (en mois)		**7**

La méthode COCOMO

Mise au point par l'américain B.W. Boehm, Directeur de recherche logicielle au sein de la SSII TRW Inc. la méthode COCOMO (COnstructive COst MOdel) permet de couvrir l'ensemble des phases d'un projet informatique :

- Conception fonctionnelle externe
- Programmation
- Conception organique détaillée interne
- Ecriture du code et test
- Intégration et recette

Parallèlement la méthode couvre les activités liées à ces phases :

- Encadrement
- Conception fonctionnelle et architecture du système
- Gestion des configurations
- Assurance qualité
- Plan de test et jeux d'essai
- Documentation et manuel
- Maintenance corrective

En revanche, COCOMO a ses limites et ne couvre pas les activités suivantes qui devront être estimées par ailleurs :

- Etude de faisabilité
- Spécifications des besoins
- Installation
- Conversion, récupération des données et démarrage
- Formation
- Administration des données.

On notera enfin que COCOMO part du principe que le projet est bien géré tant du côté des informaticiens, que du côté des utilisateurs et que les spécifications du logiciel restent stables au moins au ¾.

Les formules de COCOMO n'utilisent qu'une seule variable le **KISL** pour Kilo d'Instructions Sources Livrées. Celui-ci sera la base de la composition du logiciel livré. COCOMO considère une ISL comme étant une ligne du source, quel que soit le nombre d'instructions qu'il peut y avoir sur cette ligne. C'est pourquoi, s'il y a une, deux ou trois instructions sur la même ligne, on ne compte qu'une ligne ISL et si une instruction est répartie sur cinq lignes on comptera cinq ISL. On ne tient pas compte bien sûr des commentaires, ni des lignes qui composent le fichier d'aide bien que son développement ait pu nécessiter une partie non négligeable des ressources.

A partir du nombre de KISL, les formules de COCOMO vous donnent les charges en (mois/hommes) et les délais (en mois).

Remarque : COCOMO est une méthode conçue à une époque où les AGL n'existaient pas et où tous les programmes étaient codifiés en langage de type procédural. Il faut donc ajouter le nombre de fenêtres qui compose le logiciel à la variable KISL en prenant pour facteur supplémentaire qu'une fenêtre (fichier WDW) vaut 0,5 KISL.

Mode de développement

B.W. Boehm a mis en évidence au cours de ses travaux, l'existence d'une forte corrélation entre le nombre de KISL et l'effort pour les concevoir, les produire, les mettre en œuvre et réaliser leur documentation. Il en a déduit deux choses.

D'une part, il faut à peu près les mêmes ressources humaines pour produire et mettre au point une application de 10.000 instructions sources (soit 10 KISL). D'autre part, l'erreur d'estimation faite par un professionnel est de plus ou moins 20 % pour la taille du logiciel et plus ou moins 300 % pour le temps de développement. Il est donc préférable de demander en premier lieu, le nombre d'instructions prévu.

Le tableau ci-dessous montre la répartition entre les trois grandes phases d'un projet suivant sa taille :

	PETIT PROJET (2 KISL)	PROJET MOYEN (30 KISL)	GRAND PROJET (500 KISL)
Conception	**16 %**	**17 %**	**18 %**
Programmation	**68 %**	**58 %**	**48 %**
Recette	**16 %**	**25 %**	**34 %**

COCOMO identifie également trois modes de développement dont dépendent les coefficients des formules et des tableaux utilisées ensuite pour calculer les charges et les délais.

Mode organique	Mode médian ou semi-détaché	Mode imbriqué ou intégré
Projet avec une petite équipe qui travaille dans un milieu familial sur des applications bien comprises. On perd peu de temps à communiquer, les membres savent ce qu'ils font et ils s'acquittent rapidement de leur tâche.	Equipe composé à la fois de personnes expérimentées et de débutants. Les membres ont une expérience limitée de ce type de projet et ne sont pas familiers de certains aspects du système qu'ils développent.	Développement de logiciels complexes. Il est pratiquement impossible de modifier les besoins pour contourner les problèmes et les coûts de validation sont élevés. Les membres de l'équipe n'ont pas une grande expérience.

Calcul de la charge brute

On détermine en premier lieu la charge brute du projet à l'aide de la formule suivante :

$$\mathbf{MH = A \times (KISL)^{B}}$$

Où MH représente la charge en mois/homme, A et B des coefficients dépendant du mode de développement. Voir le tableau ci-dessous :

Mode organique	Mode médian	Mode imbriqué
$MH = 2,4 \times (KISL)^{1,05}$	$MH = 3 \times (KISL)^{1,12}$	$MH = 3,6 \times (KISL)^{1,2}$

Remarque : un mois/homme représente 19 jours ou 152 heures.

Facteurs correcteurs

COCOMO identifie 15 facteurs correctifs ayant une influence mesurable sur la charge de réalisation.

		Facteur correcteur	Très faible	Faible	Moyen	Fort	Très fort	Except.
Produit	RELY	Fiabilité	0,75	0,88	1	1,15	1,4	-
	DATA	Volume des données	-	0,94	1	1,08	1,16	-
	CPLX	Complexité du code	0,7	0,85	1	1,15	1,30	1,65
Matériel	TIME	Temps d'exécution	-	-	1	1,11	1,3	1,66
	STOR	Taille mémoire	-	-	1	1,06	1,21	1,56
	VIRT	Stabilité système	-	0,87	1	1,15	1,3	-
	TURN	Temps de réponse	-	0,87	1	1,07	1,15	-
Personnel	ACAP	Compétence CDP	1,46	1,19	1	0,86	0,71	-
	AEXP	Expérience CDP	1,29	1,13	1	0,91	0,82	-
	PCAP	Compétence équipe	1,42	1,17	1	0,86	0,7	-
	VEXP	Expérience équipe	1,21	1,1	1	0,9	-	-
	LEXP	Expérience AGL	1,14	1,07	1	0,95	-	-
Projet	MODP	Utilisation méthode	1,24	1,1	1	0,91	0,82	-
	TOOL	Utilisation d'outils	1,24	1,1	1	0,91	0,83	-
	SCED	Contrainte de délai	1,23	1,08	1	1,04	1,1	-

Explication des facteurs :
RELY indique quelle doit être la fiabilité du logiciel. Très fort et exceptionnel pour des logiciels en temps réel où la vie humaine est mise en danger (guidage de missile).

DATA représente le volume des données hébergé sur le système. On appliquera la formule

R = nombre de Ko (kilo-octets) / KISL

Faible..... si R < 10
Moyenne..... si 10 < R < 100
Forte ..... si 100 < R < 1.000
Très forte ..... si R > 1.000

CPLX indique quelle sera la complexité des traitements. Compter un facteur exceptionnel en cas de programmation avec des classes (POO).

TIME indique l'occupation CPU tolérée. Moyen pour un taux de 50 %, forte pour 70 %, très forte pour 85 % et exceptionnel pour un taux de 95 %.

STOR indique l'occupation en mémoire de l'application. Même taux que pour TIME.

VIRT représente la stabilité de l'environnement si le logiciel doit subir des modifications dans l'année.

TURN indique le temps de réponse que le logiciel devra offrir à l'utilisateur.

ACAP correspond à la compétence du chef de projet ou des analystes.

AEXP correspond à l'expérience du chef de projet ou des analystes.

PCAP correspond à la compétence de l'équipe sur le projet.

VEXP correspond à l'expérience de l'équipe sur des projets du même type.

LEXP correspond à l'expérience de l'équipe dans l'AGL.

MODP indique si l'équipe utilise des méthodes d'estimation, de planification et de suivi pour le projet.

TOOL indique si l'équipe utilise des outils pour la planification et le suivi du projet.

SCED indique si le projet a des contraintes de délai vis-à-vis du client.

Calcul de la charge nette

On détermine ensuite la charge nette en multipliant la charge brute par le produit des 15 facteurs correctifs.

On pourra appliquer à cette charge nette, trois autres facteurs supplémentaires de minoration :

- L'instabilité des spécifications (jusqu'à 1,6)
- La qualité de gestion du projet (de 1 à 2)
- La qualité des relations avec les utilisateurs (de 1 à 2)

Calcul du délai total normal

Le délai total est une fonction dépendant d'abord de la charge totale nette puis du mode de développement. La formule est la suivante :

$$\mathbf{TDEV = 2{,}5 \times (MH)^{C}}$$

Où TDEV représente le délai total de développement (en mois) et c le coefficient dépendant du mode de développement.

Mode organique	Mode médian	Mode imbriqué
$TDEV = 2,5 \times (MH)^{0,38}$	$TDEV = 2,5 \times (MH)^{0,35}$	$TDEV = 2,5 \times (MH)^{0,32}$

Répartition des charges et des délais par phase

Il s'agit de ventiler les charges totales nettes pour chacune des grandes phases de développement. Cette répartition de la charge dépend à la fois de la taille du logiciel et du mode de développement. Le tableau ci-dessous est exprimé en pourcentage.

Charges	Petit 2 KISL	Interméd. 8 KISL	Moyen 32 KISL	Grand 128 KISL	Très grand 512 KISL
MODE ORGANIQUE					
▪ Planning	6	6	6	6	
▪ Conception	16	16	16	16	
▪ Programmation	68	65	62	59	
▪ Intégration et tests	16	19	22	25	
MODE MEDIAN					
▪ Planning	7	7	7	7	7
▪ Conception	17	17	17	17	17
▪ Programmation	64	61	58	55	52
▪ Intégration et tests	19	22	25	28	31
MODE IMBRIQUE					
▪ Planning	8	8	8	8	8
▪ Conception	18	18	18	18	18
▪ Programmation	60	57	54	51	48
▪ Intégration et tests	22	25	28	31	34

Il s'agit maintenant de ventiler le délai total dans chacune des 4 grandes phases du développement. Cette répartition est, elle aussi, dépendante à la fois de la taille du logiciel et du mode de développement. Le tableau est exprimé en pourcentage.

Délais	Petit 2 KISL	Interméd. 8 KISL	Moyen 32 KISL	Grand 128 KISL	Très grand 512 KISL
MODE ORGANIQUE					
▪ Planning	10	11	12	13	
▪ Conception	19	19	19	19	
▪ Programmation	63	59	55	51	
▪ Intégration et tests	18	22	26	30	
MODE MEDIAN					
▪ Planning	16	18	20	22	24
▪ Conception	24	25	26	27	28
▪ Programmation	56	52	48	44	40
▪ Intégration et tests	20	23	26	29	32
MODE IMBRIQUE					
▪ Planning	24	28	32	36	40
▪ Conception	30	32	34	36	38
▪ Programmation	48	44	40	36	32
▪ Intégration et tests	22	24	26	28	30

Exemple

Prenons à nouveau notre exemple d'application de prise de commande client et ses 40 fenêtres. D'après la complexité du projet, on estimera celui-ci avec la formule en mode organique en comptant un total de 25.000 lignes.

$$2,4 \times (25)^{1,05} = 70 \text{ mois/homme}$$

On calcule maintenant le produit des facteurs correcteurs :

		Facteur correcteur	Coef.
Produit	RELY	Fiabilité	1,15
	DATA	Volume des données	1,08
	CPLX	Complexité du code	1
Matériel	TIME	Temps d'exécution	1
	STOR	Taille mémoire	1,06
	VIRT	Stabilité système	1
	TURN	Temps de réponse	1,07
Personnel	ACAP	Compétence CDP	0,71
	AEXP	Expérience CDP	0,91
	PCAP	Compétence équipe	0,86
	VEXP	Expérience équipe	1,1
	LEXP	Expérience AGL	0,95
Projet	MODP	Utilisation méthode	0,82
	TOOL	Utilisation d'outils	0,83
	SCED	Contrainte de délai	1,08
			0,60

La charge nette est donc ramené à 70 x 0,60 = 42 mois/hommes.

On calcule ensuite le délai total normal avec $2,5 \times (70)^{0,38} = 12,56$ mois.

Pour la répartition de la charge de 42 m/h et le délai de 12,5 mois, on se positionnera sur le projet de type moyen :

	Charges	Délai
MODE ORGANIQUE		
▪ Planning	2,5	1,5
▪ Conception	6,7	2,4
▪ Programmation	26,1	6,9
▪ Intégration et tests	9,2	3,2

Conclusion

Ces méthodes nécessitent, elles aussi, un calibrage des coefficients sur des projets réels de l'entreprise. Elles reflètent bien, alors, l'expérience d'une équipe donnée dans un environnement donné et fournissent ainsi une estimation avec une bonne précision.

Cependant, il n'y a pas de méthode universelle, ni magique. Il conviendra toujours de faire l'analyse la plus détaillée possible afin de définir les tâches et leur enchaînement. La productivité est très variable selon les personnes, les méthodes et les outils. La difficulté technique entre aussi en ligne de compte mais il est certain qu'elle décroît avec la taille du projet.

Chaque fois que les enjeux le justifient (forfait, pénalité de retard…), il conviendra de faire plusieurs estimations à l'aide de plusieurs méthodes et en demandant l'avis à plusieurs estimateurs avant de s'arrêter sur une estimation raisonnable. Celle-ci servira de base à la planification, au suivi de projet. Il suffira de faire la comparaison de l'estimation au réel avec détection des écarts et correction de ceux-ci. Ce sera aussi à la fin du projet, un élément fondamental pour calibrer les règles, coefficients et formules qui permettront une meilleure estimation lors des projets ultérieurs.

Cyril Beaussier
Gestion de projet informatique

L'ÉTUDE PRÉALABLE

Choix et proposition de solution

Version 1.00 – Sept. 2000 – Nov. 2001

SOMMAIRE

PRÉAMBULE

J'ai commencé à travailler dans l'informatique en 1990. Depuis maintenant quelques années, j'ai été amené à conduire des projets informatiques dans le but de résoudre les problèmes posés par des clients que ce soit d'organisation, de gestion ou d'amélioration de systèmes existants. J'ai toujours été surpris par le manque ou l'absence de méthode de conception. De nombreux informaticiens se lancent à corps perdu dans un développement sans accorder la moindre attention à un travail d'étude en amont.
C'est ainsi que naissent des produits bancals, répondant peu ou mal au besoin initial. Combien de fois ai-je vu dans des forums, des questions posés seulement sur du matériel ou sur des logiciels à utiliser sans même poser les questions essentielles sur le besoin premier de l'utilisateur ou du client et sur les méthodes pour arriver à le satisfaire.
Le lecteur trouvera donc dans ce document de quoi alimenter sa démarche méthodologique, sous la forme d'idées et de fiches outils pour arriver à proposer des solutions plus fiables.

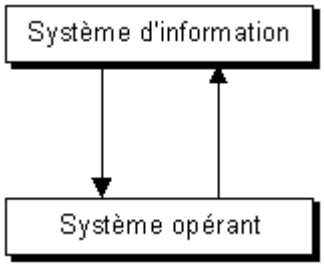
INTODUCTION

L'étude préalable s'inscrit dans un projet de création ou de modification d'un système d'information au sein d'une entreprise. C'est une étape importante, sinon primordiale dans la mesure où elle fournit des éléments permettant de prendre des décisions et d'engager des choix et surtout des budgets.
Toutefois, et en dépit de son importance vis-à-vis des orientations qui en découleront, l'étude préalable ne doit pas être trop consommatrice de temps ou d'énergie.
En conclusion, l'étude préalable doit être rapide et efficace. C'est pourquoi celle-ci doit être menée par des personnes organisées qui respecteront une démarche rigoureuse et en utilisant des outils méthodologiques.

Notons que chaque étude est unique et qu'il n'existe aucun modèle standard. En conséquence, une des premières tâches de l'étude préalable sera de définir et d'ajuster la démarche et la méthode de travail à utiliser.

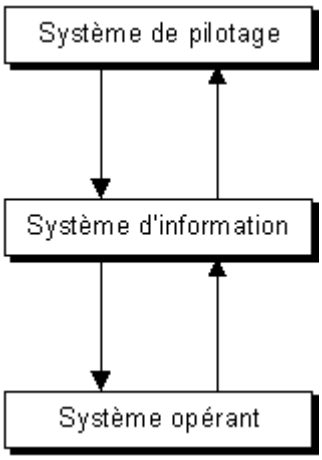
LE SYSTÈME D'INFORMATION

L'information est en elle-même impalpable. Elle n'est qu'une représentation symbolique et conventionnelle d'une réalité tangible (un client, du personnel, des produits) ou intangible (un client douteux, l'organigramme d'une société).
L'information permet d'enregistrer et de coordonner l'activité du système opérant. Par exemple : dans une société de vente par correspondance, l'opératrice qui prend une commande par téléphone peut vérifier si l'article demandé par le client est disponible.



L'information permet aussi de réguler l'activité du système opérant de l'entreprise. Par exemple : l'opératrice lance une commande si l'article est épuisé.

L'information devient alors un élément de transmission entre l'organe de décision (appelé Système de pilotage) et le système opérant. Le premier fixe des objectifs, mesure l'activité du système opérant et lance des actions de régulation.



En conclusion le système d'information représente non seulement l'information mais aussi tout ce qui est nécessaire pour le faire fonctionner.

L'information

Les informations comprennent des données qui sont elles-mêmes manipulées par des règles de gestion. Les informations représentent la mémoire de l'entreprise. Elles présentent un caractère stable lié à l'activité. Elles sont faciles à modéliser (avec des méthodes comme Merise ou UML).

Le traitement

Les traitements décrivent la manière dont les informations sont manipulées. A un haut niveau d'abstraction, les traitements présentent un caractère stable car lié à la finalité de son objectif. Par exemple : enregistrer la commande d'un client.

Plus on s'approche du système opérant, plus le traitement est décrit de façon concrète et plus on va y trouver de variantes. Ainsi l'enregistrement d'une commande client devient : remplir un bon de commande, en saisir les données dans le système, transmettre la commande au dépôt en fin de journée.

L'architecture technique et le support

L'architecture comprend la description des équipements matériels ainsi que les moyens de communication qui les relient.

Le support comprend la description des logiciels ou des progiciels utilisés ainsi que les documents servant de support aux informations (bon de commande, facture, bon de livraison…).

Les moyens humains

Les profils et les effectifs engagés selon les postes de travail.

La méthodologie de développement et d'entretien

La méthodologie de développement et d'entretien du système d'information comprend :

- ✓ Les équipes de développement et de maintenance, l'administration des données ou le support.
- ✓ La méthodologie de développement des applicatifs (méthode, documentation…).
- ✓ Les langages et les outils de conception et de développement.

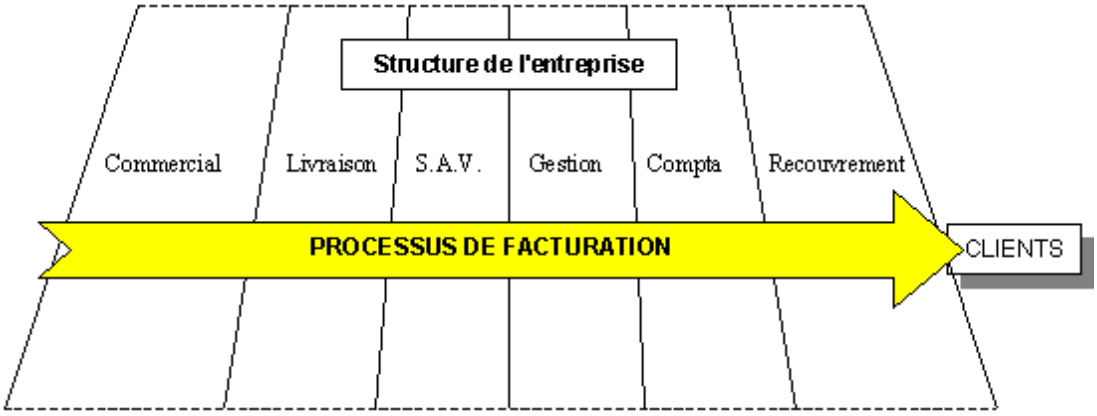
L'ÉTUDE PRÉALABLE

Comme nous venons de le voir, l'entreprise est un monde complexe. En comprendre les rouages est donc vital pour apporter des solutions claires qui répondront aux problèmes posés.

Objectif

Le but de l'étude préalable est de permettre d'effectuer des choix en terme d'organisation ou de solution technique. Pour cela, elle se limite en principe à un domaine précis de l'entreprise tout en prenant en compte la cohérence globale.

Ainsi, une étude portant sur l'amélioration de la facturation client ne touchera pas que le service qui établit les factures mais également les services en liaison avec celui-ci.
Elle examine, compare et modélise différentes solutions qui répondent au besoin et affine la planification des tâches à réaliser jusqu'à la fin du projet. Ces tâches sont couramment appelés des processus.



L'exemple ci-dessus montre l'enchaînement de tâches, activités ou opérations réalisées par des entités différentes. Cet enchaînement se fait à l'aide de moyens (équipement, matériel, procédure ou information) en traitant des objets de gestion (dossier, contrat, commande et facture) en vue de résultat attendu (ici le paiement).

Notons que dans le secteur administratif, l'approche par structure est souvent insuffisante pour garantir la cohérence et la coopération des services. L'approche par processus permet de fixer un cadre d'étude lié à un objectif plus précis que l'étude d'un maillon de structure.

Place

L'étude préalable fait souvent suite à un schéma directeur ayant préconisé des choix en termes d'architecture, de techniques et de méthodes au niveau de l'entreprise elle-même et fournissant un échéancier de projets à informatiser.

Ce schéma directeur n'est pas forcément un document officiel (surtout dans les PME et PMI). Il rassemble néanmoins les grands projets que l'entreprise aimerait implanter à plus ou moins longue échéance (une nouvelle gestion du personnel, un changement dans la stratégie de livraison…). L'étude préalable s'inscrit dans la suite d'un de ces projets qu'elle précisera et pour lequel elle permettra de comparer différents scénarios.
L'étude préalable sera suivie obligatoirement d'une phase détaillée selon le scénario qui sera retenu.

Durée

Nous l'avons vu, l'étude doit permettre la prise de décision, il est donc important de trouver un équilibre entre le niveau de détail qui risque d'entraîner un coût et une durée proportionnel.

En pratique, l'étude préalable dure de un à six mois suivant la largesse du domaine. Au deçà, on perdrait en précision sur la teneur des informations exposées. De la même manière, au delà, on aura intérêt à partager le domaine ciblé en sous-domaines ou à restreindre l'étude à un sous-ensemble précis représentatif.

PRINCIPES DIRECTEURS

Les principes directeurs sont des idées maîtresses qui orientent l'étude et la modélisation du domaine ciblé. De ces principes découlent souvent des démarches. Nous citerons pour la forme deux exemples qui sont l'approche système et l'analyse de la valeur. Nous parlerons de cela plus en détail dans les chapitres suivants.

L'approche système

Il s'agit d'une méthode d'observation et de représentation de la réalité. Celle-ci est basée sur l'idée que tout système :

- ✓ est un "lieu" de transformation,
- ✓ échange avec son environnement (les entrées sont transformées en sorties),
- ✓ est assujetti à un volonté finalisatrice et régulatrice de son activité.

L'approche système porte son attention sur les échanges entre le système et son milieu plutôt que sur les procédés utilisés pour la transformation.

L'analyse de la valeur

Il s'agit d'une méthode de conception de produits ou de services visant à optimiser les performances et la rentabilité. L'analyse de la valeur s'appuie plus précisément sur le fait que tout produit ou service remplit une ou plusieurs fonctions ne présentant pas toutes le même intérêt.

La valeur d'une fonction représente le prix que le client est prêt à payer pour l'acquérir. La démarche de RAD (*Rapid Application Development*) s'appuie sur ce concept.

L'analyse de la valeur s'appuie sur le double principe de :

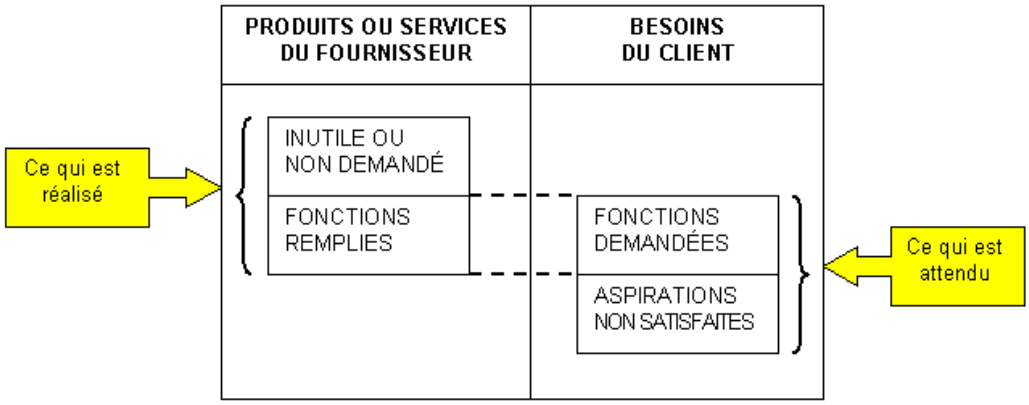
- ✓ Maximiser la satisfaction des besoins réels du client
- ✓ Minimiser les dépenses

La valeur d'un objet ou d'une activité se définit comme le rapport entre ce qu'il fournit et ce qu'il nécessite.

$$\text{VALEUR} = \frac{\text{APPORTS UTILES}}{\text{COÛTS + EFFORTS}}$$

En milieu administratif, les processus remplissent différentes fonctions pour répondre aux besoins des utilisateurs ou de l'entreprise : gestion des commandes, facturation, livraison. On peut donc analyser successivement les fonctions que le processus est censé remplir et les dépenses qu'il engage.

L'analyse de la valeur vise la remise en question des fonctions utiles ou non rentables. Elle permet d'optimiser les moyens par rapport aux fins. C'est une recherche systématique des opérations nécessaires et suffisantes pour que soient satisfaites les fonctions requises. Elle permet d'identifier les travaux de faible utilité. Des réalisations menées avec elle ont fait ressortir un accroissement de la satisfaction client de l'ordre de 10 à 35 %, tout en réduisant les charges de 20 à 50 %.



Typologie des fonctions

On distingue des fonctions à caractères différents. La fonction d'usage principale qui correspond à la vocation du processus ou de l'objet (établir une facture).
La fonction secondaire qui est utile pour l'usage mais non essentielle (classement de la facture).
La fonction d'estime liée à l'appréciation, au besoin affectif ou aux attentes subjectives (qualité de présentation).
La fonction de contrainte qui correspond à la réglementation (faire la déclaration de TVA).
Enfin, la fonction de construction ou d'assemblage servant uniquement à assurer les rouages avec les autres processus (photocopie et envoi à un autre service).

