

# GÉNIE LOGICIEL

# Génie logiciel

## Contenu

2

- Prérquis
- Introduction
  - ▣ Qu'est ce qu'un logiciel ?
  - ▣ La crise du logiciel
  - ▣ Le génie logiciel
- Qualités d'un logiciel
  - ▣ Qualités externes
  - ▣ Qualités internes

# Génie logiciel

## Contenu

3

- Outils du génie logiciel
  - ▣ Cycle de vie d'un logiciel
    - Introduction
    - Les différentes étapes d'un cycle de vie
    - Les différents documents d'un cycle de vie
  - ▣ Modèles de développement d'applications
    - Introduction
    - Approches classiques
    - Approches intégrant l'utilisateur
    - Approches orientées objet

# Génie logiciel

## Contenu

4

- ▣ Ateliers de génie logiciel
  - Qu'est ce qu'un atelier de génie logiciel ?
  - Outils CASE
  - Intégration d'outils CASE
  - Différents types d'AGL
  - Exemples d'AGL

### ▣ Quelques patterns

- ▣ Objectif des patterns
- ▣ Différents types
- ▣ Exemples de patterns créationnels
- ▣ Exemples de patterns structuraux
- ▣ Exemples de patterns Comportementaux

# Génie logiciel

## Contenu

5

- **Planification et gestion de projet**
  - ▣ Introduction et Rappels
  - ▣ Mise en œuvre d'un projet avec MS Project
  - ▣ Gestion des tâches
  - ▣ Gestion des ressources
  - ▣ Contrôle, suivi et audit d'un projet

# GÉNIE LOGICIEL

*Prérequis*

# Prérequis

7

- Bases de Données
  - ▣ Conception
  - ▣ SQL
  - ▣ PL/SQL
- L'Orienté Objet
  - ▣ Concepts de l'Orienté Objet (classe, objet, encapsulation, classes abstraites et interfaces, ....)
  - ▣ Langage UML
  - ▣ Langage de programmation Orienté Objet (Java, C++, C#, PHP5, ...)
- ..

# GÉNIE LOGICIEL



# Qu'est ce qu'un logiciel ?

9

- «Le logiciel est l'ensemble des programmes, procédés, règles et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitements de l'information» (arrêté du 22 dec. 1981)
- Documentation du logiciel sert à :
  - ▣ Installation : procédure à suivre pour installer
  - ▣ Utilisation : Guide utilisateur, guide programmeur
  - ▣ Développement : Code source
  - ▣ Maintenance : Code source

# La crise du logiciel (Source)

10

- Apparue à la fin des années 60
- Provient du décalage entre le progrès matériels d'une part et logiciel d'autre part.
  - ▣ Matériel : de plus en plus puissants, de moins en moins coûteux
  - ▣ Logiciel : plusieurs développeurs, plusieurs méthodes avec des notations et des concepts différents

# La crise du logiciel (Signes)

11

## Encadré1

- La construction de logiciel coûtait très cher (200 millions de dollars pour fabriquer OS-350)
- Les délais ne sont pas respectés (2 ans de retard pour les premiers compilateurs PL/1, Algol 68 et ADA)
- Les logiciels ne sont pas évolutifs (parfois écrits en assembleur pour un type particulier de machine)
- D'après une étude sur 8380 projets (Standish Group, 1995)
  - Succès : 16%
  - Problème (budget ou délai non respecté, défaut de fonctionnalités, ...) : 53%
  - Echec (Projet abandonné) : 31%

# La crise du logiciel (Signes)

12

Signes apparus au niveau de :

- ▣ **Coût** : logiciel à coût élevé
- ▣ **Délais** : délais non respecté
- ▣ **Non fiabilité** : erreurs d'exécution, pannes, erreurs de fonctionnalités, ...

► ► Besoin d'une nouvelle discipline !!

# Génie logiciel

## Nouvelle discipline (Motivation)

13

### Matériel

- Représente 20% du marché
- Fabrication assurée par quelques fabricants
- Fabrication standardisée (Structure de Van newman)

### Logiciels

- Représentent 80% du marché
- Conçus et développés par plusieurs développeurs
- Utilisent plusieurs méthodes, langages et outils

**Donc, dans un système informatique, les problèmes informatique sont essentiellement liés à des problèmes de logiciels**

# Le génie logiciel

## Nouvelle discipline (Motivation)

15

### Naissance d'une nouvelle discipline :

# Génie Logiciel

# Génie logiciel

## Définition

16

- «le génie logiciel est l'ensemble des activités de conception et des procédures tendant à rationaliser la **production** du logiciel et de son **suivi**» (arrêté du 30 dec. 1983)
- En d'autres termes, le génie logiciel est l'ensemble de :
  - ▣ Procédures (démarches)
  - ▣ Méthodes
  - ▣ Langages
  - ▣ Imposés (Règles)
  - ▣ Ateliers

Pour favoriser la **production** et la **maintenance** de composants logiciels de **qualité avec le moindre coût dans les plus bref delai.**



# Génie logiciel

## Définition

17

On distingue des méthodes :

- ▣ d'analyse (permet de définir les problèmes)
- ▣ de conception (permet de trouver des solutions aux problèmes)
- ▣ de codage (permet de traduire les solutions en code source)
- ▣ de test (permet de vérifier la validité du code source)
- ▣ de maintenance (permet d'assurer la pérennité du code source)

# Génie logiciel

## Définition

18

Parmi les procédures on peut citer :

- ▣ la gestion de projet (organiser les tâches, les ressources et les coûts)
- ▣ l'assurance qualité (garantir un niveau de fiabilité et de performance)
- ▣ les métriques (mesurer les performances)
- ▣ le suivi (assurer la gestion au quotidien, les rapports d'anomalie, etc.)