Démarche de l'analyse de la valeur

L'analyse de la valeur en milieu administratif obéit à une démarche pas à pas. Le tableau ci-dessous établit les différentes phases de cette démarche.

Phase	Démarche
0	Préparation et orientation préalable de l'étude ✓ S'informer sur la situation d'ensemble ✓ Identifier les principales activités ✓ Fixer le problème à étudier ✓ Définir les modalités de l'étude en constituant des groupes de travail
1	Détermination des fonctions du processus étudié ✓ Fonctions attendues par le client ✓ Fonctions fournies ✓ Critères d'évaluation des fonctions
2	Analyse descriptive du processus
3	Evaluation des coûts et valorisation du processus Cette valorisation est difficile en milieu administratif car elle nécessite de connaître un grand nombre de données : volumes, temps, montants financiers… On pourra se contenter d'évaluer sur des ordres de grandeurs réalistes ou des standards de l'entreprise.
4	Recherche des modifications ✓ Trouver de nouveaux moyens ✓ Faire des abandons
5	Evaluation des solutions et décisions
6	Préparation et mise en place des actions retenues
7	Bilan et suivi des résultats obtenus

PLAN D'UNE ÉTUDE

Réalisé par une démarche classique, l'étude préalable comprend cinq phases distinctes décrites ci-dessous.

Initialisation

Cette phase d'une durée relativement courte a pour but de préciser les frontières du domaine, les personnes concernées, les entités géographiques ou fonctionnelles qui sont à prendre en considération.

Le contexte de l'entreprise et les faits historiques précurseurs de cette étude (schéma directeur ou politique global) sont à prendre en compte. C'est au cours de cette phase que sont déterminés les objectifs du changement et les contraintes en termes de ressources (planning, budget, etc).

Étude de l'existant

L'étude de l'existant est avant tout une phase d'observation. Elle permet de recueillir des informations concrètes telles que les documents utilisées par l'entreprise, un organigramme du personnel, l'inventaire des données, des matériels, etc.

Ce recueil s'effectue le plus souvent lors d'entretiens avec les utilisateurs. Au cours de ces interviews, on laissera libre court aux personnes interrogées, la possibilité d'exprimer leurs opinions ou idées sur la situation.

Cette phase occupe le plus souvent la moitié du temps de la durée totale de l'étude. Elle doit donc être conduite avec rigueur et aboutir à une modélisation parfaite de l'existant.

Les entretiens

Dès les premiers entretiens, on va poser certaines questions pour identifier immédiatement ce qui ne fonctionne pas ou les améliorations à apporter au système en place. Ce questionnaire n'a pas précisément à être utiliser au cours de l'entretien mais il permet un travail préparatoire au questionnement et d'en faire une synthétisation des besoins.

Questions à poser aux acteurs internes du système

✓ Qu'est ce qui est insuffisant dans ce qui est fait aujourd'hui (manque de fonctionnalités, qualité douteuse, charge d'activité trop lourde…) ?

✓ Qu'est ce qui permet de constater que cela ne va pas ? Existe-t-il des critères ou des signaux d'alerte observables ?

✓ D'après vous, qu'est ce que vos "clients" souhaitent et n'obtiennent pas ou mal ?

✓ Sur quels points pourrions-nous être meilleurs ?

Questions à poser aux "clients" du système

Même question qu'au dessus avec en plus :

✓ Quels sont vos buts et en quoi le système y contribue-t-il ?

✓ Quels sont les liens entre le système et les autres systèmes ?

✓ Quelles sont les conséquences du [dys]fonctionnement actuel ?

✓ Que se passera-t-il si le système reste en l'état (statu quo, dégradation, faillite) ?

✓ Parmi les changements attendus, y en a-t-il de plus efficaces à court terme ?

Les outils d'investigation

Le Q.Q.O.Q.C.C. permet de faire un tour complet du système étudié. L'outil consiste à être en mesure de répondre aux six questions suivantes :

QUOI :	• Que fait-on ? De quoi s'agit-il ? • Quelles sont les objectifs, la mission du système ? • Quel est l'historique ?

QUI :	• Quels sont les acteurs qui sont référencés ? • Quelle est la structure interne, le type de relations ? • Quelle est la qualification, les compétences des personnes impliqués ? • Quels sont les différents rôles et qui les occupe ?
OU :	• Dans quels lieux (établissement, service) ? • A quel niveau de la structure (délégation, décentralisation) ?
QUAND :	• A quel moment ? • Phénomènes routiniers ou exceptionnels ? • En réaction à quel évènement ? • Sous quel délai ?
COMMENT :	• Quelles sont les procédures ? • Quels équipements particuliers ? • Quels documents sont utilisés et de quelle manière ?
COMBIEN :	• Voir le domaine de l'analyse quantitative.

Il s'agit d'un outil analogue avec le Q.Q.O.Q.C.C. Toutefois, il convient davantage à une étude en milieu industriel. Les questions portent sur cinq domaines.

LE MILIEU	• Où : l'espace, l'implantation, les distances, les proximités, la propreté, le nettoyage, l'encombrement, les obstacles ou le bruit. • Quand : à quelle fréquence ou à quelle heure ?
LES MATIERES	• L'énergie. • Les fournitures, les consommables. • Les composants, les documents.
LA MAIN D'OEUVRE	• Les opérateurs. • Les ouvriers. • L'encadrement. • La sous-traitance.
LE MATERIEL	• Les machines et l'outillage. • Les équipements.
LA METHODE	• Le mode opératoire. • Les technologies. • Les outils.

Les outils d'analyse

Pour s'aider dans cette modélisation on peut avoir recours à deux documents qui peuvent être proposés en synthèse des entretiens pour validation. Ils fournissent des informations sur l'importance relative des tâches.

- ✓ La fiche descriptive de document qui permet de recueillir les informations sur chaque document émis ou reçu dans le domaine de l'étude (voir annexe).
- ✓ La fiche d'attribution qui récapitule les activités des intervenants. Elle oblige l'intéressé à décrire ses travaux et à les quantifier (voir annexe).

On peut également utiliser un tableau de répartition des tâches. Il fournit une vue d'ensemble des activités par secteur. Il met en évidence des erreurs dans la répartition des tâches ou de juger du degré de polyvalence et de spécialisation du personnel. Enfin, il renseigne sur le niveau d'émiettement des tâches et l'intérêt probable du personnel.

Dans l'exemple ci-dessous, est détaillé un tableau de répartition des tâches pour le service des ventes d'un concessionnaire automobile.

Fonction **Activités**	**H.**	**Ingénieur commercial**	**H.**	**Agent Technico-commercial**	**H.**	**Secrétaire**	**H.**
Vente		Prospection tél. Visite client Réception client Tél. Client Courrier	4 15 4 2 3	Prospection tél. Visite client Réception client Tél. Client Courrier	6 2 10 6 5	 Accueil client Tél. client Frappe courrier	 2 3 12
56 %	74	57 %	28	68 %	29	42 %	17
Administration des ventes		Courrier Traitement litiges Statistiques	2 2 1	Courrier Statistiques Gestion	3 1 1	Courrier Gestion Classement	8 3 3
18 %	24	10 %	5	12 %	5	35 %	14
Organisation		Réunion Gestion planning Action commercial	3 3 5	Réunion Gestion planning Action commercial	3 2 2	 Réunion Tenue planning	 2 3
18 %	23	23 %	11	16 %	7	13 %	5
Relations avec les autres services		Service technique Service administratif	3 2	Service technique Service administratif	1 1	Service technique Service administratif	2 1
8 %	11	10 %	5	4 %	2	10 %	3
100 %	**132**	**100 %**	**49**	**100 %**	**43**	**100 %**	**39**

La mesure des temps

On vient de le voir dans l'exemple précédent, il est important de mesurer les temps. Cela permet de comparer deux équipements ou de définir des procédures de travail plus rapide. La mesure du temps peut par la suite vous permettre de prévoir un effectif, mieux répartir une charge de travail ou établir un planning.

Comment mesurer ces temps ? Quelle que soit la méthode employée, il ne faut pas oublier d'évaluer les temps de repos (que les utilisateurs ont des difficultés à aborder) ainsi que les temps consacrés aux imprévus. L'estimation des temps est à manipuler avec prudence car elle est souvent ressentie comme lourde de conséquences notamment pour le Personnel.

L'auto-analyse continue : l'opérateur enregistre lui-même les heures de début et de fin de chacune de ses activités. L'avantage de la méthode est sa facilité. En revanche, il y a une certaine difficulté pour une personne à enregistrer son propre travail. De la même façon, la sensation de contrôle et l'appréhension de l'utilisation de cette analyse sont fortes. Il est donc recommandé de préparer psychologiquement les personnes, voire les associer à l'intérêt de ce travail et le limiter dans le temps à quelques jours maximum. Enfin, le dépouillement de cette masse d'information sera longue et fastidieuse. Il est préférable dans ce cas de préparer des imprimés faciles à remplir.

L'analyse continue par observateur : c'est une personne extérieure qui observe et enregistre l'activité. Si la méthode est très fiable, elle a l'inconvénient de modifier l'attitude des personnes observées. Elle oblige à dédier un observateur pour étudier les événements.

L'auto-analyse semi-continue : les différentes activités sont codifiées et pré-imprimés sur un tableau. Il ne reste plus qu'à cocher les cases.

	Activités			
Horaires	**Courrier**	**Visite client**	**Réunion**	**Statistiques**
8h00 - 8h30	x			
8h30 - 9h00			x	
Etc…				

Une variante du même tableau permet à l'utilisateur d'indiquer en fin de période les durées respectives des activités qu'il a eues. Cependant, la fiabilité des informations dépend de l'interprétation de l'utilisateur et est en fonction du contexte psychosocial et de l'écart de temps qui s'est écoulé entre l'activité réelle et le remplissage du tableau.

Ce tableau a l'avantage d'un dépouillement et d'une comptabilisation facilités car dépourvu d'interprétation personnelle. Elle demande cependant une étude pour la codification des activités et ne convient pas aux tâches courtes (moins d'1/2 heure). On tiendra également compte de la marge d'erreur due à l'intervalle de temps : une tâche d'1h15 ou d'1h35 sera codifié sur 1h30.

L'observation en instantanée : c'est une observation qui s'effectue de manière discontinue à des moments pris au hasard. Ainsi, toutes les *n* minutes, on peut prendre la "photo" d'une situation. Plus les observations sont nombreuses, plus les résultats seront fiables.
L'avantage de cette méthode est sa simplicité de mise en œuvre car elle fournit des résultats assez rapidement. Il faut cependant trouver un intervalle de temps au hasard pour éviter de fausser les résultats en se calquant sur la fréquence naturelle d'un phénomène. On notera que cette méthode convient surtout aux phénomènes binaires ou simples à observer.

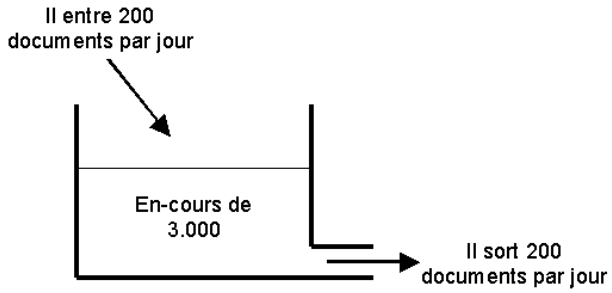
La mesure des délais

Un délai est une période de temps qui sépare deux étapes. Le délai inclut le temps d'exécution de certaines opérations et le temps d'attente. Ce dernier est souvent plus long que le temps de traitement : il est aussi plus difficile à mesurer.
Par exemple, le temps de réalisation d'une carte d'identité est de l'ordre de quelques minutes alors que le délai est de plusieurs jours.

Comment mesurer efficacement un délai ? Il existe plusieurs méthodes.

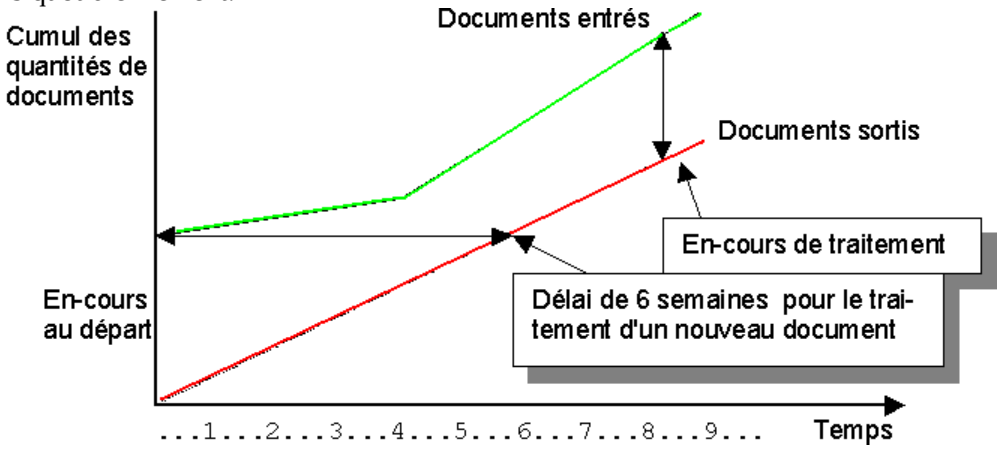
Le datage : les documents sont datés à l'entrée et à la sortie, il suffit alors d'observer un certain nombre de ces documents et d'en calculer le délai par différence. Cette analyse permet de faire ressortir plusieurs catégories de délai selon les événements traités.
Par exemple dans une banque, on aura trois types de délai pour un dossier complet (3 jours), pour un dossier incomplet (10 jours) et pour un dossier refusé (1 jour).
L'avantage de cette méthode est sa fiabilité en revanche le calcul des délais est fastidieux (sauf si l'on dispose d'un outil) et il y a nécessité d'effectuer un grand nombre d'observations et d'analyser au besoin les types de documents.

Le réservoir :



Il faudra 3.000 / 200 soit 15 jours pour qu'un document entré soit traité et sorti du système. L'avantage de cette méthode est de fournir des éléments très rapidement, en revanche elle nécessite de connaître les débits en amont et en aval ainsi que l'en-cours. Cette méthode ne fournit qu'une moyenne et est surtout adaptée pour des débits qui ne changent jamais.

Le diagramme cumulatif : on mesure le nombre de documents reçus et le nombre de documents transmis quotidiennement.



Cette méthode a l'avantage de mettre en évidence les fluctuations de débit et de délai. En revanche, elle nécessite une observation plutôt longue.

L'analyse quantitative

L'analyse quantitative vise à répondre à la question "COMBIEN ?" et ainsi compléter l'analyse qualitative du travail. Elle porte sur des volumes, des coûts, des temps ou des délais. Elle fait appel à deux techniques principales : l'observation et les estimations.

Elle doit être calibrée selon le besoin de l'étude car elle engendre elle aussi des coûts et des délais. Elle navigue entre deux écueils : ne rien vouloir mesurer et tout mettre en équation. Cette partie est difficile à réaliser en milieu administratif pour plusieurs raisons :

- ✓ Les données ne sont pas connues ou difficiles d'accès.
- ✓ Les personnes redoutent de mettre à jour la réalité.
- ✓ Il n'y a pas de recul sur l'exercice de l'activité (avec des réponses comme "ça dépend…").
- ✓ Peur de se tromper dans les estimations.

L'intérêt de l'analyse quantitative est de fournir un guide de réflexion pour mieux comprendre dans un premier temps l'organisation du travail et prendre des décisions pour l'optimiser en matière de :

- ✓ Temps de réponse.
- ✓ Volumes de documents à classer.
- ✓ Temps de recherche.
- ✓ Pourcentage d'occupation d'un équipement.
- ✓ Coût d'une erreur.

Pour la réalisation de cette partie, nous devrons faire appel à différentes techniques : le dénombrement, la règle de trois, les statistiques ou les probabilités. Ces résultats pourront être illustrés à travers des représentations graphiques de type histogramme ou camembert.

Diagnostic et orientations

Cette phase permet de préparer les critères qui permettront à leur tour d'évaluer les nouvelles solutions. Ainsi vous avez d'une part, l'étude de l'existant qui fait ressortir les dysfonctionnements (redondances d'information, ressaisies inutiles, documents inutilisés, etc). Et d'autre part, vous avez les commanditaires de l'étude qui formulent des besoins nouveaux, des contraintes ou des orientations particulières (budget limité, délai maximum à respecter, etc).

Pour établir un bon diagnostic, on veillera à s'appuyer sur des faits précis ou à citer les sources de l'information de départ. Attention car il ne s'agit pas de porter un jugement de valeur personnel sur la situation mais bien de fournir un nouvel éclairage au client.
Ainsi si le diagnostic se présente comme un inventaire de critiques, on gagnera à en faire une synthèse qui regroupera les différents points en catégorie.

Exemple :

- ✓ *manque de fiabilité de l'information saisie :*
  - *les fichiers du stock ne sont pas mis àjour en temps réel*
  - *la liste des clients douteux ne tient pas compte des avoirs accordés*
- ✓ *redondance de l'information :*
  - *il existe 3 fichiers clients dans la société*
  - *les bons de commande sont enregistrés une seconde fois à la facturation*
- ✓ *etc.*

Le tableau ci-dessous récapitule les principaux dysfonctionnements qui peuvent vous servir d'argumentaire dans votre diagnostic d'étude préalable.

Insuffisance	Des fonctions ne sont pas réalisées. Il y a un manque d'information ou d'outils de pilotage.
Qualité douteuse	Les informations ne sont pas cohérentes. Le temps de rafraîchissement des informations est trop long.
Saturation	Le temps de réponse est trop long. Les fichiers sont trop volumineux. Il n'y a pas de procédure d'archivage.
Sécurité	Il n'y a aucune sauvegarde du système. Les protections d'accès sont inexistantes ou insuffisantes.
Coût élevé	L'utilisation d'un système (logiciel ou matériel) ne justifie pas son existence. Les coûts de maintenance sont trop élevés.
Risque	La maintenance n'est plus assurée (ou impossible). La dépendance est importante vis à vis d'un founisseur. Les techniques sont mal connues de l'entreprise.

Recherche de solutions

Dans un premier temps, il s'agit d'étudier des solutions en termes de scénarios d'organisation. On se limitera à **trois** solutions au maximum. Celles-ci doivent être décrites selon un plan organisationnel avec la liste des matériels nécessaires et des coûts de fonctionnement à engager.

Analyse des coûts informatiques

Pour évaluer les coûts d'une solution informatique, il faut prendre en compte les différentes catégories de frais :

- ✓ Personnel
- ✓ Matériel
- ✓ Logiciel
- ✓ Maintenance
- ✓ Télécommunication
- ✓ Locaux spécifiques
- ✓ Equipements spécifiques
- ✓ Prestations
- ✓ Fournitures

Attention, vous devez penser à isoler :
Les coûts d'investissement qui ont lieu une fois (et sont éventuellement répartis sous forme d'investissement).
Les coûts de fonctionnement qui sont liés au cycle de l'activité.

Avantages et inconvénients d'un scénario

Dans un second temps, il s'agit d'apprécier les solutions proposées en termes d'avantages et d'inconvénients dans le but final de les comparer entre elles. Les critères peuvent être :

- ✓ Fonctionnels, en terme de service rendus, de possibilité d'évolution stratégique.
- ✓ Qualitatif, en terme de rapidité, de fiabilité ou de sécurité.
- ✓ Economique, en terme de gains ou d'économies supplémentaires possibles.

Faisabilité et impact organisationnel

Vous devez penser à la faisabilité de votre solution, notamment à sa rapidité de mise en œuvre, au facteur de nouveauté qu'elle va susciter dans l'entreprise et aux risques qu'elle peut engendrer (fournisseurs jeunes, techniques non stabilisées…).

Votre scénario risque d'impacter également l'organisation du travail. Les conséquences en termes de qualification et d'emploi sont à établir. Par exemple : une simple opératrice de saisie ne pourra peut être pas remplir une tâche où une décision est à prendre.

Délai

Enfin, n'oubliez pas les délais que va engendrer votre solution. Un mauvais calibrage peut remettre en cause tout votre planning. Il faudra donc se faire préciser les différentes date en ce qui concerne les livraisons de matériel, la réalisation de logiciels spécifiques, le paramétrage des progiciels, le recrutement ou la formation. Certaines obligations légales sont également à prévoir.

Préconisations

Cette phase a pour but de recommander une solution en argumentant les raisons de ce choix. Parfois, les solutions sont progressives, on peut alors commencer par la solution minimale et l'incrémenter par la suite.
Exemple : une société a plusieurs succursales. La solution minimale est d'informatiser chaque bureau puis de les fédérer par la suite au sein d'un réseau de type Intranet.

Toutes les tâches à effectuer jusqu'à la fin du projet sont à inventoriées avec :

- ✓ La durée de chacune.
- ✓ Le planning prévisionnel.
- ✓ La responsabilité des interlocuteurs.

Au cours de cette phase, on prépare un rapport de fin d'étude préalable dont la finalité est de permettre la prise de décision. On l'appellera également rapport de synthèse.

COMPRENDRE PAR L'EXEMPLE

Vous êtes toujours là ? Nous avons vu dans les chapitres précédents beaucoup de termes et de vocabulaire, tâchons d'y voir un peu plus clair au travers d'exemples. Ceux-ci sont fictif et relativement simple mais ils ont le mérite de vous faire aborder concrètement l'approche de l'étude préalable.

Analyse des acteurs et des flux

La mairie de P…, petite commune de 7.000 habitants, est organisée en différents services selon ses activités. Madame B. s'occupe du service social et plus précisément de l'organisation des séjours de vacances pour les enfants de la commune. Depuis deux ans, Mme B. se plaint de ne plus pouvoir remplir correctement cette tâche. L'étude porte sur son travail et nous allons dans un premier temps analyser les différents flux.

Compte-rendu d'entretien

Après plusieurs entretiens avec Mme B. sur son travail quotidien, vous avez pu faire ressortir les faits exposés ci-dessous.
Plusieurs séjours sont organisés dans l'année à l'occasion des congés scolaires. La ville est trop petite pour disposer d'un budget conséquent et la population des enfants se répartit sur trop d'âges différents, ce qui constitue des groupes à effectif faible. C'est pourquoi la Mairie n'organise pas elle-même ses séjours mais sous-traite cette activité à des associations spécialisées gérant des colonies de vacances (UFOVAL, Vacances pour tous…).
Mme B. est amené à s'informer auprès de ces partenaires des possibilités de séjours auxquels la commune peut prétendre. Pour cela, elle se procure de la documentation. Lorsqu'elle a en main la totalité des brochures, Mme B. les examine en vue de faire une sélection limitée. Elle retient quelques séjours en fonction de différents critères comme l'âge, le lieu de destination (mer, montagne, campagne), le style (activité dominante ou multiples), le prix. Mme B. tient à faire une offre variée sans pour autant mettre les familles dans l'embarras du choix.

Pour chaque séjour retenu, Mme B. doit réserver un certain nombre de places. Elle doit également verser un acompte provisionnel. Mme B. remplit un formulaire destiné à la comptabilité pour le déclenchement des versements de ces acomptes. Elle ouvre enfin un dossier pour chaque séjour où est conservé toutes les pièces (brochure descriptive, copie du formulaire, etc).

Mme B. élabore ensuite un mini-catalogue à partir des brochures reçues. Elle y ajoute également les conditions de réservation et d'accessibilité. Ce catalogue est à la disposition des familles qui peuvent le chercher directement à la Mairie ou se le faire envoyer sur demande téléphonique ou écrite. Mme B. passe également un avis d'information dans le journal communal.

La réservation se fait pendant une période limitée. Au delà, Mme B. annule auprès des organisateurs les places qu'elle avait réservées en trop. Elle confirme cela par courrier en demandant le remboursement éventuel des acomptes trop perçues. Le chèque qu'elle reçoit en retour est transmis à la comptabilité pour encaissement.

Lorsqu'une famille veut réserver, elle fournit à Mme B. plusieurs pièces :

- ✓ Une attestation de domicile.
- ✓ Un avis d'imposition de l'année précédente.
- ✓ Une copie du livret de famille.
- ✓ Le carnet de santé de chaque enfant.
- ✓ Le chèque de réservation.

Les parents doivent en plus remplir un questionnaire relatif à l'état civil de l'enfant et à sa situation médicale (vaccins, allergie, traitement en cours…). Le prix à facturer dépend du quotient familial (revenu / nombre de parts fiscales), Mme B. effectue ce calcul.
Un mois avant le début du séjour, Mme B. envoie aux parents un courrier pour le règlement du solde. Les paiements sont transmis à la comptabilité avec un formulaire récapitulatif des places réservés et des acomptes versés par les parents.

Lorsque la période de réservation est terminée, Mme B. transmet aux organisateurs la liste des réservations avec une copie des questionnaires. La liste du trousseau est transmise directement par les organismes à chaque parent, toutefois des ratés peuvent survenir et Mme B. se charge alors de débloquer la situation par téléphone.

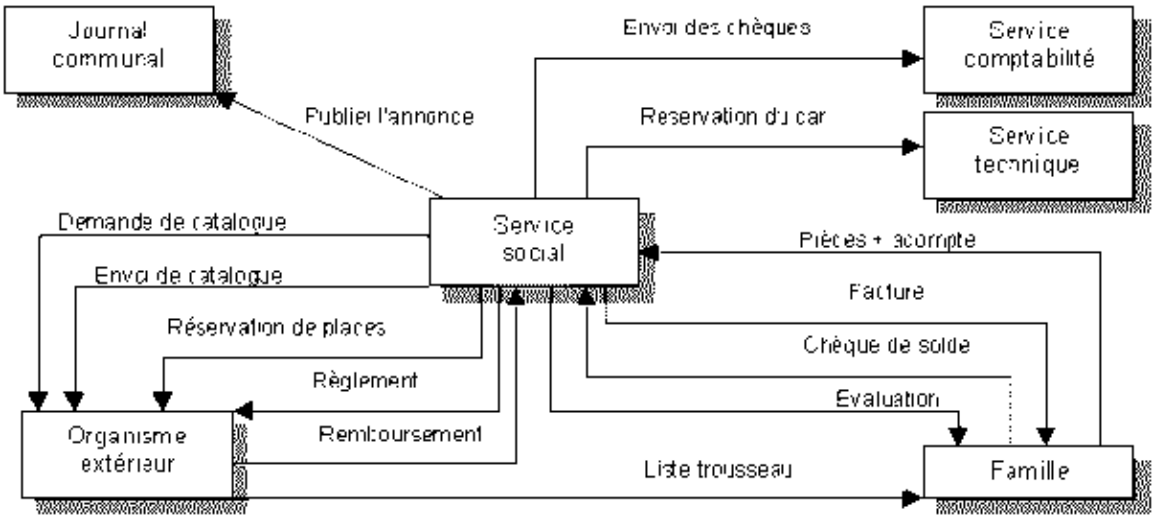
Pour certains séjours, le lieu de convocation est fixé assez loin de la ville. Mme B. tient donc à proposer un service de transport. Elle contacte le service technique pour planifier les réservations du car municipal et de son chauffeur. Elle adresse ensuite un courrier aux parents concernés avec le lieu et l'heure du rendez-vous. Si l'enfant à moins de 12 ans, il doit être sous la surveillance d'un adulte. Les parents doivent retourner le coupon-réponse s'ils désirent profiter de ce service gratuit.
Quinze jours après les vacances, Mme B. adresse aux familles une fiche d'évaluation. Celle-ci lui permet de mieux cerner les goûts des enfants et d'apprécier la qualité des séjours. Après examen, ces fiches sont classées dans les dossiers des séjours concernés.

Analyse de l'entretien

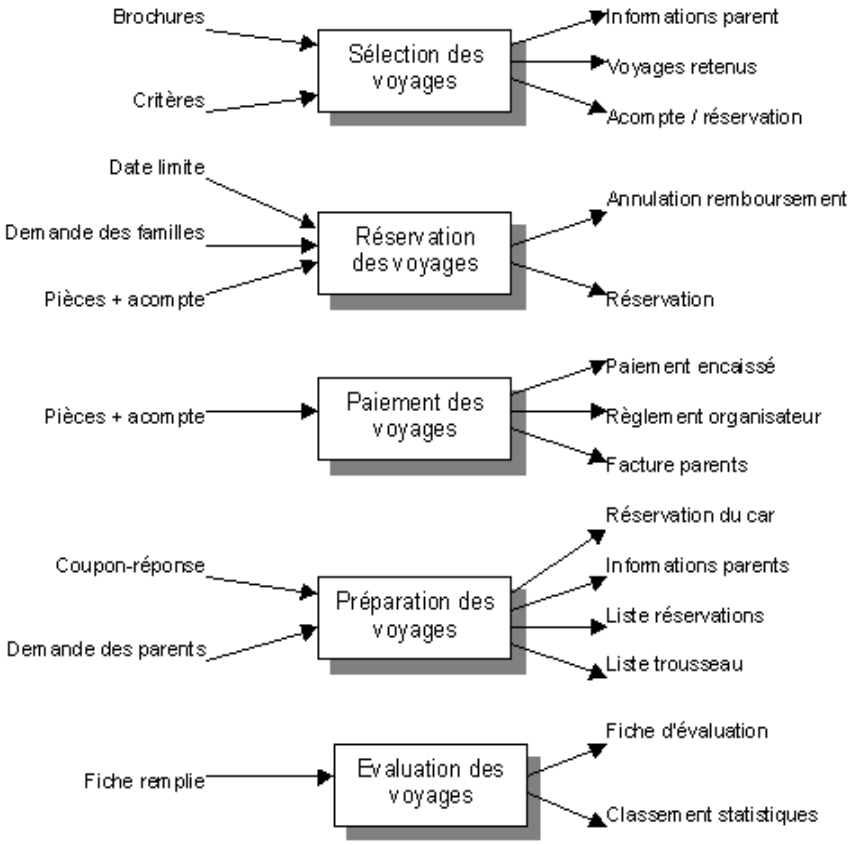
Comme vous pouvez le constater, même pour un service assez simple (une seule personne), la liste des tâches effectuée devient assez vite importante. De plus, l'utilisateur va vous "noyer" sous un flot de détails souvent inutile (voir inutilisable). Il faut donc apprendre à trier objectivement et rester à un niveau de détail raisonnable.

Les acteurs

L'organigramme ci-dessous détaille non seulement les acteurs du système mais également les flux qui circulent entre eux.



Le diagramme des flux



On notera le découpage différent de celui des acteurs avec différents processus (modules établis par tâche). Chaque processus est découpé afin de détailler :

- ✓ Les composants qui interviennent avec les flux d'entrée à gauche.
- ✓ Les résultats produits avec les flux de sortie à droite.

PLAN TYPE D'UN RAPPORT D'ÉTUDE PRÉALABLE

Ce chapitre conclue en récapitulant les différents éléments qui doivent composer votre rapport d'étude préalable.

Mission

- ✓ Le contexte sommairement décrit : où, quand, cadre de l'étude.
- ✓ Le but de l'étude ou le pourquoi de cette étude.
- ✓ Eventuellement la démarche de l'étude ou la présentation du rapport.

Etat de l'existant

- ✓ Aller du général au particulier, ce qui permettra d'arrêter la présentation écrite à la maille de détail la plus pertinente et de reporter éventuellement les détails en annexe.
- ✓ Faire obligatoirement une représentation schématique (DCD[1] par exemple, schéma des processus ou MOT[2] Merise).
- ✓ Recenser des faits et des volumes.
- ✓ Garder un ton neutre (aucune critique, pas de sous-entendu).

Diagnostic

Il s'agit de répertorier dans ce chapitre, l'ensemble des dysfonctionnement que révèle l'étude. On y regroupe les critiques en catégories de problème type rencontrés, ce qui permet de dégager les idées forces des solutions en termes de qualité ou de coût.
Par exemple, votre étude porte sur des difficultés dans l'identification des articles. Les problèmes types peuvent être :

- ✓ Le service des expéditions ne trouve pas certains articles dans le fichier.
- ✓ Les recherches dans les dépôts sont souvent infructueuses.

Attention :
une liste de critiques négatives n'est pas constructive et risque d'assommer le lecteur (il les connaît déjà).

Etayer ce qu'on dit par des observations. Fournir la source des informations (ont-elles été vérifiées ou sont-elles seulement dîtes par les autres ?). Spécifier, expliciter ce que l'on dit, éviter le flou et l'abstrait. Par exemple, ne pas dire « mauvaise gestion » mais expliquer pourquoi la gestion est mauvaise.
Dégager les faiblesses mais aussi les forces du système. Se garder de préconiser des solutions, ce qui témoignerait d'un manque d'esprit de conception générale.

Objectif et contraintes

Rappeler ou détailler les objectifs. Par exemple : mettre en place un nouveau service, augmenter les plages horaires, etc.
Enumérer les contraintes. Par exemple : utiliser les moyens informatiques existants, dégager une enveloppe budgétaire, obligation d'arrêter la production pendant la durée de certaines opération, etc.
On peut profiter de ce chapitre pour récapituler :

- ✓ Les risques encourus par l'entreprise si on ne change rien.

[1] Diagramme de circulation de documents
[2] Modèle organisationnel des traitements

- ✓ Les bienfaits que l'entreprise trouverait à aller vers une amélioration.

Ce dernier est là pour préparer la motivation du lecteur pour le chapitre suivant.

Solutions

Cette partie est la plus représentative et la plus importante du rapport. Elle contient la plus-value générée par l'étude.
On doit y trouver au moins deux solutions globales qui se déclinent en :

- ✓ Principes généraux, idée-force, ligne directrice.
- ✓ Moyens techniques.
- ✓ Processus, procédures.
- ✓ Documents annexes (ceux qui ne rentrent pas dans le domaine).
- ✓ Circuits.
- ✓ Une représentation schématique.
- ✓ Les coûts.
- ✓ Avantages et inconvénients (incluant la faisabilité et l'impact).

Ne pas hésiter à faire preuve d'innovation dans les solutions techniques. Par exemple : utilisation des codes barres, recours au multimédia, etc.

Attention :
Si votre solution modifie l'effectif de l'entreprise ou de la structure en plus ou en moins, elle doit être sérieusement justifiée.

Baptiser les solutions de façon pertinente permettra à l'entreprise de mieux les appréhender et de se les approprier.
On peut conclure en faisant un tableau récapitulatif des solutions qui fera ressortir pour chaque solution :

- ✓ Son nom.
- ✓ Ses principes généraux.
- ✓ Ses avantages et ses inconvénients.
- ✓ Son coût.

Attention :
Pour les coûts, vous devez isoler ce qui est de nature investissement (achat de matériel) et de qui est de nature fonctionnement (location des lignes, maintenance, etc.).

Propositions

Ce chapitre permet à l'organisateur de donner son avis personnel en argumentant sa position sur une des solutions présentées au-dessus. On y trouve un plan d'action prévoyant :

- ✓ Les différentes étapes.
- ✓ La charge de travail.
- ✓ Les besoins en personnel (maître d'œuvre, groupe de travail…).
- ✓ Les délais.

Conclusion

Ce chapitre n’est pas obligatoire. Il permet d’adresser des souhaits, des remerciements et des appréciations d’ordre affectif ou commercial.

ANNEXE

FICHE DESCRIPTIVE DE DOCUMENT	
Référence ou numéro interne	Titre :
Format Support : listing - pré-imprimé - page Nombre d'exemplaires : Taille : Reliure spécifique :	Nombre de documents en annexe : Nom ou référence des documents annexés :
Emetteur : A quelle occasion :	Volumes : Fréquence ou périodicité : Durée de vie :
Rôle du document ou descriptif :	
Destinataires successifs :	Utilisation :
Classement Mode : Lieu : Format :	
Archivage Mode : Lieu : Format :	

FICHE D'ATTRIBUTION					
Nom :				Bureau : Poste ou téléphone : Fax :	
Direction :				Fonction :	
Activité	**Tâche**	**Fréquence Q, M, H**	**Nombre moyen**	**% temps travail**	**Observation**

Cyril Beaussier
Gestion de projet informatique

La conduite de projet

Version 1.0 - Février 2001

COPYRIGHT ET DROIT DE REPRODUCTION

Relecture et corrections : Evelyne Henry

Sommaire

Introduction

L'objectif de ce support n'est pas de faire de vous un chef de projet exceptionnel, mais de jeter un regard sur les différents aspects de ce métier et sa complexité grandissante et d'en connaître suffisamment pour gérer un projet dans de bonnes conditions.

Car beaucoup (beaucoup trop) de chefs de projet n'utilisent pas ou peu de méthode pour la conduite de projet souvent important sinon critique pour leur entreprise. Cela donne des aberrations avec des exemples comme SOCRATE à la SNCF ou VITAL pour la Sécurité Sociale qui laissent penser que l'informatique est une vaste usine à gaz.

L'application de méthode souvent simple et évidente vous permettront d'acquérir des réflexes en cas de problème. Il s'agit tout simplement de faire de vous un bon chef de projet et ce n'est déjà pas si mal…

La vie du projet

Un projet connaît, au cours de sa vie, un cycle caractérisé par les phases suivantes :

1. **Le lancement :** suite à l'appel d'offre, le moral est au beau fixe, c'est l'euphorie.
2. **La conception :** malgré quelques remous, l'équipe est renforcée dans sa conviction qu'elle peut bien faire. L'adhésion du commanditaire à la solution retenue justifie souvent cet état de grâce.
3. **L'étude détaillée :** avec sa colonie de grains de sable, elle entraîne des variations plus ou moins sensibles du moral des troupes, c'est l'inquiétude.
4. **La réalisation :** la panique s'empare de l'équipe face à l'ampleur du travail à effectuer dans le temps imparti.
5. **L'installation :** l'équipe essaye de justifier les premiers dysfonctionnements à l'utilisateur en recherchant des coupables.
6. **Les essais :** ils déboucheront certainement sur une chasse aux sorcières où on ne fera que punir des innocents.
7. **La mise en œuvre :** elle voit naître un climat de suspicion où chacun cherche à dégager sa responsabilité dans la dérive du projet initial. Elle ne fera que promouvoir ceux qui n'ont jamais participé au projet.

Les principaux griefs que l'on reproche souvent aux projets informatiques sont divisés en trois catégories.

Les utilisateurs

- Difficulté de dialoguer avec les informaticiens
- Les délais sont trop longs
- Il y a un manque de formation
- Il n'y a pas ou peu de documentation
- Inadéquation de la solution livrée avec le besoin exprimé
- Lancement sans procédure officielle
- Sous estimation de la charge de travail au démarrage
- Mauvaise qualité des produits livrés, temps de réponse trop longs, possibilités insuffisantes, manque de normalisation, etc.

La Direction

- Absence de plan informatique précis
- Augmentation des délais
- Augmentation des coûts
- Manque de lisibilité sur l'avancement du projet
- Manque d'évolutivité des systèmes proposés
- Mauvaise rentabilité de la production informatique
- Qualité douteuse
- Documentation insuffisante

Les informaticiens

- Produit non figé : ce qui est demandé n'est pas écrit dans le cahier des charges.
- Contraintes irréalistes en délai, charges ou coûts.
- Manque de moyens matériel et humain
- Difficultés de maintenance
- Absence d'outils dans un environnement technologique en évolution
- Manque de reconnaissance

Concepts

Les préoccupations du concepteur restent du domaine de la créativité. Le processus et les méthodes de conception s'organisent autour des réponses à la question : « comment faire ? ».

En revanche, pour conduire un projet, il convient de répondre à d'autres questions :
Comment gérer de façon optimale le processus de conception en termes de ressources humaines, matérielles et financières ?
Comment assurer le respect du délai et du coût ?
Comment assurer et contrôler la qualité des opérations de conception ?

N'oublions pas que la conduite de projet n'est pas une fin en soi mais bien un moyen pour se prémunir contre les débordements d'un processus de créativité sans fin. Il n'est pas rare en effet que l'informaticien se laisse déborder en ajoutant de multiples fonctionnalités supplémentaires qui lui semble être des prouesses techniques mais qui ne sont en fin de compte pas demandées par l'utilisateur.

Les méthodes de conduite de projet apportent essentiellement une réponse à la question :

« comment gérer le comment faire ? »

Caractéristiques

Le projet doit avoir les caractéristiques suivantes :

1. Un objectif. Il s'agit de se concentrer sur « ce qu'on doit livrer ».
2. Un propriétaire (ou un demandeur). C'est le maître d'ouvrage du projet. Il représente souvent l'utilisateur.
3. Un délai. Le projet doit avoir un calendrier.
4. Un budget. Il s'agit de connaître les ressources humaines et matérielles dont on dispose pour la réalisation.
5. Un chef. Il s'agit du maître d'œuvre du projet.

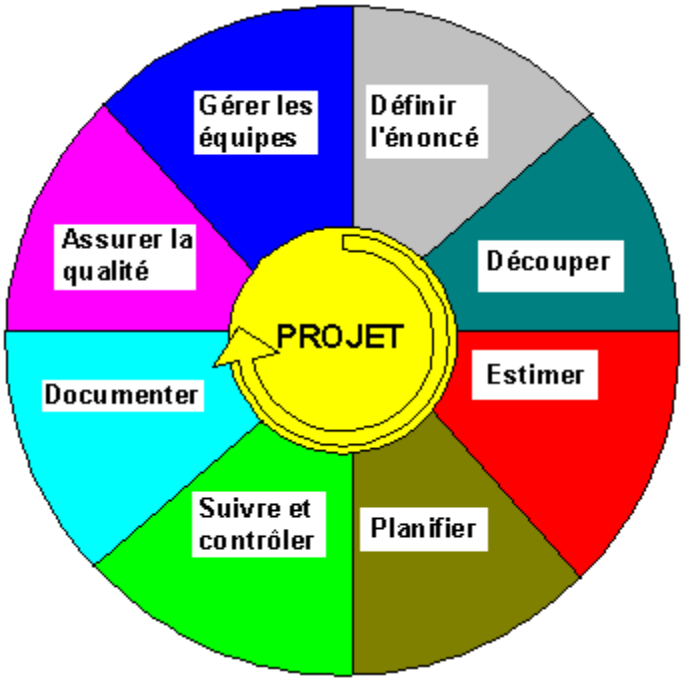
Nature d'un projet

Il existe quatre grandes catégories de projet. En déterminer la nature exacte permet déjà de révéler au chef de projet quelles compétences il aura à utiliser pour sa réalisation.

Production opérationnelle	De loin les projets les plus nombreux et souvent les plus anciens dans l'entreprise, ils portent sur la prise en charge de travaux administratifs comme la paye, la gestion client, la prise de commande, etc. Leur durée de vie est assez longue (entre 4 et 10 ans). Un nombre assez limité de partenaires est concerné et le problème comme la solution sont assez spécifiques.
Technique ou multimédia	Projets très spécifiques, ils prennent place des domaines fonctionnels bien cernés : enregistrement de commande à partir de carte à puce, borne d'information, etc. Leur durée de vie est assez courte (1 à 3 ans) mais font appel à un nombre limité d'acteurs très initiés. Le problème est spécifique mais les solutions sont restreintes.
Aide à la décision	Plus généraux, ils répondent aux attentes des cadres chargés de piloter le fonctionnement d'une unité (un service marketing ou commercial). La durée de vie est très longue (supérieure à 10 ans). Les problèmes sont complexes et les solutions diverses.
Stratégique	Assez rares, ils trouvent leur origine au sein de la Direction générale (logiciels financiers, PGI, etc.). La durée de vie est moyenne (environ 5 ans). Le problème est simple, souvent mal exprimé ou confus et la solution est complexe et souvent innovante.

On s'intéresse également à cerner le type de projet en regardant les règles de gestion qui vont servir à décrire le produit à fournir. On saura ainsi si ces règles ont un caractère éphémère, stable ou évolutif. La réflexion vise à se prononcer sur la durée de vie prévisible du logiciel et sur son évolution dans le temps. Ceci afin de réduire le degré d'incertitude qu'il pourra y avoir sur le cycle de maintenance.

Les 8 points clés du projet



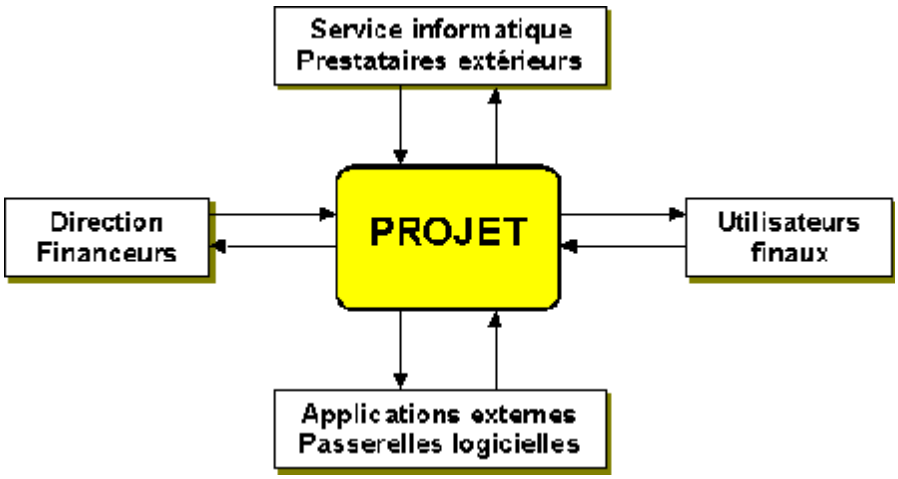
Définition de l'énoncé

C'est l'ensemble des questions à se poser pour mettre d'accord les acteurs sur la nature du projet. Il s'agit d'avoir une idée claire sur ce qui doit être livré.

Une bonne méthode consiste à utiliser la technique du QQOQC[1] (quoi, qui, où, quand et comment).

Quoi

Quel est le projet ? Il faut déterminer impérativement par écrit les limites du projet. Les frontières ainsi définies permettront de savoir ce qui fait ou ne fait pas partie du projet.



En principe, ces limites ont été établies dans l'étude préalable. Néanmoins, il est bon de faire une piqûre de rappel au client pour restituer le projet dans son contexte.

Rappel : la progression du projet se fait par différentes étapes :

[1] Voir le manuel de l'étude préalable disponible sur ce même site.

Schéma directeur	Son champ d'investigation est l'entreprise tout entière ou l'une de ses activités essentielle. Il s'attache à identifier les domaines, il recense et normalise les données stratégiques. Il met en évidence les interfaces entre les domaines et élabore des scénarios. Enfin, il définit les priorités de développement et attribue les masses budgétaires.
⇓	
Etude préalable	Elle s'applique à un domaine défini par le schéma directeur. Son objectif est de définir l'opportunité et la rentabilité d'un projet. Elle débouche sur un dossier de choix.
⇓	
Etude détaillée	C'est l'analyse détaillée de la solution retenue pour le projet, du point de vue des utilisateurs. Son but est de définir les spécifications externes visant à obtenir l'accord des utilisateurs sur les options décrites. Enfin, elle précise le planning de réalisation.
⇓	
Réalisation	Elle comprend deux étapes : l'étude technique qui définit les spécifications internes portant sur les contraintes matériel et la production du logiciel en lui-même.
⇓	
Mise en œuvre	C'est la mise en exploitation du logiciel après sa réception par les utilisateurs.
⇓	
Maintenance	Elle peut être : • Curative (correction d'anomalies ou de bogues) • Perfective (amélioration du logiciel) • Adaptive (changement d'environnement matériel) • Evolutive (modification des règles de gestion)

L'étude détaillée vous permettra ensuite de :

- déterminer la forme du logiciel (sur mesure ou acheté à l'extérieur),
- d'identifier l'environnement (matériel, système et langage à utiliser),
- de choisir un mode de fabrication (en interne ou en sous-traitance).

La définition de l'énoncé vous permettra en conclusion de :

- choisir la forme du logiciel (sur mesure ou du progiciel),
- d'identifier l'environnement (les matériels et les langages),
- de choisir un mode de fabrication (en interne ou en sous-traitance).

Qui

La conduite de projet repose avant tout sur des facteurs humains. Quelles que soient les méthodes et les techniques employées, leur efficacité dépend de la performance des personnes en présence, tant du côté utilisateur qu'informaticien.

Le choix d'un bon dispositif humain, sa gestion au fur et à mesure des événements du projet constituent le point clé d'un bon processus de conduite de projet. Ce dispositif doit ainsi maintenir un fort degré d'adhésion des acteurs.

Si l'on considère qu'un projet a pour but de livrer « un produit apte à satisfaire les besoins des utilisateurs », on distingue alors deux grandes classes d'acteurs.

	Le client ou maître d'ouvrage	Le fournisseur ou maître d'œuvre
ROLE	C'est le propriétaire du projet. Son rôle est de définir les objectifs du projet et les besoins fonctionnels. Il fixe le cadre des travaux confiés et s'assure du financement. Il recette les prestations fournies et organise les formations.	Son rôle est d'identifier et de planifier les tâches. Il détermine les moyens et réalise les travaux. Il fournit les logiciels et rend compte de l'avancement au client.
ACTEUR	• Les décideurs qui choisissent, commandent ce qui est bon pour eux. • Le pilote ou chef de projet qui représente les intérêts du commanditaire auprès du fournisseur. • Les usages qui sont les futurs consommateurs du produit	• Le chef de projet qui est responsable vis à vis du client mais aussi animateur interne de l'équipe projet. • Les concepteurs. • Les réalisateurs.

Où

Il s'agit de préciser les lieux géographiques où vont se dérouler chaque phase du cycle de développement du logiciel.

Un certain nombre de question peuvent alors se poser : Quels sont les besoins en locaux ? Faut-il de nouveaux aménagements ? Veut-on des lignes de communication supplémentaires ? etc…

Quand

Deux cycles caractérisent les projets :

Cycle de vie	Cycle de décision (pour le client)
Il se définit comme une suite de tâches ordonnées et dépendantes conduisant à la mise à disposition d'un logiciel de qualité auprès d'un client. La nature du découpage, la maille adoptée tout comme les libellés varient d'une société à une autre.	Il se définit comme la liste ordonnée des décisions à prendre au fur et à mesure de l'avancement de la fabrication du produit. Chaque phase du cycle de vie conduit les acteurs en présence à s'accorder sur des décisions. Ces décisions sont prises à des moments clés : les **jalons**.

Il est également important d'identifier dès le début du projet les contraintes de date qu'il faudra impérativement respecter. Il ne faut pas non plus oublier les contraintes liées à la disponibilité des ressources.

Par exemple : commencer la réalisation sans avoir la date de livraison précise du serveur qui hébergera votre base de données.

Note :
Il faudra le plus vite possible élaboré un macro-planning du projet. Il montrera les dates clés, les jalons et les principales contraintes.

Comment

La préoccupation de l'utilisateur d'une méthode est de disposer d'un processus reproductible, structuré, puis manageable, lui permettant d'obtenir les résultats attendus en fonction du projet qu'il souhaite entreprendre.

La méthode retenue doit permettre une prise en charge des différentes activités du projet comme le découpage, l'estimation, la planification, le suivi, le contrôle, la documentation…

Une méthode apporte une aide à trois niveaux :

- Des éléments de raisonnement
- Des outils
- Un formalisme

Le tout avec un vocabulaire commun, une acquisition d'expériences sur les mêmes bases et des comparaisons possibles entre projets et entre entreprises.

Il existe une bonne demi-douzaine de méthode parmi les plus connues : Merise et UML[2]. La maîtrise du vocabulaire est essentielle. Quelle que soit la méthode de conduite de projet retenue, elle passe par un minimum de vocabulaire commun.. On ne répétera jamais assez que les seules véritables causes d'échec d'un projet résident dans l'incapacité à communiquer et à dialoguer entre les partenaires. L'un des freins provient de la confusion des termes associés aux travaux de conception et à ceux de la conduite du projet.

Découpage

Opération consistant à réduire la complexité d'un problème en le décomposant en morceaux de complexité moindre tout en gardant une vue globale.
Il s'agit de délimiter le projet en suite de phases, sous-phases jalonnées de points de contrôle.

Pourquoi découper ?

Un projet est constitué d'une série de tâches dont la complexité croît avec son ampleur. Il s'agit donc de faire face à cette complexité. Ces tâches peuvent se dérouler en parallèle ou en séquence. Certaines d'entre elles sont critiques.

Cette multitude de travaux à exécuter dans un laps de temps de plusieurs semaines ou mois rend impossible la maîtrise par une seule personne de l'avancement du projet si cette dernière ne dispose pas d'un outil de planification des opérations à mener.

Un découpage précis du chemin à parcourir et des différentes parties du produit à construire permet une bonne identification et optimisation des ressources, des délais, des charges et des coûts. L'établissement de cette planification et du suivi contribue très sensiblement à améliorer les processus d'estimation. Le chef de projet dispose ainsi de la liste des travaux à exécuter et de leur nature sans laquelle l'estimation des ressources, la planification et le suivi sont impossible.

Il faut dans un second temps progresser vers l'industrialisation. Cette démarche s'appuie sur une analogie avec la fabrication d'un produit usiné. La décomposition d'un produit fini en une série de constituants élémentaires (à prix de revient connu) fournit la liste des tâches à exécuter lors de chacune des phases de l'élaboration avec un processus de production maîtrisé et reproductible. Il devient alors possible de réaliser des économies par la réutilisation de modules déjà fabriqués.

Il faut diminuer les risques de dérive. Une dérive arrive rarement d'un bloc (sauf événement exceptionnel). Elle naît et grossit petit à petit au cours de l'enchaînement des diverses tâches du projet. Il est donc essentiel de contrôler celle-ci au plus tôt, c'est-à-dire tâche par tâche.

[2] Voir le manuel sur les méthodes d'évaluation disponible sur ce même site.

Quelles sont les difficultés ?

Ne rien oublier Etablir la liste complète des tâches d'un projet paraît souvent impossible. L'un des moyens de ne pas oublier de tâches est de disposer d'un cadre méthodologique qui fournit une liste exhaustive des tâches d'un projet.

Choisir la bonne "maille" Faire un bon découpage en tâches homogènes, pour rendre le projet facilement maîtrisable demande un professionnalisme certain. Les tâches ne doivent être ni trop longues, ni trop courtes, ni trop nombreuses. Selon l'importance du projet, on pourra avoir plusieurs niveaux de découpage.

Identifier la notion de tâche C'est être capable de fixer des bornes, c'est-à-dire de délimiter de façon précise chacune des tâches en terme de durée, de début, de fin et de résultat produit. C'est aussi savoir comment l'on va suivre, valider ou contrôler la qualité de la tâche une fois réalisée.

Intégrer l'aspect documentaire Le reflet d'avancement du projet doit s'appuyer sur une série de documents faisant l'objet de validations successives. Cela suppose d'associer à chaque tâche et à chaque fin de phase, un ou plusieurs documents puis de déterminer pour chaque document la liste des tâches conduisant à sa réalisation.

Prendre en compte l'encadrement Le découpage du projet en phases et en tâches concerne la fabrication du produit. A cela, il faut y ajouter les tâches spécifiques liées à la conduite du projet. Celles-ci comprennent un ensemble d'activités comme l'établissement et la mise à jour du planning, le suivi de travaux, le contrôle de la qualité ou l'analyse des incidents.

Estimation

C'est l'ensemble des techniques plus ou moins sophistiquées permettant au chef de projet d'évaluer à l'avance les charges et les coûts d'une ou plusieurs phases et d'ajuster au fur et à mesure que l'on avance dans la phase.

L'estimation du coût des projets informatiques est un **art périlleux**. Ils coûtent toujours plus chers que prévu et durent plus longtemps qu'on ne l'imaginait à leur lancement. Cerner les temps de développement, évaluer l'effort à fournir et la répartition de cet effort dans le temps, tout cela relève bien souvent de la voyance.

Pourquoi estimer ?

Il s'agit avant tout d'estimer la durée du projet. La préoccupation principale des équipes reste la faisabilité du travail demandé en terme de temps. Une méthode d'estimation aide à prévoir la charge par phase d'une façon assez fine.

Il faut disposer à l'avance de suffisamment d'informations sur la charge éventuelle afin de réagir en terme de ressources humaines pour tenir les délais imposés par le client.

Il faut améliorer la productivité en établissant une liste d'éléments comparatifs destinés à mesurer les délais, les charges et les coûts.

Les principaux obstacles

La difficulté majeure reste de loin la **non-identification** de la nature du produit à développer. Lorsque le cahier des charges existe, les lacunes portent le plus souvent sur l'aspect qualitatif des fonctionnalités. L'absence de précision sur ces critères comme la maintenabilité, l'adaptabilité ou d'autres caractéristiques qualitatives, nuit à une estimation précise des charges de réalisation.

Les conséquences d'un mauvais découpage du projet en une série de modules plus ou moins homogènes. Certains facteurs sont susceptibles d'affecter l'estimation comme le surnombre de modules, leur complexité ou la mauvaise appréciation de leurs interactions.

Le véritable obstacle à la mise en place des méthodes d'estimation demeure la résistance au changement. L'estimation requiert un effort et une discipline quotidienne et fait appel à un double travail : prévision et contrôle. L'apparition systématique d'écart entre la prévision et la réalisation suscite au sein des équipes de développement un certain découragement. D'autre part, ces méthodes sont souvent ressenties comme un moyen plus ou moins avoué de contrôler la productivité, ce qui amène certaines réticences dans l'application.

Enfin, le chef de projet a souvent tendance à la sous-estimation. Celle-ci est fréquente et trouve ses causes dans le désir de plaire au client, un excès d'optimisme ou une expérience limitée du domaine.

Ce que recouvre une estimation

Bien sûr, l'estimation est d'abord une expression des coûts. Ceux-ci recouvrent la somme des dépenses associés à la fabrication du logiciel. On peut les éclater en trois grandes catégories :

Personnel	Logiciel	Matériel
Exprimé le plus souvent en mois/homme et traduit ensuite en K€ grâce à un coefficient lié au profil de la personne.	Système d'exploitation, nombre de licences par site, etc.	Ordinateur et imprimante nécessaires mais aussi câblage et équipements de réseau (pont, routeur…).

A cela, il faudra encore ajouter (liste non exhaustive) :

- Frais de formation
- Achat des outils de développement : AGL, méthodes…
- Télécommunications
- Frais de déplacement
- Infrastructure : aménagement ou location de locaux
- Fournitures

Vous trouverez en annexe 1 un tableau récapitulatif de ces coûts qui peut vous servir à une bonne estimation de chaque phase de votre projet.

L'objectif suivant est de déterminer des temps en terme de charge de travail. Ceux-ci sont exprimés en mois/homme ou jour/homme, ils permettent d'établir des durées et donc de fournir des délais. Nous avons ainsi le schéma suivant :

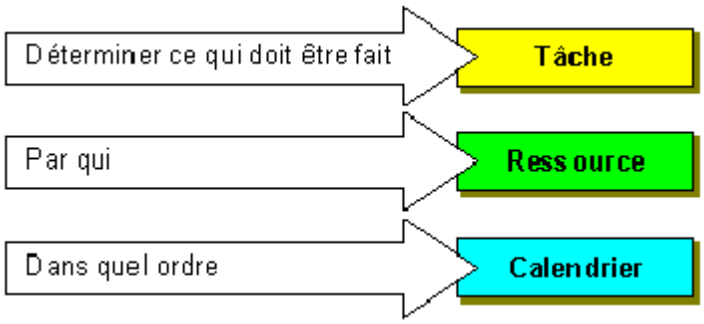
Pour calculer les délais, il faut connaître le nombre de jours effectivement productifs. En effet, il faut penser que ces délais s'expriment en jours ouvrables et que les jours fériés, les congés et les absences justifiées (maladie) ne font que les rallonger. Il faut également tenir compte des formations et des réunions de travail. On peut dire que sur 10 jours de travail effectif, le coefficient de productivité est égal à 8 jours.

Planification

Technique consistant à ordonnancer dans le temps les ressources nécessaires à l'exécution d'une phase.

Objectif

Il s'agit d'organiser un processus de fabrication en se dotant d'un outil pour le suivi. La planification est un processus continu d'aide à la décision tourné vers le futur. La planification va s'appuyer sur deux éléments : le découpage (par tâche) et l'estimation des charges (ressource). Le tout sera projeté sur un calendrier.



Gardons à l'esprit qu'une planification n'est pas faîte une fois pour toute en début de projet. Elle est à revoir au moins à chaque début de phase ou en cas d'incident ou de modification importante.

Obstacles

Il existe de nombreux écueils qui vont empêcher la réalisation d'un bon plan. On peut citer plusieurs exemples qui illustrent cet état de fait.

Vaincre la résistance
La planification est d'abord vécu par l'équipe comme un outil de contrôle. Si tout le monde est motivé et prêt à planifier les actions à entreprendre pour conduire à bien telle ou telle phase du projet, il en va tout différemment si l'on se trouve jugé (ou sanctionné) pour le non-respect de ses engagements.
Mauvaise estimation
Il existe une confusion entre estimation et planification. La seconde présuppose la première. Dans les faits, c'est l'inverse qui se produit : il paraît en effet plus facile de construire un planning des tâches que d'estimer des charges et des délais.
Optimisme
La tendance naturelle est de minimiser le risque afin d'éviter une trop forte tension au sein de l'équipe. *"On verra bien au moment opportun"* est une phrase trop souvent dite. Chacun a bien conscience de l'existence de points critiques mais la vigilance se relâche vite. L'optimisme reflète ici une confiance dans le professionnalisme artisanal de l'équipe.

Suivi et contrôle

Technique consistant à mettre en place les actions correctives suite aux variations dues à des événements extérieurs. Il s'agit de contrôler la conformité des travaux et du résultat face aux prévisions.

Obstacles

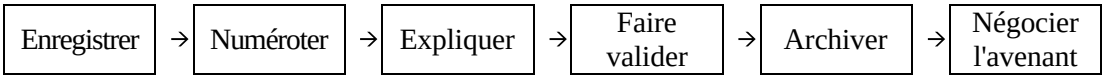
La véritable cause de réticence à la mise en place d'un système de suivi est le sentiment d'une perte de temps. En effet, la pratique du suivi fait appel à l'organisation d'une série de tâches hors projet. Il s'agit des réunions formelles qui s'accompagnent ensuite de la saisie d'information sur leur état des lieux. Il y a ainsi la production d'une série de documents comme les comptes-rendus d'activité, les fiches d'alerte, la tenue d'un livre de bord ou l'actualisation des plannings et des budgets. Tout ce travail administratif s'additionne au travail de fabrication du logiciel. Il n'est pas perçu comme partie intégrante du projet.

Considérés comme un ensemble de tâches non productives, le suivi comme le contrôle sont vécus comme un mal nécessaire par l'équipe et pis encore, par le chef de projets lui-même. Ce sentiment s'amplifie avec le décalage entre les rapports d'activité et la situation réelle du projet.

Objectif

Il s'agit de contrôler avant tout la dérive du projet. Celle-ci peut découler de plusieurs facteurs.

Des changements inévitables dus à l'évolution des spécifications. Attention, elles ne doivent toutefois pas dépasser 25 %. Ces évolutions sont naturelles mais elles doivent donner lieu, après des négociations, à un accord avec le client. L'important réside dans leur détection et dans leur gestion. Il est conseillé d'adopter une procédure pour les prendre en compte.



Des difficultés techniques peuvent survenir lors d'une phase du développement auxquelles on n'avait pas pensé. Il y a alors dépassement du calendrier.

Des dérives invisibles plus sournoises, peuvent apparaître. Elles naissent d'une accumulation de faits non significatifs. Par exemple, un des programmeurs affecté au projet répond par téléphone à la demande d'assistance d'un utilisateur sur un autre projet. Si l'intervention dure deux heures et se reproduit quatre fois dans la semaine, elle n'est pas comptabilisée et grèvera les délais.

Les principaux objectifs du suivi de projet sont donc de :

- Contrôler le déroulement du planning.
- Collecter des informations.
- Suivre les budgets.
- Détecter les écarts.
- Informer les responsables.
- Allouer de nouveaux moyens.
- Se créer un historique pour le futur.

Moyens

L'atteinte de ces objectifs se réalise au travers de réunions catégorisées de la façon suivante :

1. Réunion de suivi planifiées avec les différentes intervenants du projet qui comprennent :
    - Les réunions de suivi de projet (**RS**) qui concernent l'avancement du projet et l'affectation du travail
    - Les réunions d'avancement (**RA**) qui concernent le suivi du projet par phase
    - Les réunions décideurs (**RD**) qui permettent d'arbitrer
2. Réunion de suivi spécifiques pour la gestion des alertes

La **RS** se fait au niveau de l'équipe, une remontée formelle hebdomadaire est une pratique courante. Elle s'organise autour de la phase et est matérialisée par la production d'un compte-rendu d'activité. Ce dernier est nominatif, il sert au chef de projet pour la mise à jour du tableau d'avancement. Ces réunions de suivi de projet sont internes. Elles sont aussi l'occasion de réunir régulièrement l'équipe. Elles ont aussi pour but de résoudre en commun les problèmes, de maintenir le moral et de prévenir les conflits. En annexe 2 figure un exemple schématique de compte-rendu d'activité.

La **RA** est de périodicité mensuelle. Son objectif dépasse la simple remontée d'information, il vise la prise de décision dans la mise en place d'actions d'assistance de toute nature. Il prépare la réunion décideur. Elle réunit autour du chef de projet, les principaux responsables directement intégrés au projet comme le responsable utilisateurs, le responsable de l'exploitation ou le directeur informatique. Il est important de disposer d'indicateurs et de variables pendant son déroulement.

La **RD** survient en fin de phase ou à la demande du chef de projet. Elle a pour but d'informer le comité directeur (le client). Il s'agit de connaître le montant des ressources consommées en rapport avec le degré d'avancement du projet compte tenu des prévisions annoncées. Les deux questions qui seront posées sont *"à partir de quelle date le logiciel sera opérationnel ?"* et *"doit-on engager de nouvelles dépenses pour tenir l'objectif ?"*.

Documentation

C'est ce qui va représenter la réalité du projet. La documentation, comme les autres activités du projet, consomme des moyens en temps, en ressources et en outils. A ce titre, elle fait elle aussi l'objet d'une estimation, d'une planification et d'un suivi. N'oublions jamais l'équation d'un bon logiciel :

LOGICIEL = PROGRAMME + DOCUMENTATION

La diversité des acteurs comme des objectifs débouche sur une classification de la documentation suivant une normalisation (voir en annexe 3) et autour de trois pôles :

Le projet

La documentation regroupe l'ensemble des données sur les ressources, les délais et les coûts des travaux. Elle fournit au chef de projet les constats et les prévisions sur le déroulement du projet en terme d'événement ou de perturbation. Elle regroupe les documents comme les plannings, les comptes-rendus et les rapports d'activité. Elle s'inscrit dans une dynamique :

prévision ⇨ réalisation ⇨ suivi.

Le client	**Le produit**
La documentation est centrée sur la connaissance de l'usager, les caractéristiques de son poste de travail ainsi que les procédures d'organisation ou les ressources informatiques capables d'accueillir le nouveau produit. La documentation client fournit l'information utile à une meilleure intégration du système dans l'environnement utilisateur.	Composante à part entière du projet, la documentation regroupe toutes les données actualisées sur le logiciel depuis son architecture jusqu'à son mode d'utilisation. Véritable pierre angulaire du système, elle constitue la seule source d'information à partir de laquelle la formation, l'assistance et l'exploitation peuvent être mise en œuvre.

Les acteurs

Plusieurs acteurs évoluent et interviennent autour du projet. Ils ont des motivations diverses face à l'utilisation d'une documentation.

Le Chef de projet	Il souhaite disposer d'un maximum d'informations sur le logiciel et sur le client. En cas de dérive, il assurera le recentrage de la documentation sur l'objectif.
L'utilisateur	Il attend des moyens d'utilisation correcte du système pour le quotidien et de pouvoir se sortir d'affaire en cas d'incident.
Le développeur	Il attend une simplification de sa tâche pour réaliser le produit en conformité par rapport à la demande.
Le service informatique	Il attend des consignes d'exploitation, de sauvegarde et de reprise en cas de panne lui permettant une bonne prise en charge du logiciel dans le système existant.
Le formateur	Il attend une bonne compréhension du produit pour construire une action de formation efficace.
Le client	Il recherche à travers la documentation des moyens de suivre l'évolution du produit et se prémunir contre d'éventuelles malfaçons.
Le responsable utilisateurs	Il attend le fil conducteur lui permettant de vérifier la bonne adéquation par rapport aux besoins exprimés.

Obstacles

Tenue des délais

Une idée reçue et bien ancrée dans l'esprit des informaticiens est la forte consommation de temps découlant de la rédaction d'une documentation. Cette démarche ne résiste pas si une analyse approfondie de la répartition des temps de travail des membres de l'équipe est réalisée. Car l'absence de documentation plus ou moins formalisée débouche sur un surcoût en temps lié aux itérations successives découlant de l'imprécision de la demande ou de la non traçabilité de la décision prise lors de réunions antérieures. La solution consiste à inclure dans les estimations des délais, du temps pour la documentation et à la développer au fur et à mesure de l'avancement du projet.

Obsolescence

Autre argument avancé de toute bonne foi par les développeurs qui porte sur la gestion des mises à jour de la documentation. De nombreux exemples viennent illustrer ce problème. La structure de classement et la pagination choisies sont incompatibles pour envisager des mises à jour. Il faut retrouver les anciennes informations pour les remplacer par les nouvelles. Ces problèmes font qu'il y a plus souvent ajout que substitution d'informations. Cela se traduit lors de recherches futures par un véritable jeu de patience pour identifier la dernière version parmi un lot d'informations obsolètes.

Complexe de l'écrivain

La production d'une documentation, sans être un exercice de style, nécessite quand même des qualités dans l'expression écrite afin de se faire comprendre des lecteurs. La maîtrise de la langue et l'aspect pédagogique en sont deux caractéristiques essentielles. Si on y a joute un vocabulaire enrichi et une orthographe correcte, on comprend mieux le peu d'enthousiasme de l'équipe pour s'engager dans ce travail de rédaction.

Propriété intellectuelle

Enfin, un frein caché à la rédaction de la documentation réside dans le sentiment de dépossession de leur œuvre qu'éprouve le réalisateur. Car à partir du moment où quelqu'un d'autre possède les clés lui permettant de reconstituer ou de s'inspirer du logiciel fabriqué, il y a perte de pouvoir.

Arguments

La documentation est d'abord un outil de mémoire. Trop souvent les développeurs avancent comme documentation la présence des sources et des fameux commentaires insérés pour expliquer le code. Plusieurs années de pratique conduisent à remettre en cause ce postulat. Quiconque s'est trouvé confronté à la modification d'un programme dont il n'était pas l'auteur est un ardent défenseur de la documentation. Côté utilisateur, c'est la même chose, faute de disposer d'un support écrit, le savoir et le savoir-faire se transmet oralement et au coup par coup. Il y a alors perte de mémoire du produit au profit de pratiques polluantes. Cela se traduit à terme par l'abandon de l'application.

La documentation n'est pas seulement un mode d'emploi de l'applicatif. C'est aussi un moyen de contrôle permanent pendant la durée de la réalisation. Elle intervient dans la normalisation et la standardisation des procédures de codage du logiciel d'un développeur à un autre. Partie intégrante d'une démarche d'assurance qualité, elle réduira ainsi les dépenses de formation et simplifiera les corrections lors de la maintenance. La documentation reste un des moyens de faire adhérer l'équipe à des changements de comportement en imposant des réflexes. Elle sera donc établie au fur et à mesure de l'avancement du projet.

La conception d'un logiciel constitue la réponse à un besoin du client exprimé lors de la phase d'étude. L'absence d'une formulation claire et précise de la demande comme de la solution adoptée conduite à une incompréhension voire un conflit entre les acteurs. La pauvreté de la documentation entraîne une absence de communication entre utilisateurs et concepteurs. Beaucoup de projets échouent alors lors de leur implantation à la suite de ces problèmes d'incompréhension sur les finalités et les possibilités offertes par le produit.

Enfin la gestion de la maintenance n'en sera que simplifiée. En effet, les demandes d'évolution suite à des modifications réglementaires ou à des améliorations attendues par l'utilisateur sont nombreuses. La formulation de la demande doit se faire par la mise en place d'un formulaire type de façon à prendre en compte de manière efficace la demande.

Elaboration

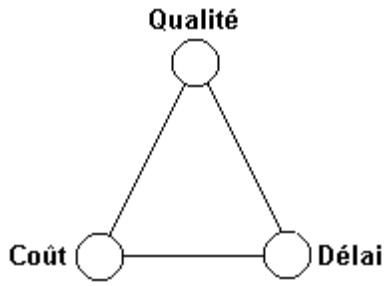
Dans le cas de petits projets, tout est à la charge du chef de projet. Celui-ci se voit confier la tenue des divers documents. Il capitalise la connaissance de l'ensemble du projet.

Une autre solution consiste à faire participer le client ou l'utilisateur du logiciel. Celui-ci construit par exemple, le manuel de prise en main à partir de sa compréhension du logiciel.

Dans le cadre de projets plus importants, la documentation se trouve intégrée dans les différents niveaux du dispositif de conduite du projet. Si l'ensemble est supervisé par le chef de projet, il est aidé dans cette tâche par une cellule technique chargée de la mise en forme du document suivant une charte graphique fixée au début.

Qualité

Le contrôle a posteriori reste insuffisant. La garantie d'un processus de fabrication de logiciel doit respecter des normes et des standards. Cela passe par la réalisation d'un plan d'assurance qualité. On peut résumer la qualité d'un projet à l'aptitude à développer le bon logiciel en maîtrisant les coûts et les délais fixés en satisfaisant le besoin de l'utilisateur.



On se rend compte alors que ces trois notions sont opposées et que le logiciel ne peut se positionner qu'à un seul endroit du rectangle.

Facteurs de qualité

Voici quelques questions à se poser et à poser à l'utilisateur sur le logiciel fourni :

Utilisation	Maintenance	Transfert
Conformité : fait-il ce que demande l'utilisateur ? **Fiabilité** : fait-il ce qui est demandé en toutes circonstances ? **Efficacité** : consomme-t-il uniquement les ressources nécessaires ? **Intégrité** : est-il protégé des erreurs de manipulations et des intrusions ? **Maniabilité** : est-il facile à utiliser ?	**Maintenabilité** : peut-on lui apporter facilement des corrections ? **Flexibilité** : peut-on lui supprimer ou lui ajouter des fonctionnalités ? **Testabilité** : peut-on mettre en œuvre des tests pour en vérifier le fonctionnement correct après des modifications ?	**Portabilité** : est-il facilement utilisable sur d'autres plates-formes ? **Compatibilité** : peut-il être facilement raccordé à d'autres logiciels ? **Réutilisabilité** : les modules développés peuvent-ils être intégrés dans d'autres applications ?

Critères de qualité

Auto description :	Attribut permettant d'expliquer comment est réalisé une fonction.
Banalité des communications :	Attribut dans lequel il existe des standards de réalisation des protocoles et des interfaces de communication.
Banalité des données :	Attribut dans lequel il existe des standards de représentation des données.
Clarté :	Attribut pour lequel les entrées et les sorties sont faciles à comprendre.
Cohérence :	Attribut dans lequel notations et terminologie sont uniformes.
Complétude :	Attribut dont tous les éléments constitutifs existent.
Concision :	Attribut tel qu'il ne comporte pas d'éléments inutiles ou redondants.
Efficacité d'exécution :	Attribut d'un logiciel qui n'utilise que le minimum de temps machine pour s'exécuter.
Efficacité mémoire :	Attribut d'un logiciel qui n'utilise que le minimum de place mémoire pour s'exécuter.
Extensibilité :	Possibilité d'ajout de nouvelles fonctions pour accroître la taille des données traitées.
Facilité d'apprentissage :	Attribut d'un logiciel facile à utiliser par des débutants.
Facilité d'utilisation :	Attribut pour lequel les données sont aisées à préparer et les résultats aisés à interpréter.
Généralité :	Attribut d'un logiciel dont le domaine d'application n'est pas très spécifique.
Indépendance machine :	Attribut d'un logiciel qui n'est pas lié aux spécificités matériel.
Indépendance système :	Attribut d'un logiciel qui n'est pas lié au système d'exploitation.
Instrumentation :	Attribut permettant de mesurer le fonctionnement ou l'identification des erreurs.
Modularité :	Attribut d'un logiciel pouvant être décomposé en éléments indépendants.

Observation des accès :	Attribut d'observation des accès au code ou aux données.
Précision :	Attribut d'un logiciel effectuant les calculs avec la précision requise.
Protection des accès :	Attribut d'un logiciel où le code et les données sont protégés contre les accès indus.
Robustesse :	Attribut d'un logiciel apte à poursuivre un fonctionnement correct malgré des conditions non conformes à celles prévues.
Simplicité :	Attribut pour lequel les choix de réalisation sont faciles à maîtriser.
Traçabilité :	Attribut par lequel on peut remonter de l'implémentation vers les spécifications.

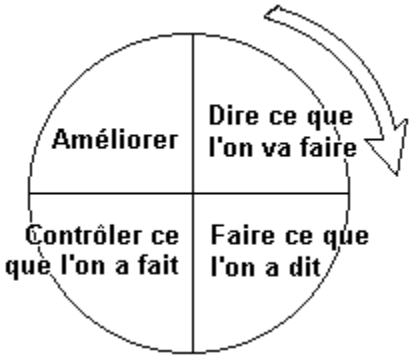
Relation facteurs/critères

Il est maintenant possible de dresser un tableau permettant de comprendre la relation pouvant exister entre un facteur et un critère. Celle-ci peut être compatible (+), antagoniste (-) ou neutre.

	Réutilisable	**Conforme**	**Compatible**	**Efficace**	**Fiable**	**Flexible**	**Intègre**	**Maintenable**	**Maniable**	**Testable**	**Portable**
Auto description :	+			-		+		+		+	+
Banalité des communications :	+		+								
Banalité des données :	+		+								
Clarté :	+			-		+		+	+	+	
Cohérence :	+	+			+	+		+		+	
Complétude :		+			+				+		
Concision :		+		+				+		+	
Efficacité d'exécution :				+							-
Efficacité mémoire :				+						-	-
Extensibilité :	+			-		+					
Facilité d'apprentissage :	+								+		
Facilité d'utilisation :	+			-			-		+		
Généralité :	+		+	-	-	+					
Indépendance machine :	+		+	-		+					+
Indépendance système :	+		+	-		+				+	+
Instrumentation :				-				+	+	+	
Modularité :	+		+	-		+	+	+			+
Observation des accès :			-	-							
Précision :				-	+		+		+		
Protection des accès :				-		-			+		
Robustesse :		+		-	+				+		
Simplicité :	+	+		+	+	+		+		+	+
Traçabilité :											

Assurance et contrôle qualité

L'important est d'être toujours dans un état d'esprit *"qualité"*. Celui-ci consiste à s'interroger pour fixer des objectifs, à entreprendre des actions, à mesurer des résultats, à analyser les écarts et les corriger et enfin à recommencer. De cela, on peut en faire le cycle de processus qualité :



Note :
Rappelons quand même que la recherche de la qualité est une démarche **sans fin** au long de laquelle on se fixe des objectifs **réalistes** avec une exigence de plus en plus grande.

S'agissant du contrôle de la qualité du produit, des listes peuvent être élaborées pour les différents documents de fin d'étape afin d'évaluer leur niveau de qualité. Dans le même temps, d'autres techniques peuvent être mise en œuvre.

Lecture simple
Elle nécessite l'auteur du document à contrôler et son lecteur. Son objectif est de vérifier le fond et la forme. Pour éviter que ce genre de contrôle ait un caractère inquisiteur, le lecteur ne doit pas être systématiquement le chef de projet, il est cependant conseillé qu'il appartienne à la même équipe pour des raisons de connaissance du projet. Cette technique se découpe en quatre étapes :

- Fourniture du document par l'auteur
- Lecture du document par le lecteur
- Retour du document annoté ou réunion
- Correction du document

Faisant partie des tâches réelles et sérieuses du projet, la lecture simple doit être planifiée et sa charge prévue dans le cadre du budget.

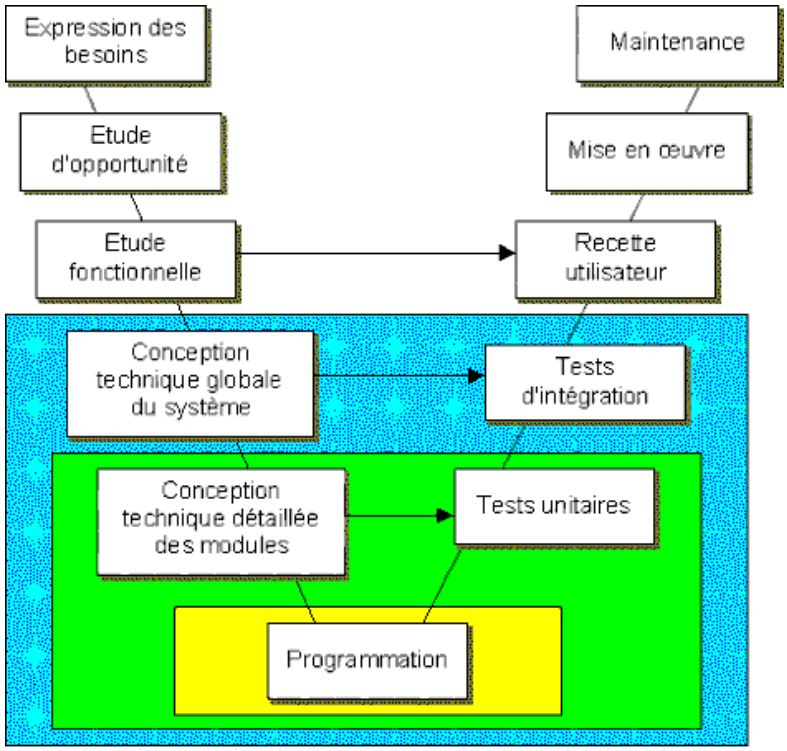
Lecture croisée
Elle s'adresse plutôt à des grands projets ou au contrôle de documents importants. Les différences avec la lecture simple sont les suivantes :

- Le nombre de lecteurs va de trois à cinq
- Les lecteurs peuvent être externes au projet
- Une réunion doit être obligatoirement organisée

Inspection
Elle s'applique particulièrement aux documents de nature technique pour lesquels il faut plus qu'un avis mais une expertise. Les intervenants sont l'auteurs, les inspecteurs (ou les experts) et le modérateur (qui joue le rôle de l'arbitre). Les inspecteurs ont ici pour tâche non seulement de relever les erreurs mais de proposer des solutions.

Revue de projet
L'objectif est de décider si l'équipe peut passer à la phase suivante. En plus de l'étude des différents documents, on statuera sur la validité de la démarche méthodologique suivie.

Le contrôle durant le cycle de vie du projet peut être vu comme des boîtes noires pour lesquelles on définit des spécifications et des tests.



Gestion d'équipes

La qualité d'une application repose pour beaucoup sur la performance des équipes engagés. Cela implique une structure de projet efficace et motivante, une relation client fournisseur confiante et un encadrement de l'équipe dynamique.

Le chef de projet

Le chef de projet agit comme un chef d'orchestre. Il a de nombreuses missions :

- Organiser les réunions de suivi, d'avancement et de décision.
- Gérer les ressources, planifier, contrôler et suivre les délais, les budgets et les moyens matériels ainsi que la sous-traitance.
- Informer, former et rendre compte aux différents acteurs du projet.
- Assurer la qualité, motiver l'équipe, résoudre les problèmes et les conflits.

Quels que soient les méthodes et les outils utilisés et l'organisation mise en place, il convient de ne jamais oublier que l'élément humain constitue le point central du problème. Il suffit d'une mésentente pour faire capoter le projet le mieux structuré. L'expérience montre que tout type d'encadrement non fondé sur la motivation et sur la compétence des hommes est vouée à l'échec.

Important !
La conduite de projet c'est un savoureux mélange de :
30 % de technique, 30 % de méthode et 40 % de communication

Etes-vous un bon chef de projet ? Avant de répondre, voici le profil idéal :
Capacités techniques
Plus que de la pratique au quotidien des techniques, c'est la compréhension de leurs spécificités, de leur évolution et leurs interactions qui importe.
Capacités d'organisateur
Le chef de projet joue le même rôle qu'un architecte ayant sous ses ordres plusieurs corps de métiers. Il doit coordonner, planifier et suivre une multitude d'événements.

Connaissance du domaine fonctionnel
Pour être crédible vis-à-vis de ses interlocuteurs clients, un bon bagage dans le domaine de gestion considéré est une garantie de reconnaissance.
Capacité de résolution de conflit
Sur la durée du projet, il n'est pas rare que des problèmes apparaissent au sein de l'équipe sur des options techniques voire sur des personnes. La capacité du chef de projet à résoudre ce type de problème sans recours à la hiérarchie est fondamental pour le maintien d'une bonne harmonie au sein de son équipe.
Capacité de diriger
Cela se traduit par une mise en avant permanente des objectifs du projet auprès des membres de l'équipe. Le chef de projet développe une disponibilité et une capacité d'écoute importantes. Il défend les intérêts de son équipe auprès de la Direction.

Toujours sur la brèche, le chef de projet doit également posséder une fibre de *"manager"*. Les principales composantes en sont :
L'engagement
Il reflète l'appropriation complète du projet par le chef de projet. Il se traduit par une adéquation des objectifs personnels et de ceux du projet.
La coopération
Il s'agit de réunir l'ensemble des conditions nécessaires à la création d'une équipe soudée, engagée et gagnante. Cela suppose la présence d'un climat de confiance, des relations efficaces entre partenaires, une production de groupe et un climat propice à la communication.
La motivation
Une équipe voit son moral passer par des hauts et des bas lors de la vie du projet. Elle perd souvent de vue le pourquoi et le comment du projet. C'est au chef de projet d'insuffler une motivation permanente à l'équipe par sa politique et son comportement. Une partie de cette motivation repose sur les éléments suivants :

- La maîtrise du temps. Il vérifie les dispositifs de conduite de réunion, de compréhension des informations. Il crée des courroies de transmission des actions à mener et des priorités.
- Le respect des décisions. Cela implique qu'il faille apprendre à dire non, à gérer les priorités et à décider vite.
- La communication directe. Il privilégie le contact direct à travers un processus oral informel. Il est préférable de réserver la communication écrite formelle pour la définition du cadre de la phase en terme de résultat et la communication écrite informelle sous forme de note pour s'assurer du bon *feed-back* du message.
- La délégation. Il confie des missions à ses subordonnés en leur précisant les objectifs, en s'assurant de la présence des moyens nécessaires et en laissant une marge d'initiative. Il prévoit un contrôle d'avancement et la mesure du résultat.

Organisation

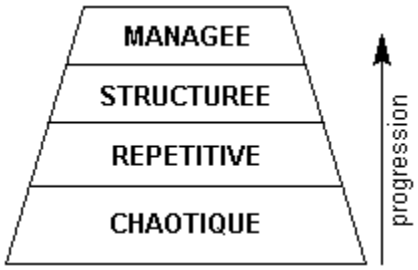
Dans le cas de petits projets, le chef de projet assure seul l'ensemble de ces fonctions. Pour assurer la qualité, il aura recours à la technique de la lecture simple. En revanche pour des projets importants, il lui faudra organiser et gérer le travail des membres de son équipe. Cette tâche sera l'activité la plus difficile à réaliser.

Avec la pratique, il est recommandé de ne pas avoir à gérer plus de 8 personnes en ligne directe. Dans le cadre de projet très important, il est conseillé de mettre en place une structure d'aide autour du chef de projet. Il s'agit de :
L'administrateur ou adjoint au chef de projet, il assiste celui-ci pour tout ce qui touche à la logistique et à la documentation du projet depuis la conception jusqu'à la réalisation.
Le support méthodes et qualité, il assiste les membres de l'équipe dans l'utilisation des méthodes et des outils mais aussi pour assurer la qualité des documents résultant du travail de chacun par l'intermédiaire de technique de lecture croisée.

A savoir :
Dans le cadre de projet supérieur à 10 personnes, il est conseillé d'éclater les deux fonctions du **support M&Q** au profit de deux postes distincts.

Conclusion

Sur la base du chapitre des Concepts, on peut conclure que les projets se classent sur quatre niveaux de maturité des processus de conduite de projet auxquels sont associés quatre stratégies de changement.



Maturité chaotique

Les symptômes de ce type de projet sont :

- La course au délai, une mise en œuvre de plans successifs malgré des échecs répétés
- Des efforts désespérés pour tenir les délais aux dépens des tests et de la documentation
- Une fuite en avant
- L'absence systématique d'étude sur la faisabilité réelle du projet tandis que la hiérarchie passe son temps à gérer les dérives.

Tout cela entraîne un fatalisme de l'équipe, le rejet des outils et la recherche de recettes miracles pour estimer et planifier alors que le projet croît en complexité.

Les remèdes proposés :

- Définir une structure du projet au sein d'un plan ou d'un schéma directeur.
- Négocier les conditions du succès
- Suivre et contrôler le bon avancement du projet

Pour passer au niveau supérieur :
Le chef de projet recherchera le succès de l'opération plutôt que la mise en œuvre de standard découlant d'une méthode de conduite de projet applicable à tous les projets.

Maturité répétitive

Les symptômes de ce type de projet sont :

- Il existe des outils de planification non standardisé mais construits à partir de projets similaires
- Le chef de projet suit l'avancement des travaux au cours de réunions plus ou moins formelles
- La notion de phase et de validation de fin de phase apparaît
- L'ébauche d'un contrôle qualité s'instaure au niveau de la réalisation
- La gestion de projet est constituée de techniques empiriques pour améliorer la qualité du produit fabriqué plus que celle du processus en lui-même

Pour passer au niveau supérieur :
Le chef de projet doit gérer l'évolution du logiciel. Il doit documenter les différentes représentations du logiciel au cours du cycle de vie et s'assurer d'un processus de fabrication de qualité.

Attention, le passage au mode structuré demande un travail collectif

Maturité structurée

Cette phase se caractérise par une stabilisation et une diffusion par osmose des techniques et des outils.

Les symptômes de ce type de projet sont :

- Une cellule méthode apparaît.
- Le découpage des activités du projet se fait en fonction de sa nature, il y a négociation des ressources et une estimation empirique des délais, des charges et des coûts.
- La documentation se formalise.

Toutefois, il manque encore :

- Une généralisation de la méthodologie
- Des techniques d'estimation et de planification
- Une véritable démarche contractuelle avec le client

Pour passer au niveau supérieur :

Le niveau *"managé"* implique la présence d'une série de réflexes et l'absence d'inhibitions afin d'accepter la mise en place de mesures qualitatives.

Maturité managée

Ce dernier niveau, le plus élevé indique qu'il y a beaucoup d'adeptes et peu d'élus. Ce processus se caractérise par une série d'attitudes permanentes dans l'amélioration du processus de fabrication du logiciel plus que par la recherche effrénée de recettes.

Le chef de projet a mis en place les actions suivantes :

- Une approche qualitative par la présence d'indicateurs de mesure adéquats
- Le refus de certains projets comme étant des non-projets
- L'encadrement des équipes dans une approche communicante

Cette maturité montre une véritable approche CLIENT-FOURNISSEUR.

Lois de GOLUB

Ces quinze lois (absurdes) qui datent de 1973, nous donnent les grands principes qui régissent encore aujourd'hui les projets en informatiques.

Loi n° 1	Aucun projet informatique n'est jamais mis en place dans les délais, dans les limites du budget et avec le même personnel qu'au départ. Le projet ne fait pas ce qu'il est censé faire et il est fort improbable que le vôtre soit le premier.
Loi n° 2	L'un des avantages de fixer des objectifs vagues à un projet est que vous n'aurez pas de difficultés à estimer les dépenses correspondantes.
Loi n° 3	L'effort nécessaire pour redresser le cap croît géométriquement avec le temps.
Loi n° 4	Les buts, tels que les entend celui qui en décide, seront compris différemment par les autres.
Loi n° 5	Seuls les bénéfices mesurables sont réels. Or les bénéfices immatériels ne sont pas mesurables. Donc les bénéfices immatériels ne sont pas réels.
Loi n° 6	Toute personne qui peut travailler à temps partiel pour un projet n'a sûrement pas assez de travail en ce moment.
Loi n° 7	Plus grande est la complexité d'un projet, oins vous avez besoin d'un technicien pour le diriger. Trouvez le meilleur manager possible, lui trouvera le technicien.
Loi n° 8	Un projet mal planifié prendra trois fois plus de temps à réaliser qu'in n'est prévu. Un projet bien planifié ne prendra seulement que deux fois plus de temps.

Loi n° 9	S'il existe un risque que quelque chose marche mal, cela marchera mal.
Loi n° 10	Quand les choses vont bien, quelque chose va aller mal. Quand les choses semblent aller mieux, c'est que vous avez oublié quelque chose.
Loi n° 11	L'équipe de projet déteste les comptes-rendus d'avancement des travaux parce que ceux-ci mettent trop vivement en lumière l'absence de progrès.
Loi n° 12	Les projets progressent rapidement jusqu'à 90 % de leur achèvement puis, ils restent achevés à 90 % pour toujours.
Loi n° 13	Si on laisse le contenu d'un projet changer librement, le taux de changement dépassera le taux d'avancement.
Loi n° 14	Si un utilisateur ne croit pas au système, il créera un système parallèle. Ni l'un, ni l'autre ne fonctionneront très bien.
Loi n° 15	Enfin, aucune loi n'est immuable.

Annexes

Annexe 1

RECAPITULATIF DES COÛTS

Projet :

Phase :
Durée prévue :
Date début :
Date de fin :

Personnel	Main d'œuvre.....		
	Formation.....		
	Déplacement.....		
Logiciel	Système d'exploitation.....		
	Licences		
	Outils		
Matériel	Achats directs	Serveurs	
		Ordinateurs	
		Périphériques.....	
		Eléments réseau.....	
	Location	Matériels	
		Lignes télécom	
	Infrastructure	Locaux.....	
		Mobilier	
	Fournitures	Papier.....	
		Autres	
		Total général :	

Annexe 2

Participants :

Période :

Réalisations de la période
Objectifs atteints :
Objectifs non atteints (liste, causes, conséquences) :

Prévisions période suivante
Objectifs et date :
Objectifs difficiles (liste et causes) :

Annexe 3

Plan documentaire du projet avec classification (selon norme AFNOR).

Documents de relation contractuelle
▪ Cahier des charges ▪ Proposition ▪ Conventions ▪ Engagement complémentaire ▪ Document de réception

Documents de gestion de projet
▪ Plan de développement ▪ Document de suivi de projet ▪ Document de bilan

Documents d'étude et de développement
▪ Rapport de l'étude préalable ▪ Rapport de l'étude détaillée ▪ Rapport de développement

Documents d'assurance qualité
▪ Plan d'assurance qualité ▪ Rapport d'audit interne ▪ Rapport d'évaluation en fin de phase ▪ Plan de test ▪ Document de coordination

Documents d'utilisation et de soutien
▪ Document de présentation générale ▪ Manuel de référence ▪ Document pédagogique ▪ Manuel d'utilisation ▪ Manuel d'installation ▪ Manuel d'exploitation

Les systèmes de gestion de base de données et Windev®

par Cyril Beaussier

Sommaire

Introduction

L'informatique et les systèmes qui la compose permet de stocker des données représentant des informations. Ces données sont regroupées au sein d'une base appelé ainsi base de données (database). Les exemples d'application utilisant des bases de données sont nombreux. On peut cependant les diviser en deux catégories.

Les bases de données classiques

Au sein desquelles on retrouve les applications les plus courantes : les bases de gestion (salaires, stocks…), les bases transactionnelles (comptes bancaires, centrales d'achats…), les bases de réservations (avions, trains…).

Les bases de données multimédias

Plus récentes, on les retrouve dans les domaines de la documentation, la géographie, le génie logiciel ou la C.A.O. (conception assisté par ordinateur).

Définition

La base de données

Une base de données est donc un gros ensemble d'informations qui sont structurées et mémorisées sur un support permanent.
Un fichier texte composé de noms est déjà une base de données même si l'accès aux informations est pénible puisqu'il faudra lire à chaque recherche l'ensemble du fichier en le parcourant séquentiellement.

Pour accéder plus rapidement aux informations, on a donc recours à un index. Le format de base de données de Windev, l'Hyper File est un système de ce type en étant composé d'un fichier de données et d'un fichier d'index. On retrouve ce type de base pour l'ISAM, ou le format dBase.

Le système HF, malgré ce que dit son éditeur PC SOFT, a ses limites. Une base de données de ce type mise en réseau face à plusieurs dizaines d'utilisateurs subit une fragmentation de ses données très importante. En effet, si l'accès en lecture ne pose à priori pas de problème, c'est l'accès en écriture (création, suppression ou mise à jour) qui fait risquer à la base une incohérence entre ses données et son indexation. La base est en effet, directement attaqué par le programme, elle n'a pas d'intelligence propre.

Le SGBD

Si la base de données n'est composé que de données, le système de gestion de base de données est en fait un logiciel couplé à une base de données. Un système de gestion de base de données (couramment appelé SGBD) est un logiciel de haut niveau qui permet de manipuler ses informations.

Le système de gestion de base de données est architecturé sur trois niveaux :

1. **Le niveau physique**. C'est ce que fait le SGBD physiquement. La gestion des données et des index. Le partage de ces données et de la concurrence des accès. La reprise sur panne. La distribution des données à travers le réseau.
2. **Le niveau logique**. C'est ce que fait le SGBD logiciellement. La définition de la structure des données. La gestion de la confidentialité (sécurité). Le maintien de l'intégrité entre les données et les index. La consultation et la mise à mise à jour des données.
3. **Le niveau externe**. C'est ce que voit l'utilisateur. Il opère sur le SGBD à travers différents outils. Les programmes de saisie ou d'édition d'états. Les outils de programmation et de débogage.

Le système de gestion de base de données va donc permettre de gérer un important volume d'informations pour un grand nombre d'utilisateurs. Ces informations seront persistantes (sur plusieurs années). Elles resteront fiables face à des pannes physique ou logique. Elles seront partageables (tant au niveau utilisateur qu'au niveau programme). Elles pourront être manipulées indépendamment de leur représentation physique.

Préambule

Ce document est à coupler avec *la programmation en client/serveur avec Windev* disponible sur mon site à l'adresse www.beaussier.free.fr.
Ce document ne rentre pas dans les détails de l'administration des systèmes de gestion de base de données sur serveur dédié de type Oracle par exemple.

Conception

Il existe de nombreux systèmes de gestion de base de données. Certains peuvent tourner sur micro-ordinateur comme Access, d'autres doivent se placer sur des serveurs dédiés comme *Oracle*, *Sybase*, *DB2* ou *SQL/Server*. La décision du choix dépendra du nombre de transactions qui s'opérera sur le SGBD.

Bien entendu, je conseille fortement de ne pas se lancer sur des SGBD serveur. Sans faire le pro Microsoft, MS-ACCESS représente une bonne alternative qualité/prix. Le moteur du SGBD est performant. La conception du modèle de données est facilité par l'interface graphique.

Schéma de données

C'est la façon de représenter les informations du monde réel dans le système de gestion de base de données. Comme pour la création de l'analyse des données dans Windev, le modèle de données dans Access est similaire avec :

- Une structuration des objets
- Les opérations sur ces objets

Opération sur les données

Il existe quatre types d'opérations classiques. Ces opérations correspondent à des requêtes sur le système de gestion de base de données.

- La création (ou l'insertion)
- La modification (ou la mise à jour)
- La destruction
- La recherche

L'opération la plus complexe est la recherche en raison de la richesse des critères.

Concurrence d'accès

Le système de gestion de base de données doit pouvoir répondre aux requêtes de plusieurs utilisateurs. Ces utilisateurs de leur côté doivent pouvoir accéder en même temps aux mêmes données. Le SGBD doit donc savoir :

- Gérer les conflits si deux utilisateurs font des mises à jour.
- Offrir un mécanisme de retour en arrière si l'utilisateur décide d'annuler des modifications en cours.
- Donner une image cohérente des données si un utilisateur fait des recherches et un autre fait une mise à jour.

Le but final est bien sûr d'éviter les blocages et les lenteurs, tout en empêchant des modifications anarchiques.

Le langage SQL

Le SQL est l'abréviation de Structured Query Language (traduction pour Langage Structuré de Requête), c'est le mode de communication avec le système de gestion de base de données. Il a été standardisé par l'ANSI et adopté en 1986 par l'ISO/IEC.

Principe

L'opération de recherche est la plus complexe. L'opération s'exprime par une projection, une sélection et une jointure par un bloc de commande.

SELECT < liste des attributs à projeter >
FROM < liste des tables ou des arguments >
WHERE < conditions sur un ou plusieurs attributs >

La normalisation SQL

Le SQL en est à sa version 2 (ISO 91). Ses principales caractéristiques sont :

- La standardisation des codes réponses en ajoutant la variable de retour sqlState (géré par Windev avec la commande sqlErreur).
- La possibilité de renommer les colonnes résultat
- La possibilité d'utiliser les mots réservés comme nom de table ou d'attribut.

Attention, certains SGBD ont des mots supplémentaires qui ne sont pas standardisé SQLv.2. Il est bien sûr fortement recommandé de ne pas les utiliser.

Là encore Windev reconnaît que son moteur Hyper File n'est pas un véritable SGBD et qu'il n'est absolument pas compatible avec le SQLv.2. Il ne faut donc pas utiliser le langage SQL sur une base HF. Le pilote ODBC fourni pour Hyper File ne peut exécuter que des requêtes de recherche simple (SELECT).

Vocabulaire

Dans une base de données classique, on parle de fichier, dans un système de gestion de base de données on parle de table. Même chose pour les champs d'un fichier qui sont appelés attributs dans une table. Une table est composé de colonnes, chaque colonne est qualifié par un attribut. Lorsque deux tables sont liés par un attribut, on appelle cela une jointure.

Les mots réservés

SELECT, FROM, WHERE	Expression de base
ALL, DISTINCT	Permet d'avoir tout ou de supprimer les doublons
=, <>, <, >, <=, >=?	Opérateur de comparaison
BETWEEN, AND, OR	Opérateur d'intervalle
LIKE	Opérateur de comparaison de texte
ALL, ANY, IN, EXISTS	Permet d'opérer sur des sous-requêtes
UNION	Permet de croiser deux requêtes
COUNT, SUM, AVG, MAX, MIN	Fonction de calcul
GROUP BY, HAVING	Permet le regroupement de données
INSERT INTO	Création
UPDATE SET	Mise à jour
DELETE	Suppression

Expression simple sur une table

Sur le schéma de données d'une gestion des commandes clients d'une société d'outillage, voici quelques requêtes SQL simples juste pour vous familiariser avec le langage.
Sachant que le SGBD est composé d'une seule table décrite ci-dessous.

La table `Client` est composée de son numéro (`NumCl`), de son nom (`NomCl`), de son adresse (`AdrCl`), la ville (`VilleCl`) et du chiffre d'affaire qui lui est associé (`CaCl`).

Langage naturel	Expression SQL

Informations sur toutes les clients	`SELECT NumCl, NomCl, AdrCl, CaCl`
	`FROM Client ;`
	ou…
	`SELECT *`
	`FROM Client ;`

Le nom des clients dont le C.A. est compris entre 10.000 et 50.000 F	`SELECT NomCl`
	`FROM Client`
	`WHERE CaCl BETWEEN 10000 AND 50000 ;`

Les nom et adresse des clients dont le nom commence par C	`SELECT NomCl, AdrCl`
	`FROM Client`
	`WHERE NomCl LIKE 'C%' ;`

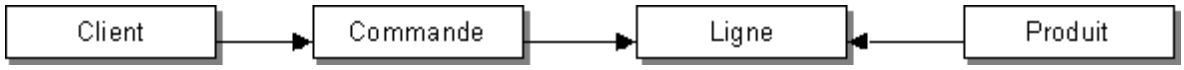
Combien de clients dans la table ?	`SELECT COUNT(*)` `FROM Client ;`

Les noms des clients classés par chiffre d'affaire	`SELECT NomCl`
	`FROM Client`
	`ORDER BY CaCl ;`

Notons que le symbole étoile (*) remplace la liste de tous les attributs, que le symbole pourcentage (%) remplace une chaîne de caractères quelconque et que les chaînes de caractères doivent être placées entre cote ('). Enfin le point-virgule (;) signale la fin de la requête.

Expression simple sur plusieurs tables

Compliquons maintenant en ajoutant une table Commande et une table Produit. Un client passe une commande composée de produits. Le lien est entre la commande et les produis est une table Ligne. Le modèle des données est alors le suivant :



La table `Commande` est composée de son numéro (`NumCo`), du n° du client (`NumCl`) et du total hors taxe (`TotalCo`).
La table `Produit` est composé de son numéro (`NumPr`), de son nom (`NomPr`), de son prix (`PrixPr`) et de la quantité en stock (`QtePr`).
La table `Ligne` est composé des n° de commande (`NumCo`) et de produit (`NumPr`) et de la quantité commandée (`QteCo`).

Langage naturel	Expression SQL

Qui a passé la commande n° 15 ?	`SELECT NomCl`
	`FROM Client Cl, Commande Co`
	`WHERE Cl.NumCl = Co.NumCl`
	`AND NumCo = 15 ;`

Combien de commande a passé le client Martin ?	`SELECT COUNT(*)`
	`FROM Client Cl, Commande Co`
	`WHERE Cl.NumCl = Co.NumCl`
	`AND NomCl = 'MARTIN' ;`

Le nom des produits de la commande n° 24	`SELECT P.NomPr`
	`FROM Produit P, Ligne L`
	`WHERE P.NumPr = L.NumPr`
	`AND NumCo = 24 ;`

Langage naturel	Expression SQL

Les clients ayant commandé des tournevis	`SELECT NomCl`
	`FROM Client Cl, Commande Co,`
	`Ligne L, Produit P`
	`WHERE Cl.NumCl = Co.NumCo`
	`AND Co.NumCo = L.NumCo`
	`AND L.NumPr = P.NumPr`
	`AND NomPr = 'Tournevis' ;`

Dans le cas de jointure entre deux tables, il faut préciser dans la requête les attributs que l'on veut récupérer. Cependant, si les noms des attributs sont identiques, il faut préciser le nom de la table devant. Pour éviter des requêtes trop longues, on peut alors remplacer le nom de la table par un alias. C'est cet alias unique qui se placera devant l'attribut.

Expression complexe sur une table

Langage naturel	Expression SQL

Les noms des clients et leur chiffre d'affaire qui sont supérieur au client n° 143	`SELECT NomCl, CaCl`
	`FROM Client C1, Client C2`
	`WHERE C2.NumCl = 143`
	`AND C1.CaCl > C2.CaCl ;`

Revenons au système de l'alias, C1 et C2 représentent deux instances de la même table. La première condition permet d'éliminer les clients qui n'ont pas le n° 143 et la seconde permet d'obtenir le chiffre d'affaire du n° 143 pour le comparer aux autres.

Expression complexe ou imbriquée

Le langage SQL peut aller très loin en croisant le résultat d'une requête avec une autre ou en récupérant le résultat d'une requête pour en refaire une sélection avec l'aide d'une seconde.

Langage naturel	Expression SQL

Les noms des produits non commandés	`SELECT NomPr`
	`FROM Produit`
	`WHERE NumPr NOT IN (SELECT DISTINCT NumPr`
	`FROM Ligne) ;`

Les noms des clients ayant passé au moins une commande	`SELECT NomCl`
	`FROM Client`
	`WHERE NumCl EXISTS (SELECT NumCl`
	`FROM Commande) ;`

Le nom des clients n'ayant jamais commandé	`SELECT NomCl`
	`FROM Client`
	`WHERE NumCl NOT EXISTS (SELECT NumCl`
	`FROM Commande) ;`

Notons que le SGBD va d'abord exécuter la requête se trouvant entre parenthèses. Le résultat sera alors mis en corrélation avec la requête principale tapé au début.

Expression avec fonctions de calcul

Le langage SQL est très souple également puisqu'il peut calculer directement un résultat de requête et le renvoyer. On a vu plus haut que l'on pouvait déjà compter le nombre d'enregistrement par la fonction `COUNT`, le langage SQL permet également d'additionner (`SUM`), de faire une moyenne (`AVG`) ou renvoyer la valeur la plus grande (`MAX`) ou la plus petite (`MIN`). Le langage tolère également les opérateurs de comparaison égal (=), supérieur (>), supérieur ou égal (>=), inférieur (<), inférieur ou égal (<=) et différent (<>).

Voyons ci-dessous quelques exemples très simples qui récapitule ce que l'on a vu plus haut afin de vous montrer la souplesse du langage SQL :

Langage naturel	**Expression SQL**

Nombre de tournevis commandé	`SELECT COUNT(DISTINCT NumPr)`
	`FROM Produit P, Ligne L`
	`WHERE L.NumPr = P.NumPr`
	`AND NomPr = 'Tournevis' ;`

Quantité total de tournevis commandé	`SELECT SUM(QteCo)`
	`FROM Ligne L, Produit P`
	`WHERE L.NumPr = P.NumPr`
	`AND NomPr = 'Tournevis' ;`

Chiffre d'affaire moyen des clients	`SELECT AVG(CaCl)` `FROM Client ;`
Le plus gros chiffre d'affaire	`SELECT MAX(CaCl)` `FROM Client ;`
Le plus petit chiffre d'affaire	`SELECT MIN(CaCl)` `FROM Client ;`

Les clients ayant commandé plus que la moyenne des quantités commandées de tournevis	`SELECT NomCl`
	`FROM Client Cl, CommandeCo, Ligne L`
	`WHERE Cl.NumCl = Co.NumCl`
	`AND Co.NumCo = L.NumCo`
	`AND QteCo > (SELECT AVG(QteCo)`
	`FROM Ligne L, Produit P`
	`WHERE L.NumPr = P.NumPr`
	`AND NomPr = 'Tournevis' ;`

Expression d'agrégation

Langage naturel	**Expression SQL**

Nombre de clients par ville	`SELECT VilleCl, COUNT(NomCl)`
	`FROM Client`
	`GROUP BY VilleCl ;`

Dans cette exemple, nous voyons qu'il est possible de regrouper les résultats d'une requête suivant l'attribut. Voyons ce que fait exactement la requête :

Contenu de la base

VilleCl	NomCl
Paris	MARTIN
Paris	HENRI
Paris	CHARLES
Lyon	ROBERT
Lyon	GERARD

Résultat de la requête

VilleCl	COUNT(NomCl)
Paris	3
Lyon	2

Notons que s'il y a une condition `GROUP BY`, l'attribut contenu dans celle-ci doit figurer obligatoirement dans la clause `SELECT`.

Continuons avec d'autres exemples…

Langage naturel	**Expression SQL**

La quantité moyenne de chaque produit	`SELECT NomPr, AVG(QtePr)`
	`FROM Produit`
	`GROUP BY QtePr ;`

Langage naturel	Expression SQL

Le classement des produits par nombre commandé	`SELECT NomPr`
	`FROM Produit P, Ligne L`
	`WHERE P.NumPr = L.NumPr`
	`ORDER BY QteCo`
	`GROUP BY NomPr ;`

Compliquons maintenant la requête avec la clause HAVING. Celle-ci permet d'éliminer des partitionnements.

Langage naturel	Expression SQL

Produits commandés par plus de deux clients en quantité supérieure à 5	`SELECT NomPr`
	`FROM Produit P, Ligne L, Commande C`
	`WHERE P.NumPr = L.NumPr`
	`AND C.NumCo = L.NumCo`
	`AND QteCo > 5`
	`GROUP BY NomPr`
	`HAVING COUNT(NumCl) >= 2 ;`

Prenons de nouveau le contenu de notre base…

Avant la clause HAVING

NomPr	NumCl	QteCo
Marteau	12	6
Tournevis	12	6
Tournevis	18	8

Après la clause HAVING

NomPr	NumCl	QteCo
Tournevis	12	6
Tournevis	18	8

Expression d'écriture

Comme nous l'avons vu plus haut dans le chapitre Opération sur les données. Les requêtes d'écriture à la différence des requêtes de recherche, modifient le contenu de la base. Il existe trois requêtes d'écriture : la création (`INSERT`), la modification (`UPDATE`) et la destruction (`DELETE`). Ces requêtes doivent être manipulées avec précaution car une mauvaise rédaction peut entraîner des catastrophes irrémédiables.

Langage naturel	Expression SQL
Ajout du produit Pince au prix de 50 F. et en quantité 10	`INSERT INTO Produit(NomPr, PrixPr, QtePr)` `VALUES('Pince',50,10) ;`

Modification de l'adresse du client n° 17	`UPDATE Client`
	`SET AdrCl = '5 rue Petit',`
	`VilleCl = 'PARIS'`
	`WHERE NumCl = 17 ;`

Destruction du client dénommé MARTIN	`DELETE FROM Client` `WHERE NomCl = 'MARTIN' ;`

Notons qu'une requête `INSERT` ressemble beaucoup à une requête `SELECT` notamment dans la rédaction des valeurs à ajouter.

Il est très important de préciser la clause `WHERE` dans les requêtes `UPDATE` et `DELETE` car l'oubli ou la mauvaise rédaction des attributs peut causer des dégâts importants. Oublier la clause `WHERE` dans une requête `DELETE`, effacera **tous** les enregistrements de la table.

Optimisation du SGBD

Pour optimiser un système de gestion de base de données, l'administrateur ou le concepteur de la base peut ajouter des index ou des vues sur ces tables afin d'accroître les temps d'accès et de réponse.

Les index

Comme pour une base de données classique, l'index va permettre au SGBD d'accéder plus rapidement à l'enregistrement voulu. Il permet également d'assurer l'unicité des clés primaire ou secondaire. En résumé, un index est une structure contenant l'adresse physique de chaque enregistrement d'une table. Un index permet l'accès direct à l'information.

Que doit-on indexer :
- Toujours indexer les clés primaires des tables dont le nombre dépasse 100 lignes. Une clé primaire est une clé dont le numéro n'aura aucun doublon (exemple : le numéro de client).
- Toujours indexer les attributs intervenants dans des jointures entre tables. Par exemple, le n° de produit que l'on retrouve dans les tables Produit et Ligne.
- Indexer les attributs intervenants dans les clauses ORDER BY, GROUP BY et les fonctions de calcul MIN, MAX, etc.

Que ne doit-on pas indexer :
- Les colonnes contenant peu d'enregistrements
- Les colonnes contenant peu de valeurs différentes. En règle générale, lorsque 20 % ou plus des occurrences d'une satisfait le critère de recherche.
- Les colonnes très souvent modifiés et très peu utilisées lors de recherche.

Pour la création d'index, reportez vous au manuel d'utilisation de Microsoft Access.

Les vues

Une vue est une perception logique sur les données d'une ou de plusieurs tables ou même d'autres vues. Elle est définie à partir d'une requête d'interrogation du langage SQL. Elle hérite des mêmes caractéristiques que les objets auxquels elle se réfère.

Une vue est définie pour :
- Fournir un niveau de sécurité supplémentaire sur les données d'une table. Par exemple, un employé A ne peut consulter les informations d'un employé B dans un logiciel de gestion de personnel (salaire, prime, etc).
- Donner à l'utilisateur l'impression que la base lui appartient.
- Cacher une certaine complexité des données. Une vue pouvant être créer à partir de plusieurs jointures entre des tables différentes.
- Simplifier la formulation de requête SQL.
- Présenter les données d'une table sous un autre aspect sans en modifier la définition.
- Sauvegarder d'une façon indirecte les requêtes complexes.

Les vues ne peuvent être que créer ou remplacer.

Un exemple de vue :

Langage naturel	**Expression SQL**

Avoir directement les gros clients	`CREATE VIEW GrosClient(NomGc, AdrGc, VilleGc) AS`
	`SELECT NomCl, AdrCl, VilleCl`
	`FROM Client`
	`WHERE CaCl > 100000`

Limites de l'optimisation

Malgré tout la performance du SGBD peut très vite aller en décroissant. Celle-ci se révèle souvent par une mauvaise définition de la structure logique. Une mauvaise analyse donnera un mauvais schéma de données. Il en sera de même, si les requêtes SQL sont mal

formulées. Le support physique est également important, un serveur sous-dimensionné (manque de place disque ou de mémoire) ou un réseau mal conçu grèvera les temps d'accès et de réponse du SGBD.

Critère de choix d'un SGBD

La question du choix du système de gestion de base de données est essentiel. Celui-ci représente le cerveau de votre application. Une partie client (les programmes) excellemment bien développé ne servira à rien si le SGBD n'est pas conforme.

Voici quelques questions essentiels à se poser :

1. Quels sont les systèmes d'exploitation supportés ?
2. Quel est le volume disque nécessaire pour supporter les modules du SGBD ?
3. Peut-on gérer la même base sur plusieurs serveurs ?
4. Quel est le nombre maximum d'utilisateurs ?
5. Quelle est la taille maximum d'une base ?
6. Quel est le nombre maximum de tables autorisé et leur taille maximum ?
7. Y a-t-il un nombre limite d'enregistrement, de lignes ou de colonnes ?
8. Combien y a-t-il de types de données différentes (entier, texte, date, monétaire, etc.) ?
9. Quelles sont les limites des clés et des index ?
10. Peut-on créer des vues multi-tables ?
11. Y a-t-il des outils de maintenance du SGBD ?
12. Comment fonctionne la sécurité (blocage des enregistrements, des tables, etc.) ?

Remerciement

Je tiens par le présent document à remercier Monsieur SCHOLL auprès duquel j'ai suivi pendant une année les cours du C.N.A.M. (www.cnam.fr) sur les systèmes de gestion de base de données.

Copyright

© Cyril Beaussier - Courriel : cyril.beaussier@mail.dotcom.fr
Création initiale : 29/02/2000
Dernière date de révision : 25/06/2001

Programmation avancée sous Windev®

POO

ou la programmation orientée objet

par Cyril Beaussier

Version 1.0a - Juin 2000

Sommaire

Préambule

J'ai programmé sous Windev pendant plus de dix ans. J'ai commencé avec la version 1.5 (eh oui !). Il y a quelques années, j'ai eu à m'intéresser à la POO (ou programmation orientée objets). Ce document traite ainsi de mon expérience et de mon apprentissage de ce vaste sujet. J'y reprends d'ailleurs certains passages de la documentation fourni avec Windev 4.1x, version qui a été utilisée pour la rédaction de ce support et notamment la syntaxe du code. Et si aujourd'hui la version 7 est sortie, il reste toujours d'actualité pour démarrer dans ce type particulier de programmation.

Par ailleurs, il y a peut être des inepties, des erreurs (ou pis) dans certains passages de ce document. Si vous en constatez, je vous remercie de m'en faire part immédiatement àfin de correction (mon courriel est en fin de document).

Introduction

Pour des raisons de simplification, nous emploierons indifféremment le terme de POO pour programmation orientée objets et inversement. Lorsque l'on parle de POO dans un langage (C++, Java, Windev…), on parle avant tout de classe. Cette classe est le principal outil pour coder sous la forme d'objets notamment en W-Langage.

Définition

La POO est une méthode de programmation dans laquelle les programmes sont organisés comme des ensembles d'objets coopérants. Chacun représente une instance d'une certaine classe, toutes les classes étant des membres d'une hiérarchie de classes unifiée par des relations d'héritage.

On peut donc dire que le W-Langage est orienté objet dans une certaine mesure (car il est dérivé du C++), en effet :

- Il supporte les objets.
- Les objets ont une classe associée.
- Les classes peuvent hériter d'attributs venant d'objets appelés également "super classes" (on dit aussi *métaclasses*).

Terminologie

La POO utilise donc des objets (logique non ?), voici quelques définitions que je vous donne brut de coffrage et que nous étudierons plus en détail par la suite.

- Chaque objet est l'instance d'une certaine classe.
- Une classe est constituée de membres et de méthodes.
- Les classes sont reliées les unes aux autres par des relations d'héritage.

Pour mieux comprendre, il faut voir l'objet comme une espèce de *boîte noire*. C'est sur cet objet que vous faîtes des demandes. Ces demandes sont exploitées par l'objet avec ses méthodes. Enfin l'objet vous renvoie des données.

Exemple

Concrétisons maintenant tout ce charabia technique par quelque chose de simple. Prenons l'exemple d'une application gérant un parc automobile. Dans ce parc, nous pouvons aussi bien avoir une voiture, une moto, un camion. Bref, l'objet générique permettant de rassembler tous ces objets est donc un objet véhicule.

Décrivons donc cet objet de base : un véhicule. Il est donc décrit en programmation dans une classe. La classe véhicule a une structure précise. Elle est ainsi composée de membres et codifiée comme suit.

```
Vehicule est une classe
      Marque      est une chaîne
      Modele      est une chaîne
      Immatric    est une chaine
      Puissance   est un entier
      DateAchat   est une chaîne
Fin
```

Concept de base

Classe, méthode et objet

On vient de voir dans l'exemple ci-dessus qu'un objet sous Windev est représenté en programmation sous la forme d'une classe. Cette classe véhicule est composée de membres (marque, modèle, etc.). Ces membres représentent la structure des données de l'objet véhicule. Mais une classe est aussi constituée de méthodes. Ces méthodes sont des fonctions qui permettent de manipuler les membres de la classe.

On peut ainsi appliquer sur la classe véhicule, une méthode "prix vignette" dont la caractéristique coût sera fonction du membre puissance. La codification de cette méthode est décrite ainsi :

```
FONCTION PrixVignette()
Cout est un réel
Selon :Puissance
      Cas 4
            Cout = 250.00
      Cas 5
            Cout = 345.00
      Cas 6
            Cout = 480.00
      // etc
Fin
RENVOYER Cout
```

Une classe définit donc un type de donnée et son comportement. Elle permet de créer des objets. Chaque objet créé possède les membres décrits dans sa classe et peut-être manipulé par des méthodes. On dit alors qu'un "objet est une instance de la classe". Une classe peut être considérée comme un modèle qui définit les membres et les méthodes communes à plusieurs objets.

Une fois la classe décrite, elle est utilisée sous la forme d'objet en programmation d'où le terme de POO. Le membre doit être associé à sa classe par le séparateur **:** (deux points).

```
Voiture est un objet Vehicule

Si Voiture:DateAchat < "01/01/1998" Alors ...
      Info("Contrôle technique obligatoire !")
```

Chaque objet créé contient les membres de sa classe et peut être manipulé par ses méthodes. Ainsi vous pouvez avoir la syntaxe suivante pour avoir le prix de la vignette :

```
Info("Le prix de la vignette est " + Voiture:PrixVignette() )
```

Simple non ?

Héritage

L'héritage permet d'inclure les caractéristiques d'une classe existante (classe de base) dans une nouvelle classe (classe dérivée). Cela permet de créer un nouveau type de donnée à partir d'un type connu, dans le but de lui ajouter des fonctionnalités, ou d'en modifier le comportement. La classe de base n'est pas modifiée. Une classe peut hériter d'une ou de plusieurs autres classes dont elle devient une sous-classe.
Une classe dérivée hérite donc des membres et des méthodes de sa ou ses classes mères, en plus de ses propres membres et méthodes. Il n'est pas nécessaire de dupliquer les membres et méthodes de la ou des classes mères. Ce qui fait une économie en terme de codage importante.

Reprenons notre classe Vehicule, celle-ci manque cruellement de renseignements techniques. On peut très bien créer une classe dérivée et nommée DetVehic qui contiendra les détails qui ne sont pas essentiels à la classe de base.

```
Vehicule est une classe
      Marque      est une chaîne
      Modele      est une chaîne
      Immatric    est une chaine
      Puissance   est un entier
      DateAchat   est une chaîne
Fin

DetVehic est une classe
      Un objet Vehicule
      Carburant   est une chaîne
      Couleur     est une chaîne
      NbPlaces    est un entier
Fin
```

A noter :

Notons la syntaxe "`Un objet NomObjet`" pour rappeler la classe de base (appelé aussi classe ancêtre).

Constructeur et Destructeur

La notion de *Constructeur* et *Destructeur* est importante puisqu'elle permet un appel automatique de méthode lors de la création d'un objet et lors de sa destruction. Le *Constructeur* et le *Destructeur* sont en fait, des méthodes particulières.

- La méthode *Constructeur* associée à une classe est automatiquement appelée lors de la déclaration d'un objet de la classe (dès l'écriture de "`Voiture est un objet Vehicule`"). Cela permet de s'assurer que les traitements d'initialisation de l'objet (l'affectation des membres par exemple) ne seront pas oubliés par le développeur (nous sommes tous perfectibles☺).
- La méthode *Destructeur* associée à une classe est automatiquement appelée lors de la suppression de l'objet (sortie de procédure dans laquelle l'objet a été déclaré). Cela permet de libérer sans risque d'oubli, les ressources utilisées par l'objet (zone mémoire occupée). Elle peut aussi être utilisée pour mettre à jour un fichier relatif à l'objet par l'enregistrement de ses données.

Encapsulation de données

L'encapsulation des données est sans aucun doute la notion la plus importante de la POO. Cette technique permet de garantir que les données membres de l'objet ne seront pas modifiées à tort par des fonctions (méthodes) extérieures à l'objet. Il est ainsi possible d'interdire à l'utilisateur d'un objet l'accès à certain ou à tout ses membres. Les membres dont l'accès est interdit sont appelés membres privés.
Il n'est possible d'y accéder qu'à partir des méthodes prévues à cet effet dans la classe.

Durée de vie

L'objet est créé lors de sa déclaration. Par défaut l'objet est "privé" (ou local). L'objet est automatiquement détruit à la fin du traitement contenant sa déclaration.
Un objet déclaré global dans le code d'initialisation d'une fenêtre sera détruit à la fin du traitement de fermeture de la fenêtre. Un objet déclaré global dans le code d'initialisation d'un projet sera détruit à la fin du traitement de fermeture de la première fenêtre du projet.

Stockage

La description des objets (membres et méthodes) est stockée dans un fichier ayant l'extension WDC (WinDev Class). Notons que le nom logique de la classe peut être différent du nom physique du fichier (noté sur 8.3) `classe.wdc` dans laquelle est placée la classe.

Cas d'école

Maintenant que vous savez l'essentiel de la POO, examinons sous l'éditeur de Windev comment réaliser notre première application orientée objets. Je prendrai volontairement un exemple simple : une conversion de chiffres en lettres.

Conversion simple

Nous allons d'abord faire une conversion très simple sur des nombres. Si l'utilisateur tape le chiffre 1 et clique sur le bouton Conversion, le logiciel affiche "un". S'il tape 2, "deux" sera affiché, etc. On s'arrêtera à dix mais vous pouvez aller jusqu'où vous voulez.

Créons d'abord la classe cConversion. La classe est tout d'abord initialisée avec les données qui seront envoyées à l'objet (les entrées) et celles qui seront renvoyées (les sorties).

```
cConversion est une Classe

      Privé
      // Pour usage interne de la classe,
      // les membres privés n'étant pas
      // vues de l'extérieur
      TableChaine est un Tableau de 10 chaînes

      Public
      // Utilisable partout dans la classe
      Resultat est une chaîne

FIN
```

Un petit commentaire, la classe se divise en deux parties :

- Une zone public qui contient un membre (une donnée) que va renvoyer l'objet, le résultat de la conversion.
- Une zone privée qui contient un membre (une donnée) qui ne sortira pas de l'objet, une table de chaînes qui contient la correspondance chiffre/lettre.

Créons maintenant la méthode *Constructeur*. Bien que celle-ci soit automatiquement créée, nous devons initialiser la variable `TableChaine` avec les valeurs en lettres. Pour le test de validité du nombre entrée, nous devons passer ce nombre en paramètre dans la méthode.

```
PROCEDURE cConversion::Constructeur(Nombre)

SI Nombre <> 0 ALORS // test de validité

      :TableChaine[ 1 ] = "un"
      :TableChaine[ 2 ] = "deux"
      :TableChaine[ 3 ] = "trois"
      ...
      :TableChaine[ 9 ] = "neuf"
      :TableChaine[10 ] = "dix"

SINON

      :Resultat = "zéro"

FIN // du test
```

A noter :

Notons que toutes les variables d'une classe doivent être précédées du symbole **:** (deux points) lorsqu'elles sont utilisées dans les méthodes.

Créons enfin la méthode *Calcule*. Elle va permettre de faire la correspondance entre la donnée chiffre et la donnée lettre.

```
PROCEDURE cConversion::Calcule(Nombre)

      :Resultat = :TableChaine[Nombre]
```

Voilà ! Il ne reste plus qu'à coder un bouton qui exécutera le traitement de conversion.

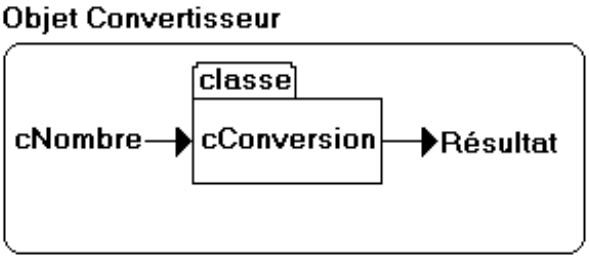
```
LeNombre est un cConversion(cNombre) // Appel de l'objet

// Envoi de la demande de conversion depuis
// le champ cNombre à l'objet
LeNombre:Calcule(cNombre)

// Récupération du résultat dans un champ nommé cLettre
cLettre = LeNombre:Resultat
```

Il n'y a plus qu'à tester et cela marche. Vous venez de créer un objet convertisseur sous la forme d'une classe appelé cConversion.

En entrée de l'objet se place le nombre. La classe le manipule à travers sa méthode et retourne le résultat en sortie :



Conversion complexe

Vous êtes en forme ? Allons plus loin en convertissant cette fois des nombres. En clair, je tape "1256" et l'application me renvoie "mille deux cent cinquante six".

L'approche est la même mais le tableau de conversion doit être ordonné de façon différente. En effet, il est hors de question de créer un tableau avec tous les nombres existants. On peut bien sûr simplifier car l'étendu des nombres donne deux combinaisons de chiffres possibles.

	1	2
1	Un	
2	Deux	Vingt
3	Trois	Trente
4	Quatre	Quarante
5	Cinq	Cinquante
6	Six	Soixante
7	Sept	Soixante
8	Huit	Quatre-vingt
9	Neuf	Quatre-vingt
10	Dix	
11	Onze	
12	Douze	
13	Treize	
14	Quatorze	
15	Quinze	
16	Seize	
17	Dix-sept	
18	Dix-huit	
19	Dix-neuf	

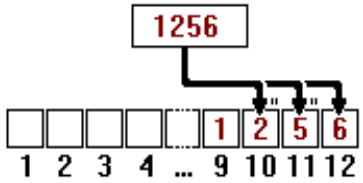
Pour nos amis belges et suisses, vous rectifierez vous mêmes le tableau avec vos *septantes* et *nonantes* ☺.

Traduire un nombre en lettres n'est ainsi qu'une combinaison de ces valeurs à l'exception de quelques cas qui sont décrits ci-dessous.

Cas particulier à gérer :

1. Les nombres dans les tranches 70 et 90 qui se combinent avec les valeurs dix à dix-neuf.
2. Les nombres dont l'unité est 1 et qui s'écrit en ajoutant "et" (vingt et un).
3. Le chiffre zéro
4. Les nombres démarrant par million ou milliard qui doivent commencer par un à la différence de cent ou mille.

Pour utiliser ce tableau de traduction, nous devons également découper le chiffre en nombres autonomes dans un second tableau de caractères.



Notons que nous nous arrêterons à des chiffres à neuf positions, ce qui nous fait quand même des traductions de chiffres d'un maximum d'un milliard.

Procédons maintenant à la création de la classe que nous appellerons *cConversion*. Oui je sais, c'est le même nom que l'exercice précédent. Aussi je vous conseille de créer un nouveau projet. Initialisons les valeurs de cette classe. Deux membres privés sont déclarés pour la communication entre les méthodes. Seul le membre *enLettres* qui renvoie le chiffre en lettres est public.

```
cConversion est une Classe

      Privé
      tNombre est un tableau de 9 caractères
      tTraduc est un tableau de 2 par 19 chaînes


      Public
      enLettres est une chaîne

FIN
```

Construisons maintenant les tableaux de traduction et de stockage. Reportez vous plus haut pour les valeurs données. Afin de test ultérieur, on initialise le tableau *tNombre* avec le caractère tiret (-).

```
FONCTION cConversion::Constructeur()

Ind est un entier court

Pour Ind = 1 à 9
      :tNombre[ Ind ] = "-"
```

```
FIN

:tTraduc[ 1, 1 ] = " un"
:tTraduc[ 1, 2 ] = " deux"
      ...
:tTraduc[ 1, 19] = " dix-neuf"
:tTraduc[ 2, 1 ] = " "
:tTraduc[ 2, 2 ] = " vingt"
:tTraduc[ 2, 3 ] = " trente"
      ...
:tTraduc[ 2, 19 ] = " zéro"
```

Rappel :

Surtout n'oubliez pas les fameux deux-points (**:**) à chaque déclaration des membres dans une méthode.

Créons une méthode *Stocke* pour stocker notre nombre dans le tableau *tNombre*. Cette méthode transforme d'abord le nombre en chaîne puis transfert chaque chiffre dans une case du tableau *tNombre*. Notons que cette fonction est privé et n'est pas visible depuis l'extérieur de la classe.

```
FONCTION PRIVE cConversion::Stocke(Nombre)

i, j, k, l sont des entiers courts
NombreChaine est une chaîne

NombreChaine = VersChaine(Nombre)
j = taille(NombreChaine)
k = j
l = 9

POUR i = 1 à k

      :tNombre[l] = Milieu(NombreChaine, j, 1)
      j--
      l--

FIN
```

Créons maintenant une méthode *MembreGauche* qui va nous permettre de renvoyer dans le membre *enLettres*, le contenu de la colonne gauche du tableau *tTraduc*.

```
FONCTION PRIVE cConversion::MembreGauche(indice)

// Attention il faut que l'indice soit supérieur
// à zéro, le tableau commencant à l'indice 1
SI indice > 0 ALORS
      :EnLettres += :tTraduc[1,indice]
FIN
```

Astuce :
Le symbole += qui permet de réduire la ligne.
Equivalent à :
`:EnLettres = :EnLettres + :tTraduc[1,indice]`

Créons une seconde méthode *MembreDroit* pour le renvoi cette fois du contenu de la colonne de droite du tableau *tTraduc* dans le membre *enLettres*.

```
FONCTION PRIVE cConversion::MembreDroit(indice)

:EnLettres += :tTraduc[2,indice]
```

Nous allons maintenant écrire la méthode *Dizaine* qui va analyser les dizaines et les unités du chiffre. Cette méthode se charge ensuite d'appeler les méthodes *MembreGauche* et *MembreDroit*. Cette méthode se charge des cas particuliers 1 et 2 (voir tableau plus haut).

```
PROCEDURE PRIVE cConversion::Dizaine(i)

// Analyse de l'unité et de la dizaine

// Le chiffre est entre 0 et 9
SI :tNombre[i] = "-" et :tNombre[i+1] <> "-" ALORS
      :MembreGauche(:tNombre[i+1])
FIN
SI :tNombre[i] = 0 ALORS :MembreGauche(:tNombre[i+1])

// Le chiffre est entre 10 et 19
SI :tNombre[i] = 1 ALORS :MembreGauche(val(:tNombre[i+1])+10)

// Le chiffre est supérieur ou égal à 20
SI :tNombre[i] > 1 ALORS
      :MembreDroit(:tNombre[i])
      // Exception pour 70 et 90
      SI :tNombre[i] = 7 ou :tNombre[i] = 9 ALORS
            :MembreGauche(val(:tNombre[i+1])+10)
      SINON
            // Exception pour la dizaine "et" un
            // par exemple : vingt et un
            SI :tNombre[i+1] = 1 ALORS :EnLettres += " et"
            :MembreGauche(:tNombre[i+1])
      FIN
FIN
```

Nous allons procéder à la même chose pour la méthode *Centaine* qui analyse le chiffre sur cette partie. Cette méthode appelle également les méthodes *MembreGauche* et *MembreDroit*. Elle gère l'exception sur le chiffre 1 qui ne doit pas générer la chaîne "un cent".

```
PROCEDURE PRIVE cConversion::Centaine(i)

// Analyse de la centaine
SI :tNombre[i] = 1 ALORS :EnLettres += " cent"
SI :tNombre[i] > 1 ALORS
      :MembreGauche(:tNombre[i])
      :EnLettres += " cent"
FIN
```

Enfin (ouf !), nous allons créer la méthode *Calcule* qui va appeler la méthode *Stocke* puis successivement les méthodes *Centaine* et *Dizaine* pour la traduction du chiffre pour les tranches "million", "millier" et "centaine". Notons qu'il y a deux exceptions d'incluses pour orthographier correctement "million" et ne pas indiquer "mille" dans certains cas.

```
FONCTION cConversion::Calcule(Nombre)

:Stocke(Nombre) // Méthode stockage du nombre

// Traitement du chiffre à partir de 1.000.000
:Centaine(1)
:Dizaine(2)
// A inscrire si le chiffre est au-dessus de 9.999
Si :tNombre[3]<> "-" Alors
      // Traitement singulier pluriel du mot million
      Si :tNombre[3]=1 et :tNombre[2]="-" Alors
            :EnLettres += " million"
      Sinon
            :EnLettres += " millions"
      Fin
Fin

// Traitement du chiffre en dessous de 1.000.000
Si pas (:tNombre[4] = 0 et :tNombre[5] = 0 et ...
       :tNombre[6]= 0 ) Alors
      :Centaine(4)
      // Exception pour le chiffre 1000
      //si pas (:tNombre[5] = "-" et :tNombre[6] = 1) alors
      Si pas (:tNombre[6] = 1) Alors
            :Dizaine(5)
      Fin
      // A inscrire si le chiffre est au-dessus de 999
      Si :tNombre[6]<> "-" Alors :EnLettres += " mille"
Fin

// Traitement du chiffre en dessous de 1.000
:Centaine(7)
:Dizaine(8)
```

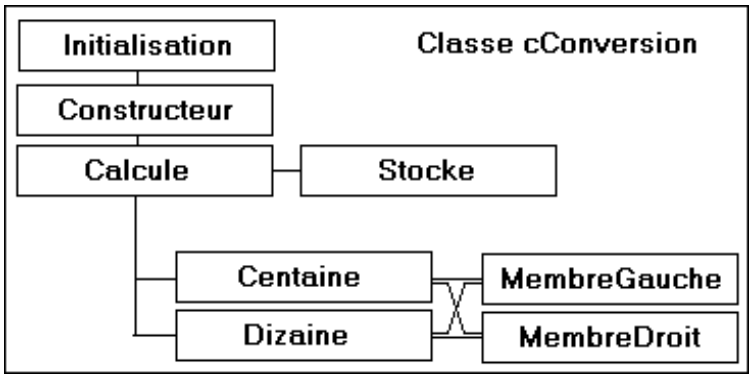
Codons maintenant pour utiliser notre classe dans une fenêtre. Insérer ainsi ce code dans le traitement clic d'un bouton de votre choix.

```
LeNombre est un cConversion() // Appel de l'objet

// Envoi de la demande de conversion depuis
// le champ cNombre à l'objet
LeNombre:Calcule(cNombre)

// Récupération du résultat dans un champ texte nommé cInfo
cInfo = LeNombre:EnLettres
```

Résumons maintenant l'ensemble des méthodes de la classe pour bien comprendre comment s'articule l'objet Convertisseur.



Terminons par écrire une petite fiche descriptive de notre classe et son mode d'emploi.

Nom du fichier physique	CONVERS.WDC
Nom de la classe	CConversion
Description	Permet la conversion d'un chiffre en lettres (maximum 9 positions)
Déclaration des objets	`n est un cConversion`
Envoi des données	`Nbre est un entier long` `n:Calcule(Nbre)`
Réception des données	`Ch est une chaîne` `Ch = n:EnLettres`

Conclusion

Bien sûr vous allez me dire que vous auriez pu faire la même chose avec des fonctions et des procédures classiques. Tout à fait exact car il n'y a pas de différence au niveau du code entre une fonction et une méthode. Alors où est l'avantage d'une programmation objet ?

Imaginons que demain, vous ayez à faire une application qui édite des chèques. Vous aurez alors grand besoin du code que vous avez placé dans votre programme. L'avantage dans la POO, c'est la réutilisation immédiate de ce code qui je vous le rappelle est dans un fichier WDC. Il vous suffira de récupérer cet objet et de l'utiliser comme bon vous semble.

Important :

Pensez à bien documenter vos classes en indiquant les entrées et les sorties ainsi que les méthodes publics utilisables et leurs syntaxes.

Un exemple : le jeu de poker

Principe

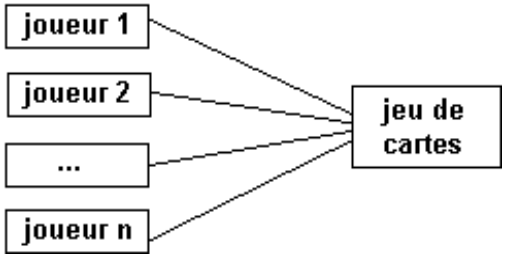
Voyons cela sur un exemple concret tel qu'un programme permettant de jouer au Poker. Un petit rappel des règles pour y jouer, il vous faut un jeu de 52 cartes et être au minimum deux joueurs. Il est distribué àchaque donne cinq cartes par joueur. Les joueurs peuvent changer un maximum de 4 cartes pour améliorer leur donne. Le joueur ayant la combinaison de cartes la plus forte gagne la partie.

Voici ci-dessous les combinaisons possibles détaillées par ordre décroissant :

Rang	Combinaison	Description
1	Quinte royale	As, Roi, Dame, Valet, 10 de la même couleur
2	Quinte flush	Suite de 5 cartes de même couleur
3	Carré	4 cartes de même valeur
4	Full	Un brelan plus une paire
5	Couleur	5 cartes de même couleur qui ne se suivent pas
6	Quinte	Suite de 5 cartes qui ne sont pas de même couleur
7	Brelan	3 cartes de même valeur
8	Deux paires	Comme son nom l'indique
9	Paire	2 cartes de même valeur

Les objets

Après cet énoncé, nous allons tenter de traduire en modèle objet le jeu de Poker. Nous allons d'abord sortir les objets principaux.



On peut déduire qu'il nous faut un objet "jeu de cartes" et des objets "joueur". L'objet "jeu de cartes" sera nommé `JeuCarte` (sans s pour être sur 8 caractères). Les objets "joueur" peuvent être rassemblés en un seul objet appelé `Joueur`. En effet, quel que soit le nombre de joueurs, chacun d'eux aura les mêmes propriétés. Pour compliquer notre exemple, un des joueurs sera pris en charge par l'ordinateur.

Les méthodes

Déduisons maintenant les méthodes qui vont interagir entre les objets.

`JeuCarte`	
Méthode	**Description**
`Constructeur`	Création du jeu de 52 cartes
`DonneCarte`	Permet de donner une carte du jeu au hasard
`MelangeJeu`	Remet les cartes dans le paquet àla fin de la partie

`Joueur`	
Méthode	**Description**
`Constructeur`	Non nécessaire
`PrendCarte`	Prend une carte que donne le jeu et la place dans la donne
`CacheCarte`	Prend une carte que donne le jeu mais la cache dans la donne
`TrieCarte`	Trie les cartes de la donne
`AfficheCarte`	Affiche la carte placée dans la donne
`ChangeCarte`	Change une carte dans la donne et la rend au jeu
`CalculeDonne`	Calcule la combinaison la plus forte de la donne

Important :

On voit ici l'importance de donner des noms clairs à chaque méthode. Ceux-ci permettront de coder plus clairement par la suite.

Programmation

Commençons par la déclaration de la classe `JeuCarte`. Nous allons comme nous l'avons déjà dit plus haut, c'est dans ce traitement que nous devons déclarer toutes les variables utilisées.

```
JeuCarte est une classe

      PRIVE
      //--------------------------------------
      Paquet est un tableau de 52 par 2 chaînes
      NumCarte est un entier court

      PUBLIC
      //--------------------------------------
      NomCarte est une chaine

FIN
```

Nous reviendrons sur chacune de ces variables dans les méthodes auxquelles elles sont affectées. Passons maintenant au constructeur de la classe.

```
PROCEDURE JeuCarte::Constructeur()
// Création du paquet de 52 cartes
PRIVE
i est un entier
// Pour la question de la valeur des cartes,
// on démarre au 2 et on monte jusqu'à l'as (14)
:NumCarte = 2
Pour i = 1 à 13
      // Pique
      :Paquet[i,1] = VersChaîne((:NumCarte),"02d") + "PQE"
      :NumCarte++
Fin
:NumCarte = 2
Pour i = 14 à 26
      // Trefle
      :Paquet[i,1] = VersChaîne((:NumCarte),"02d") + "TRE"
      :NumCarte++
Fin
:NumCarte = 2
Pour i = 27 à 39
      // Carreau
      :Paquet[i,1] = VersChaîne((:NumCarte),"02d") + "CAR"
      :NumCarte++
Fin
:NumCarte = 2
Pour i = 40 à 52
      // Coeur
      :Paquet[i,1] = VersChaîne((:NumCarte),"02d") + "COR"
      :NumCarte++
Fin
:MelangeJeu()
```

On crée chaque famille (pique, trèfle, carreau et cœur) dans un tableau de chaînes. Pour un calcul plus aisé des points, on démarre du 2 jusqu'au 10, puis à partir du valet à l'as, on numérote de 11 à 14. On formate ce numéro sur 2 positions avant de la mettre dans la chaîne et d'y ajouter l'abréviation de la famille sur 3 lettres. A la fin du constructeur, on exécute la méthode `MelangeJeu`.

Codons cette méthode. Elle met toutes les chaînes de la seconde colonne avec la valeur "**L**" pour dire que la carte est libre.

```
FONCTION PUBLIC MelangeJeu()
i est un entier
Pour i = 1 à 52
      :Paquet[i,2] = "L"
FIN
```

La méthode `DonneCarte` permet de renvoyer une carte quelconque depuis le jeu de 52 cartes.

```
FONCTION PUBLIC DonneCarte()
InitHasard()
:NumCarte = Hasard(1,52)
Boucle // infinie
      SI :Paquet[:NumCarte,2] = "L" ALORS
            // La carte est maintenant prise
            :Paquet[:NumCarte,2] = "P"
            // Renvoyons cette carte
            :NomCarte = :Paquet[:NumCarte,1]
            SORTIR
      FIN
      :NumCarte = Hasard(1,52)
FIN
RENVOYER :NomCarte
```

Dans cette méthode, on tire une carte au hasard dans le jeu de 52 cartes. On interroge la seconde colonne pour savoir si elle est libre. Si oui, on met "**P**" (pour prise) afin de ne plus pouvoir y accéder par la suite.

Rappel :

N'oubliez pas la commande `InitHasard` sans quoi votre classe renverra toujours la même carte (vous voulez tricher ou quoi ☺).

Passons maintenant au second objet qui permettra de gérer un ou plusieurs joueurs avec la déclaration de cette classe.

```
Joueur est une classe
      PRIVE
      // ---------------
      CarteEnMain est un tableau de 5 chaînes

      PUBLIC
      // ---------------
      ImgCarte est une chaîne
      Score est un entier
      Val1, Val2 sont des entiers
FIN
```

Il n'y a pas de constructeur pour cette classe. Passons alors à la méthode `AfficheCarte` qui permet d'envoyer l'image de la carte (fichier au format BMP) à l'interface.

```
FONCTION PUBLIC AfficheCarte(Num)
:ImgCarte = :CarteEnMain[Num]
RENVOYER :ImgCarte + ".BMP"
```

La méthode `PrendCarte` permet de comptabiliser une carte envoyée par la classe `JeuCarte` et de renvoyer l'image de cette carte (fichier au format BMP).

```
FONCTION PUBLIC PrendCarte(Num, NomCarte)

:CarteEnMain[Num] = NomCarte
:ImgCarte = (:CarteEnMain[Num])+".BMP"

RENVOYER :ImgCarte
```

La méthode `CacheCarte` ressemble à la précédente mais elle est là pour le joueur que gère l'ordinateur. Elle permet de comptabiliser une carte que donne la classe `JeuCarte` mais elle renvoie l'image du dos d'une carte.

```
FONCTION PUBLIC CacheCarte(Num, NomCarte)
:CarteEnMain[Num] = NomCarte
SI Num = 5 ALORS
      :TrieCarte()
      :Score = :CalculeCarte()
FIN
RENVOYER "DOS.BMP"
```

En revanche lorsque la donne est complète et qu'elle a atteint la cinquième carte, la méthode fait appel alors à sa sœur `TrieCarte` pour trier les cartes dans l'ordre croissant (du 02 à l'as) afin de faciliter le calcul.

```
PROCEDURE PUBLIC TrieCarte()
i     est un entier court
Tmp   est une chaine
TANTQUE 1 // Boucle infinie
      POUR i = 1 à 4
            SI :CarteEnMain[i] > :CarteEnMain[i + 1] ALORS
                  Tmp = :CarteEnMain[i]
                  :CarteEnMain[i] = :CarteEnMain[i + 1]
                  :CarteEnMain[i + 1] = Tmp
            FIN
      FIN
      SI (:CarteEnMain[1] < :CarteEnMain[2]) et ...
            (:CarteEnMain[2] < :CarteEnMain[3]) et ...
            (:CarteEnMain[3] < :CarteEnMain[4]) et ...
            (:CarteEnMain[4] < :CarteEnMain[5]) ALORS SORTIR
FIN
:Score = :CalculeDonne()
```

A la fin de cette méthode, on en appelle à nouveau une autre afin de calculer les combinaisons gagnantes de la donne.

```
FONCTION PRIVE CalculeDonne()

i       est un entier court
j       est un entier court
Coul    est un entier court = 1
Fam     est un entier court = 1
Suite   est un entier court = 1
ValeurA est un entier court = 1
ValeurB est un entier court = 1
c       est une chaine
g, d    sont des entiers courts

// Toutes les cartes sont de la même famille
c = :CarteEnMain[1][[3 à 5]]
POUR i = 2 à 5
      SI :CarteEnMain[i][[3 à 5]] = c ALORS Fam++
FIN

// Est une suite ?
POUR i = 1 à 4
      g = val(:CarteEnMain[i][[1 à 2]])
      d = val(:CarteEnMain[i+1][[1 à 2]])
      SI (g + 1) = d ALORS Suite++
FIN

// Les cartes sont-elles toutes de la même couleur ?
c = :CarteEnMain[1][[5]]
POUR i = 2 à 5
      SI :CarteEnMain[i][[5]] = c ALORS Coul++
FIN

// Nombre de cartes de même valeur
POUR i = 1 à 4
      g = Val(:CarteEnMain[1][[1 à 2]])
      d = Val(:CarteEnMain[i+1][[1 à 2]])
      SI g = d ALORS
            ValeurA++
            :Val1 = g
      FIN
FIN
SI ValeurA = 1 ALORS
      POUR i = 2 à 4
            g = Val(:CarteEnMain[2][[1 à 2]])
            d = Val(:CarteEnMain[i+1][[1 à 2]])
            SI g = d ALORS
                  ValeurA++
                  :Val1 = g
            FIN
      FIN
FIN
j = ValeurA + 1
```

```
// Analysons le 2e groupe de cartes
POUR i = j à 4
      g = Val(:CarteEnMain[j][[1 à 2]])
      d = Val(:CarteEnMain[i+1][[1 à 2]])
      SI g = d ALORS
            ValeurB++
            :Val2 = g
      FIN
FIN
SI ValeurB = 1 ALORS
      pour i = (j + 1) à 4
            g = Val(:CarteEnMain[j+1][[1 à 2]])
            d = Val(:CarteEnMain[i+1][[1 à 2]])
            SI g = d ALORS
                  ValeurB++
                  :Val2 = g
            FIN
      FIN
FIN
// Fin de l'analyse
SI Suite = 5 ALORS
      SI Fam = 5 ALORS RENVOYER 8 SINON RENVOYER 7
FIN
SI Coul = 5 ALORS RENVOYER 6
SI ValeurA = 4 ou ValeurB = 4 ALORS RENVOYER 5
SI ValeurA = 3 ou ValeurB = 3 ALORS
      SI ValeurA = 2 ou ValeurB = 2 ALORS
            RENVOYER 4
      SINON
            RENVOYER 3
      FIN
FIN
SI ValeurA = 2 et ValeurB = 2 ALORS RENVOYER 2
SI ValeurA = 2 ou ValeurB = 2 ALORS RENVOYER 1
RENVOYER 0
```

Astuce :

On notera la syntaxe particulière des double crochets `[[]]` qui permettent l'extraction de caractères dans une chaîne (voir l'aide du *W-Langage*).

Cette méthode est la plus complexe. C'est donc celle-ci qui est à améliorer, j'attends vos suggestions. Cette méthode retourne un entier qui est stocké dans la variable `Score`. Pour consulter ce score, on fait appelle à une nouvelle méthode codée ci-dessous.

```
FONCTION PUBLIC DonneScore()
RENVOYER :Score
```

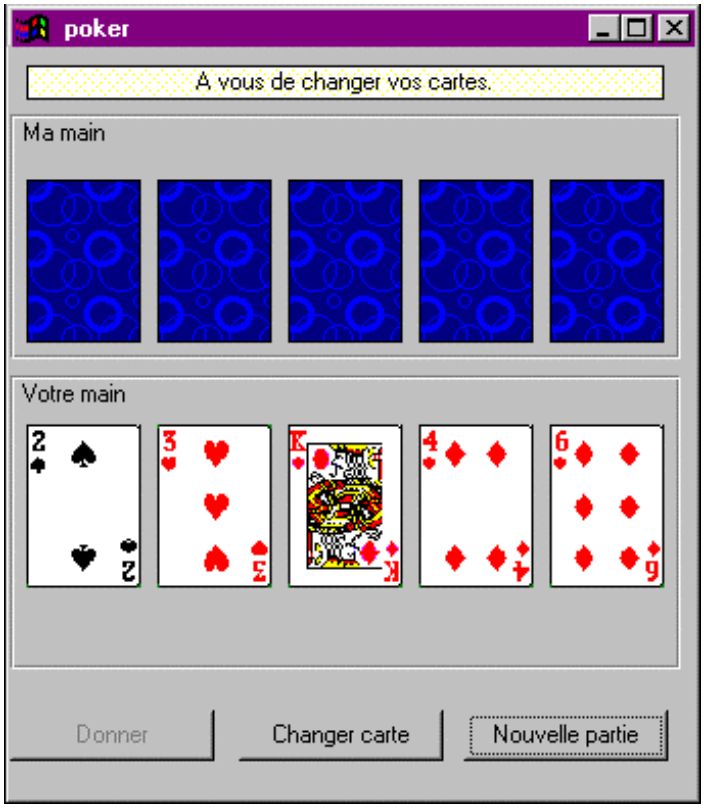
Enfin, le cas d'égalité (si deux joueurs ont la même combinaison), la méthode renvoie donc la valeur de la carte la plus forte.

```
FONCTION PUBLIC ValeurCarte()
SI :Val1 > :Val2 ALORS
      RENVOYER :Val1
SINON
      RENVOYER :Val2
FIN
```

Ouf ! C'est terminé.

Interfaçage

Les classes codées, il faut maintenant interfacer graphiquement celles-ci avec vos fenêtres, champs et autres boutons. Voici un exemple d'application qu'il est possible de réaliser. Celle-ci permet l'affrontement entre l'ordinateur et un joueur humain.



Dans cette fenêtre, nous allons décrire les champs du haut vers le bas :

Description	Nom des champs
Une zone libellé sur fond jaune pour informer le joueur du déroulement de la partie.	`ZINFO`
Cinq champs image pour afficher les cartes de l'ordinateur.	`CO1 à CO5`
Cinq zones de clic pour afficher les cartes du joueur humain et lui permettre par la suite de désigner les cartes à changer.	`CJ1 à CJ5`

Trois boutons texte (Donner, Changer carte et Nouvelle partie).	`XDONNE`
	`XCHANGE`
	`XNOUVEAU`

Nous avons en plus, cinq champs images qui sont au départ cachés et qui apparaîtront sur chacune des cartes pour symboliser un changement. Ainsi chaque fois que le joueur clique sur une carte, ce champ apparaît sous la forme d'une croix ou disparaît. Les champs sont nommées `CHG1` à `CHG5`.

Au début de la partie, les champs `CO1` à `CO5`, `CJ1` à `CJ5` sont vides. Les boutons `XDONNE` et `XCHANGE` sont grisés. Le joueur ne peut donc que cliquer sur le bouton `XNOUVEAU`.

Déclarons au début les variables dont nous aurons besoin ainsi que les objets de nos classes. Ce code est à insérer à l'ouverture de la fenêtre.

```
PUBLIC
Echange     est un entier court
j           est un objet jeuCarte
mOrdi       est un objet Joueur
mVous       est un objet Joueur
```

On notera la déclaration d'une variable `Echange` pour comptabiliser le nombre de cartes que le joueur ne peut excéder soit cinq cartes.

Le code du bouton `XNOUVEAU` permet d'avoir un état initial pour une nouvelle partie.

```
j:MelangeJeu()
Echange = 0
CJ1 = "" ; CO1 = ""
CJ2 = "" ; CO2 = ""
CJ3 = "" ; CO3 = ""
CJ4 = "" ; CO4 = ""
CJ5 = "" ; CO5 = ""
xDonne..etat   =  Actif
zInfo..libellé = "Cliquez sur le bouton Donner pour jouer !"
```

Le code du bouton `XDONNE` :

```
// Servons les joueurs
cj1 = mVous:PrendCarte(1, j:donneCarte() )
co1 = mOrdi:CacheCarte(1, j:donneCarte() )
cj2 = mVous:PrendCarte(2, j:donneCarte() )
co2 = mOrdi:CacheCarte(2, j:donneCarte() )
Cj3 = mVous:PrendCarte(3, j:donneCarte() )
co3 = mOrdi:CacheCarte(3, j:donneCarte() )
CJ4 = mVous:PrendCarte(4, j:donneCarte() )
co4 = mOrdi:CacheCarte(4, j:donneCarte() )
CJ5 = mVous:PrendCarte(5, j:donneCarte() )
co5 = mOrdi:CacheCarte(5, j:donneCarte() )
// Fin de service
xDonne..etat   =  Grise
xChange..etat  =  Actif
zInfo..libelle = "A vous de changer vos cartes."
```

Examinons d'un peu plus près la méthode d'échange pour le service des cartes. D'abord, la méthode `DonneCarte()` est appelée. Elle renvoie une carte qui est récupérée dans la méthode `PrendCarte()` ou `CacheCarte()` comme second argument. Les méthodes `PrendCarte()` et `CacheCarte()` prennent en effet l'emplacement de la carte dans la donne comme premier argument et le nom de la carte en second. Celles-ci renvoient enfin, l'image BITMAP de la carte à placer dans le champ correspondant (image pour l'ordinateur et clic pour le joueur).

Maintenant, les cartes sont distribuées. Seules, celles du joueur humain sont visibles. A ce stade, on peut changer un maximum de 4 cartes pour améliorer les combinaisons de la donne. Pour des raisons de complexité, l'ordinateur ne change pas ses cartes. Voici le code des zones de clic pour le changement des cartes du joueur humain.

Clic sur bouton `CJ1`

```
SI Chg1..etat = actif ALORS
      Chg1..etat = Invisible
      Echange--
SINON
      Chg1..etat = Actif
      Echange++
FIN
```

Clic sur bouton `CJ2`

```
SI Chg2..etat = actif ALORS
      Chg2..etat = Invisible
      Echange--
SINON
      Chg2..etat = Actif
      Echange++
FIN
```

etc.

Enfin le traitement du clic sur le bouton `XCHANGE` qui permet de terminer la partie et de savoir qui a gagné.

```
SI Echange > 4 ALORS
      Erreur("Vous ne pouvez pas échanger toutes vos cartes")
      RETOUR
FIN
SI Echange > 0 ALORS
      SI OuiNon(Non,"Changer "+Echange+" cartes ?")=Non ALORS RETOUR
FIN
// Changement des cartes
SI Chg1..etat = actif ALORS
      cj1 = mVous:PrendCarte(1, j:donneCarte() )
      Chg1..etat = invisible
FIN
SI Chg2..etat = actif ALORS
      cj2 = mVous:PrendCarte(2, j:donneCarte() )
      Chg2..etat = invisible
FIN
SI Chg3..etat = actif ALORS
      cj3 = mVous:PrendCarte(3, j:donneCarte() )
      Chg3..etat = invisible
FIN
```

```
SI Chg4..etat = actif ALORS
      cj4 = mVous:PrendCarte(4, j:donneCarte() )
      Chg4..etat = invisible
FIN
SI Chg5..etat = actif ALORS
      cj5 = mVous:PrendCarte(5, j:donneCarte() )
      Chg5..etat = invisible
FIN
// Fin du changement des cartes
xChange..Etat = grise
mVous:TrieCarte()
// Affiche maintenant le jeu de l'ordinateur
co1 = mOrdi:AfficheCarte(1)
co2 = mOrdi:AfficheCarte(2)
co3 = mOrdi:AfficheCarte(3)
co4 = mOrdi:AfficheCarte(4)
co5 = mOrdi:AfficheCarte(5)
// Qui a gagné ?
SI mVous:DonneScore() < mOrdi:DonneScore() ALORS
      zInfo..libelle = "J'ai gagné !"
FIN
SI mVous:DonneScore() > mOrdi:DonneScore() ALORS
      zInfo..libelle = "Vous avez gagné !"
FIN
// En cas d'égalité...
SI mVous:DonneScore() = mOrdi:DonneScore() ALORS
      SI mVous:ValeurCarte() < mOrdi:ValeurCarte() ALORS
          zInfo..libelle = "J'ai gagné !"
      FIN
      SI mVous:ValeurCarte() > mOrdi:ValeurCarte() ALORS
          zInfo..libelle = "Vous avez gagné !"
      FIN
FIN
```

Voilà, vous venez de réaliser une application de jeu de poker en POO. Bien sûr, le code est perfectible et il faut l'améliorer. Je compte donc sur vous…

Méthodologie objet

Avant de vous lancer dans la programmation objet à corps perdu, je vous conseille vivement de lire des livres sur cette approche. Je citerais la méthode OMT de J. Rumbaugh; l'OOD de G. Booch et plus récemment le langage de modélisation UML (Unified Modeling Language).

Principe

L'approche objet découle d'une modélisation. Celle-ci est établi avant toute réalisation et programmation pour plusieurs objectifs :

- comprendre le fonctionnement du futur système informatique ;
- en maîtriser la complexité ;
- en assurer la cohérence ;
- pouvoir faire communiquer les différents modules.

Conclusion

Vous voyez maintenant la puissance que l'on peut tirer de la programmation en objets. La réutilisation des classes, l'économie du code et la clarté des programmes en font toute la puissance.

Bonne chance !

Copyright

Dernière date de révision : 20 mai 2003

La programmation

en

client/serveur

sous

Windev®

Sommaire

Préambule

Le présent document ne propose qu'une approche de ce domaine vaste qu'est la programmation en client/serveur. Utilisant ce mode depuis maintenant plus de cinq ans, je me suis permets d'en coucher les grandes lignes et de faire profiter de mon expérience à d'autres. Je ne me prétends pas un spécialiste mais j'en ai appris certains aspects en travaillant sur plusieurs projets importants, notamment sur un logiciel de gestion d'appels (couramment appelé par l'anglicisme helpdesk). Pour moi, le mode C/S représente la solution la plus souple en matière de développement d'application lourde multi-utilisateurs. J'espère que vous y trouverez les réponses dont vous avez besoin.

Dans le document, j'utilise indifféremment les termes client/serveur ou C/S et système de gestion de base de données ou SGBD.

Introduction

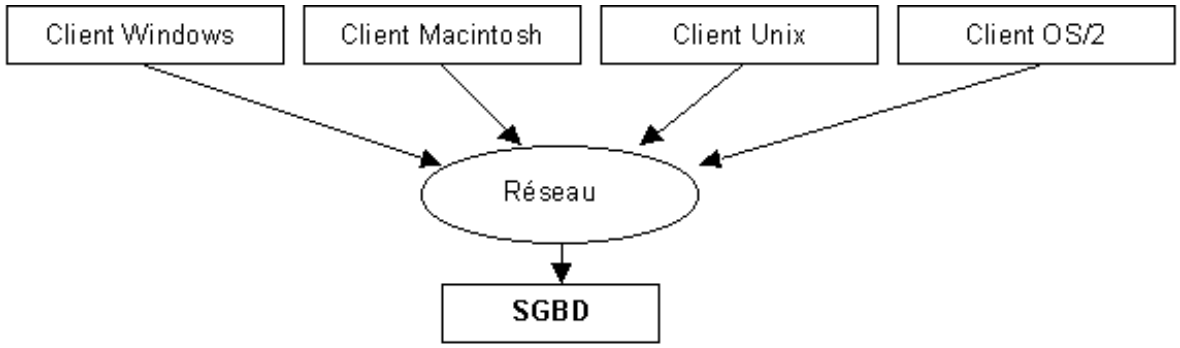
Afin de rester le plus simple possible, je décrirai dans ce document, une architecture en client/serveur avec le système de gestion de base de données de Microsoft Access. En effet, Access est certainement le SGBD le plus simple à mettre en place puisqu'il ne nécessite pas de serveur dédié et qu'il peut être installé sur la même machine que le programme client. De plus, le pilote est fourni est standard sous Windows, il n'est donc même pas indispensable d'avoir Access présent sur l'ordinateur.

Bien sûr, ce n'est donc pas véritablement du C/S mais les mécanismes sont identiques que pour d'autres SGBD du marché tels que Oracle, Sybase ou SQL/Server. D'autre part Access est un SGBD mono-utilisateur, il n'est donc pas fait initialement pour fonctionner en réseau. Il ne dispose d'ailleurs pas de fonctions de blocage d'enregistrement. Néanmoins, il est possible par certaines astuces d'y remédier (voir au chapitre : Limite d'Access).

Généralités

Le client/serveur permet d'avoir une séparation distincte entre l'application et les données. Cela présente certains avantages comme le développement multi-plateformes et l'indépendance du système d'exploitation aussi bien pour le client que pour le serveur.

Ainsi les applications clientes peuvent fonctionner sous Windows, Macintosh ou Unix et le système de gestion de base de données peut être sur n'importe quel type de serveur : Windows NT, Unix, AS400, Novell, etc.



Les fonctions du W-Langage

Windev propose deux familles de fonction permettant de gérer des programmes en C/S :

- les fonctions dont le nom commence par "od"
- les fonctions dont le nom commence par "SQL"

Les fonctions "od" sont encore là pour des raisons de compatibilité avec les versions antérieurs de Windev. PCSOFT indique d'ailleurs dans sa documentation qu'il est recommandé d'utiliser les fonctions "SQL" qui sont beaucoup plus rapides.

J'expose ci-dessous les principales fonctions que j'utilise dans la programmation C/S. Bien sûr, cette liste n'est pas complète et je vous renvoie à la documentation SQL de Windev (manuel de programmation) pour le complément des fonctions.

Fonction	**Description**
`SqlCol`	Récupère l'information contenu dans la colonne de résultat
`SqlConnecte`	Connexion à la base
`SqlDeconnecte`	Déconnexion de la base
`SqlExec`	Exécution d'une requête
`SqlFerme`	Ferme la requête
`SqlPositionne`	Se positionner directement sur une ligne du résultat de la requête.
`SqlPremier`	Se positionne sur la première ligne du résultat de la requête
`SqlSuivant`	Se positionne sur la ligne suivante du résultat de la requête
`SqlTable`	Transfert le résultat de la requête dans une table

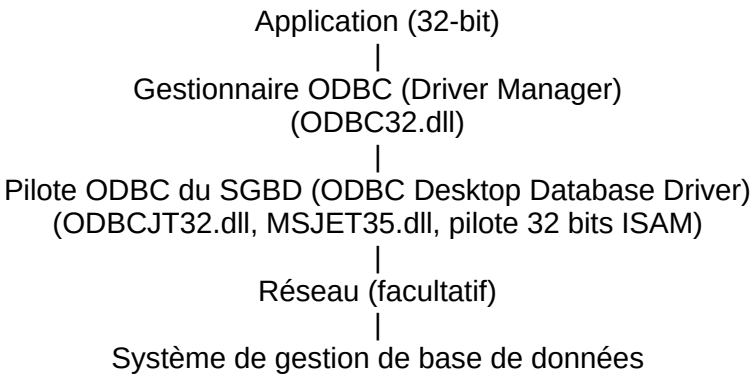
Pour les puristes, il est aussi possible de passer par l'API ODBC, en utilisant la fonction AppelDLL32. Mais on y retrouve exactement les mêmes fonctions et aucun gain en rapidité.

Architecture

Le pack de pilotes ODBC de Microsoft (ce pack est disponible au 01/02/00 à l'adresse internet http://support.microsoft.com/download/support/mslfiles/WX1350.exe) inclus les pilotes 32 bits nécessaire à la connexion pour les bases de données de Microsoft Access, dBASE, Microsoft Excel, Microsoft FoxPro, Borland Paradox et les fichiers texte.

En revanche, les pilotes en 16 bits ne sont pas fournis.

L'architecture C/S sous Windows 9x est la suivante :



Le client ODBC utilise un langage ou vocabulaire de commandes propres pour faire des requêtes sur des bases de données ou pour leur envoyer des informations (création, mise à jour ou suppression). Cependant, le SGBD ne comprend pas la requête faite par le client tant qu'elle n'a pas été passée par le pilote ODBC spécifique au SGBD. Le pilote est un logiciel présent aux côtés de l'application cliente. Il traduit la requête dans un langage compréhensible par le SGBD. De la même manière lorsque le SGBD envoie la réponse, le pilote ODBC traduit à nouveau pour l'application.

Vocabulaire

Je vous recommande la lecture d'un autre de mes supports *les systèmes de gestion de base de données* (disponible sur mon site http://beaussier.webjump.com) pour la compréhension de certains termes.

Cas d'école

Afin de partir sur un exemple concret, nous allons concevoir une application C/S assez simple : une gestion de location de K7 vidéo.

Dans l'analyse, nous déduirons la présence de trois tables. Pour des raisons de simplification, nous ne créerons aucune jointure entre les tables, ni aucun index.

Film	
NumFi	Numéro du film
NomFi	Titre du film
GenreFi	Genre du film
PrixFi	Prix de la location
DescFi	Résumé du film

Client	
NumCl	Numéro du client
NomCl	Nom du client
AdrCl	Adresse

Location	
NumFi	Numéro du film
NumCl	Numéro du client
DdebLo	Date de début de la location
HdebLo	Heure du début de la location
DfinLo	Date de fin de la location
HfinLo	Heure de fin de la location

Conception

La première chose consiste à concevoir la base de données. Grâce à l'ODBC, nous allons pouvoir tout créer par programmation depuis notre application.

```
// Indiquons le chemin de la future base
cBase est une chaîne
cBase = "c:\data\mabase.mdb"

hWnd est entier long
OdbcDSN est entier
szDriver est chaîne asciiz sur 512
szAttrib est chaîne asciiz sur 512
```

```
hWnd = Handle()
OdbcDSN = 4  // = ODBC_ADD_SYS_DSN pour ajouter un DSN système
szDriver = "Microsoft Access Driver (*. mdb)"

// pour créer une base pour la première fois
szAttrib = " CREATE_DB="+ cBase +...
                     " General" + caract(0) + caract(0)

ResCnx = AppelDll32("odbccp32", " SQLConfigDataSource", ...
     hWnd, OdbcDSN, &szDriver, &szAttrib)
Si ResCnx <> 1 Alors
     Erreur("Impossible de créer la base de données")
Fin

Req est une chaîne
Req = "HKEY_CURRENT_USER\Software\ODBC\ODBC.INI\MaBase"
RegistreCreeClé( Req)
// Récupère le chemin de Windows
AppelDLL32("kernel32","GetWindowsDirectoryA",&szDriver,260)
RegistreEcrit(Req,"Driver",szDriver+"\system\odbcjt32.dll")
//Récupère le chemin de la base
RegistreEcrit( Req,"DBQ",cBase)
RegistreEcrit( Req, "DriverId", 25)
RegistreEcrit( Req, "FIL", "MS Access;")
RegistreEcrit( Req, "UID", "")

// Création des tables
ResCnx = sqlconnecte("MaBase","","")
Si ResCnx = 0 Alors
     Erreur("Connexion à la base impossible")
Sinon
     // Création table Film
     Req = "CREATE TABLE Film(" + ...
               "NumFi     LONG, " + ...
               "NomFi     CHAR(100), " + ...
               "GenreFi   CHAR(100), " + ...
               "PrixFi    LONG, " + ...
               "DescFi    LONGTEXT );"
     ResExe = sqlExec(Req," TableFilm")
     Si pas ResExe Alors
          Erreur("Problème sur requête  TableFilm")
     Fin
     sqlFerme("TableFilm")
     // ---------------------------------------
     // Création table Client
     // etc ...
Fin
sqlDeconnecte()
```

Les premières lignes du code ci-dessus permettent d'abord de créer les clés dans la base de registre, cela est nécessaire pour la connexion au SGBD via le DSN (ordre `sqlConnecte` et `sqlDeconnecte`).

Ensuite, le programme crée la structure de la base, c'est un squelette vide de table. Enfin, le programme crée les tables. Pour des raisons de longueur, je n'ai écrit que la programmation de la première table, les suivantes étant du même tonneau, je vous laisse le soin de les écrire (il faut bien que vous travailliez un peu :o)).

Ca y est ! Votre système de gestion de base de données est prêt à fonctionner. Simple non ?

Le client

Il ne reste plus qu'à programmer l'application cliente, le plus gros du travail en somme. Mais avec Windev, cela ne devrait pas être trop compliqué…

La création

Continuons donc notre exemple de location de K7 en créant le code qui nous servira par exemple à enregistrer un film dans la base de données.

```
ResCnx = sqlConnecte("MaBase","","")
Si ResCnx = 0 Alors
      Erreur("Impossible d'accéder à la base")
Sinon
      Req = "SELECT MAX( NumFi) FROM Film ;"
      ResExe = sqlExec(Req, "MaxNum")
      Si pas ResExe Alors
            Erreur("Impossible d'exécuter  MaxNum")
      Sinon
            sqlPremier("MaxNum")
            NumFi = Val( sqlcol("MaxNum",1))+1
      Fin
      sqlFerme("MaxId")
      //-------------------------------------
      Req = "INSERT INTO Film( "+...
            "NumFi, NomFi, GenreFi, PrixFi, DescFi) "+ ...
            " VALUES(NumFi "+...
            ",'" + NomFi +...
            " ','"+ GenrFi +...
            " ', "+ PrixFi +...
            " ,'"+ DescFi ) ;"
      fin //
     ResExe = sqlExec(Req,"CreationFilm")
     Si pas ResExe Alors
            Erreur("Impossible d'exécuter  sql:CreationFilm")
     Fin
     sqlFerme("CreationFilm")
sqlDeconnecte()
```

On le voit, il faut impérativement connecter et déconnecter le client du SGBD à chaque requête. Cela sécurise la base en cas de déconnexion accidentelle (coupure de courant par exemple). Bien sûr dans le code ci-dessus, il y a deux requêtes successives. La déconnexion a donc lieu à la fin de la seconde requête. Remarquons que la première requête sert à récupérer le dernier numéro de film (le plus grand MAX). Ce numéro est ensuite incrémenté de un pour créer le nouveau film. Vous noterez enfin que les chaînes sont obligatoirement cernés de côte (') et que j'ai volontairement laissé un espace à la fin de chaque chaîne. En effet les SGBD ne comprennent pas une chaîne vide qu'il assimile à une double côte (").

La modification

Voici maintenant le code pour modifier les informations contenu dans la table Client. Imaginons que le client n° 254 ait changé de nom et d'adresse

```
ResCnx = sqlConnecte("MaBase","","")
Si ResCnx = 0 Alors
      Erreur("Impossible d'accéder à la base")
Sinon
      Req = "UPDATE Client SET "+...
            "NomCl = 'Nouveau nom', "+...
            "AdrCl = 'Nouvelle adresse' "+...
            "WHERE NumCl = 254 ;"

     //
     ResExe = sqlExec(Req,"MajClient")
     Si pas ResExe Alors
            Erreur("Impossible d'exécuter  sql: MajClient ")
     Fin
     sqlFerme("MajClient")
sqlDeconnecte()
```

Vous noterez qu'il faut bien préciser dans la clause `WHERE` le numéro de client. Sans quoi, vous mettrez à jour l'intégralité de la table avec les mêmes informations.

La recherche

Bien entendu, le plus important dans une application est de pouvoir extraire les données dans n'importe quel ordre et n'importe quel sens. Pour cela, le SQL se révèle être d'une grande souplesse. En voici quelques exemple :

Je souhaite connaître tous les genres de film et les mettre dans une liste.

```
ResCnx = sqlConnecte("MaBase","","")
Si ResCnx = 0 Alors
      Erreur("Impossible d'accéder à la base")
Sinon
      Req = "SELECT DISTINCT  GenreFi FROM Film ;"

      ResExe = sqlExec(Req,"ListeGenre")
      Si pas ResExe Alors
            Erreur("Impossible d'exécuter  sql:ListeGenre")
      Sinon
            SqlPremier("ListeGenre")
            TantQue pas sql.endehors
                  ListeAjoute("MaListe",sqlCol("ListeGenre",1)
                  SqlSuivant("ListeGenre")
            Fin //de TantQue
      Fin //de ResExe
      sqlFerme("ListeGenre")
Fin //de ResCnx
sqlDeconnecte()
```

Je souhaite connaître les noms des clients ayant loué des films pendant le mois de janvier ainsi que le nom des films et les mettre dans une table mémoire.

```
ResCnx = sqlConnecte("MaBase","","")
Si ResCnx = 0 Alors
```

```
      Erreur("Impossible d'accéder à la base")
Sinon
      Req = "SELECT NomCl, NomFi " + ...
                  "FROM Client C, Film F, Location L " + ...
                  "WHERE DdebLo >= '20000101' " + ...
                  "AND DfinLo <= '20000131' " + ...
                  "AND C.NumCl = L.NumCl " + ...
                  "AND F.NumFi = L.NumFi ; "
      //
      ResExe = sqlExec(Req,"LocJanv")
      Si pas ResExe Alors
            Erreur("Impossible d'exécuter  sql:LocJanv")
      Sinon
            SqlTable("MaTable","LocJanv")
      Fin //de ResExe
      sqlFerme("LocJanv")
Fin //de ResCnx
sqlDeconnecte()
```

Vous noterez que la table doit comporter le nombre exact de colonnes renvoyé par la requête. Dans notre exemple, la table `MaTable` devra donc avoir deux colonnes pour accueillir le nom du client et le nom du film.

Les outils

Bien sûr, connaître le SQL est assez facile, en revanche construire une requête peut parfois s'avérer ardu. C'est pourquoi, je vous recommande des petits logiciels de conception graphique de requête. Si vous avez installé sur votre ordinateur, le pack Microsoft Office (version 95 ou ultérieur), vous pouvez utiliser MS-Query. Dans le cas contraire, je vous recommande WinSql qui est un graticiel (*freeware*) fort bien fait, léger et téléchargeable sur Internet à l'adresse http://www.imranweb.com.

Limite d'Access

Comme je vous l'ai dit plus haut, Access est un SGBD de qualité facile à utiliser mais malheureusement qui n'a pas été conçu pour un accès multi-utilisateurs. Ainsi la commande sqlBloque ne fonctionne pas lorsque vous essayez de verrouiller les enregistrements que vous souhaitez mettre à jour.

Pourtant, j'ai développé pour ma part plusieurs applications en réseau avec accès d'une dizaine de personnes sur le système de gestion de base de données Access. Comment ? En ajoutant à chaque table, un champ Verrou qui permet de bloquer logiquement l'enregistrement. A cela, j'ajoute une table Login qui comporte la liste des personnes accédant à la base.

Mécanisme de verrouillage

Le mécanisme de verrouillage de la base se décompose comme suit :

1. L'utilisateur s'identifie au niveau de l'application.
2. L'application récupère l'identifiant de l'utilisateur par un SELECT dans la table Login.
3. Lorsque l'utilisateur passe dans un module où une requête d'écriture sera exécutée, une requête dit de verrouillage est d'abord appliquée.
4. Une requête SELECT récupère la ligne d'enregistrement voulue avec son champ Verrou.
5. Si le champ Verrou est à zéro, c'est que l'enregistrement est libre.
6. Une requête UPDATE met à jour le champ Verrou avec l'identifiant utilisateur.

7. Le programme se déroule ensuite de manière classique.
8. A la fin, une nouvelle requête UPDATE met à jour le champ Verrou en le remettant à zéro.

Contrôle des verrous

Dans le cas où un utilisateur veut accéder à un enregistrement en cours d'utilisation, l'application va récupérer un champ Verrou différent de zéro (étape 4). On peut alors tout à fait gérer une boîte de dialogue par exemple, qui renvoie l'information que l'enregistrement *n* est bloqué par l'utilisateur *X*.

Le problème principal est le cas où l'utilisateur est sortie anormalement de l'application (coupure de courant) en laissant l'enregistrement verrouillé. Là encore, il suffit de faire un contrôle supplémentaire (étape 4) pour que l'utilisateur puisse ouvrir un enregistrement qu'il a précédemment bloqué.

Configuration

Avant de mettre en chantier votre application, vous devez penser en terme de performance. Le principal inconvénient du C/S est sa gourmandise en bande passante réseau. Il ne faut surtout pas raisonner en nombre d'utilisateurs. Combien de fois, ai-je entendu "j'ai une grosse application avec plusieurs dizaines d'utilisateurs". Or sur 50 utilisateurs seulement une petite dizaine vont réellement écrire dans la base, les autres ne feront que de la consultation. Dans un cas comme celui là, on réduit alors considérablement le nombre de transactions lourdes (action d'écriture). Bref, avant de demander une configuration mastoc de plusieurs kilos francs, assurez-vous que vous ne surdimensionnez pas.

Comme je l'ai déjà dit, j'ai déjà fait tourné de nombreuses applications avec par exemple une gestion bibliothécaire (prêt de livres) en configuration C/S Access sur réseau Novell pour une quinzaine d'utilisateurs. Sur ces quinze utilisateurs seulement cinq réalisaient les opérations de prêt et donc d'écriture sur le SGBD. La base supporte aujourd'hui plus de 15.000 références et 5.000 clients sans aucun problème. Le coût a été divisé par 10 par rapport à une solution sous Oracle.

Bien sûr pour des solutions plus lourdes, je porterai mon choix sur de véritables SGBD. Oracle ou Sybase sont les deux grands ténors du marché. Leur réputation réconforte le client mais ils sont souvent à un coût prohibitif pour une petite structure. Il existe donc des SGBD en *open source* comme PostGres ou MySQL, ceux-ci sont inconnus du grand public mais en revanche gratuit. A vous de convaincre.

Mais je ne négligerai pas non plus le serveur sur laquelle tournera la base. Si Windows NT est facile à gérer, il est nettement insuffisant pour des bases à plus de 100 utilisateurs, en ce cas il vaut mieux porter sa solution sous Unix ou AS400 capable de supporter plusieurs centaines de personnes.

Conclusion

Voilà, vous savez maintenant l'essentiel de la programmation en mode client/serveur. Je vous conseille de démarrer par une petite application afin de vous familiariser avec le langage SQL. Bon courage !

Copyright

© Cyril Beaussier - cyril.beaussier@mail.dotcom.fr
Création initial : 28 février 2000.
Dernière date de révision : 06 juin 2001.