# Génie logiciel

19

## Encadré 2

La conférence d'OTAN en Allemagne (1968) sur le génie logiciel a fixé les points suivants:

- Un projet informatique :
  - ▣ N'est pas fiable
  - ▣ Ne répond pas aux attentes qu'elle suscite
  - ▣ Rarement réalisé dans les délais prévus
  - ▣ Coûte très cher
- Le but du génie logiciel :
  - ▣ Comment faire des logiciels de qualité avec le moindre coût dans les plus bref délai ?

# GÉNIE LOGICIEL

Qualités d'un logiciel

# Qualités d'un logiciel

21

*L'objectif du génie logiciel est de produire des logiciels de qualité en respectant le coût et les délais (OTAN 68)*



Quels sont les critères de **QUALITE** d'un logiciel?

# Qualités d'un logiciel

22

On peut séparer les qualités d'un logiciel en deux catégories :

- ▣ **Qualités externes** : perçues lors de l'utilisation du logiciel (logiciel = Boîte noire)
- ▣ **Qualités internes** : perçues lors de la maintenance du logiciel (logiciel = Boîte blanche)

# Qualités d'un logiciel lors de son utilisation

23

Un logiciel doit répondre aux besoins des utilisateurs

- ▣ Fiabilité
- ▣ Ergonomie
- ▣ Portabilité
- ▣ Utilité
- ▣ Utilisabilité (maniabilité)
- ▣ Interopérabilité ou couplabilité

# Qualités d'un logiciel lors de sa maintenance

24

Un logiciel doit être facile à maintenir pour le corriger, l'améliorer, l'adapter aux changements de son environnement, ...

- ▣ Flexibilité
- ▣ Portabilité
- ▣ Structuration



# Qualités d'un logiciel

## Utilité

25

Adéquation entre 'le besoin effectif de l'utilisateur' et 'les fonctions offertes par le logiciel'

### Solutions

- ▣ Emphase sur l'analyse des besoins
- ▣ Amélioration de la communication (langage commun, démarche participative: clients, développeurs, ...)

# Qualités d'un logiciel

## Utilisabilité

26

- Facilité d'apprentissage: comprendre ce que l'on veut faire avec le logiciel, et savoir comment le faire
- Facilité d'utilisation: proposer plusieurs alternatives, regrouper des options similaires, ...

### Solutions

- ▣ Analyser le mode opératoire de l'utilisateur
- ▣ Adapter l'ergonomie du logiciel aux utilisateurs
- ▣ Respecter les standards

# Qualités d'un logiciel

## Fiabilité

27

- Conformité et justesse :
  - ▣ le logiciel doit être conforme aux spécifications
  - ▣ les résultats obtenus sont ceux attendus
- Robustesse et sûreté :
  - ▣ le logiciel doit fonctionner en toutes circonstances (notions d'exceptions en Oracle, Gestion des erreurs en VB, Gestion des pannes en utilisant la journalisation, ...)

# Qualités d'un logiciel

## Interopérabilité

28

Un logiciel doit pouvoir interagir en synergie avec d'autres logiciels (OLE et ODBC, par exemple)

### Solutions

- ▣ Eviter les systèmes de fichiers classiques
- ▣ Utiliser des bases de données
- ▣ Externaliser certaines fonctions en utilisant des API (API de Windows, par exemple)

# Qualités d'un logiciel

## Portabilité

29

Un même logiciel doit pouvoir fonctionner sur plusieurs plateformes (machines et systèmes d'exploitation) différentes

### Solutions

- ▣ Éviter le langage d'assembleur
- ▣ Utiliser des langages de haut niveau
- ▣ Utiliser des machines virtuelles (Java, par exemple)

# Qualités d'un logiciel

## Facilité de la maintenance

30

Un logiciel doit être facile à maintenir

### Solutions

- ▣ Modularité et structuration : utilisation des modules ou des classes, des fonctions ou des procédures
- ▣ Réutilisabilité : utilisation des composants prêts à l'emploi (ActiveX, par exemple)

# Qualités d'un logiciel

## Facilité de la maintenance

31

On distingue plusieurs types de maintenance :

- ▣ Maintenance évolutive: ajouter de nouvelles fonctions au logiciel pour répondre aux nouveaux besoins et de nouveaux environnement.
- ▣ Maintenance corrective: permet de corriger les erreurs, telles que les défauts d'utilité, d'utilisabilité, de fiabilité, ...
- ▣ Maintenance adaptative: permet d'ajuster le logiciel pour qu'il continue à remplir son rôle compte tenu de l'environnement, par exemples: changement de SGBD, machines, taux TVA, Euro, ...

Outils de logiciel – Cycle de  
vie d'un logiciel

GÉNIE LOGICIEL



# Cycle de vie d'un logiciel

33

- Processus (ensemble d'activités) nécessaire au développement et à la maintenance d'un logiciel
- Composé de plusieurs phases autonomes mais dépendantes (interdépendantes).
- Chaque étape se termine par la remise de un ou plusieurs documents validé conjointement par l'utilisateur et le développeur.

# Cycle de vie d'un logiciel

34

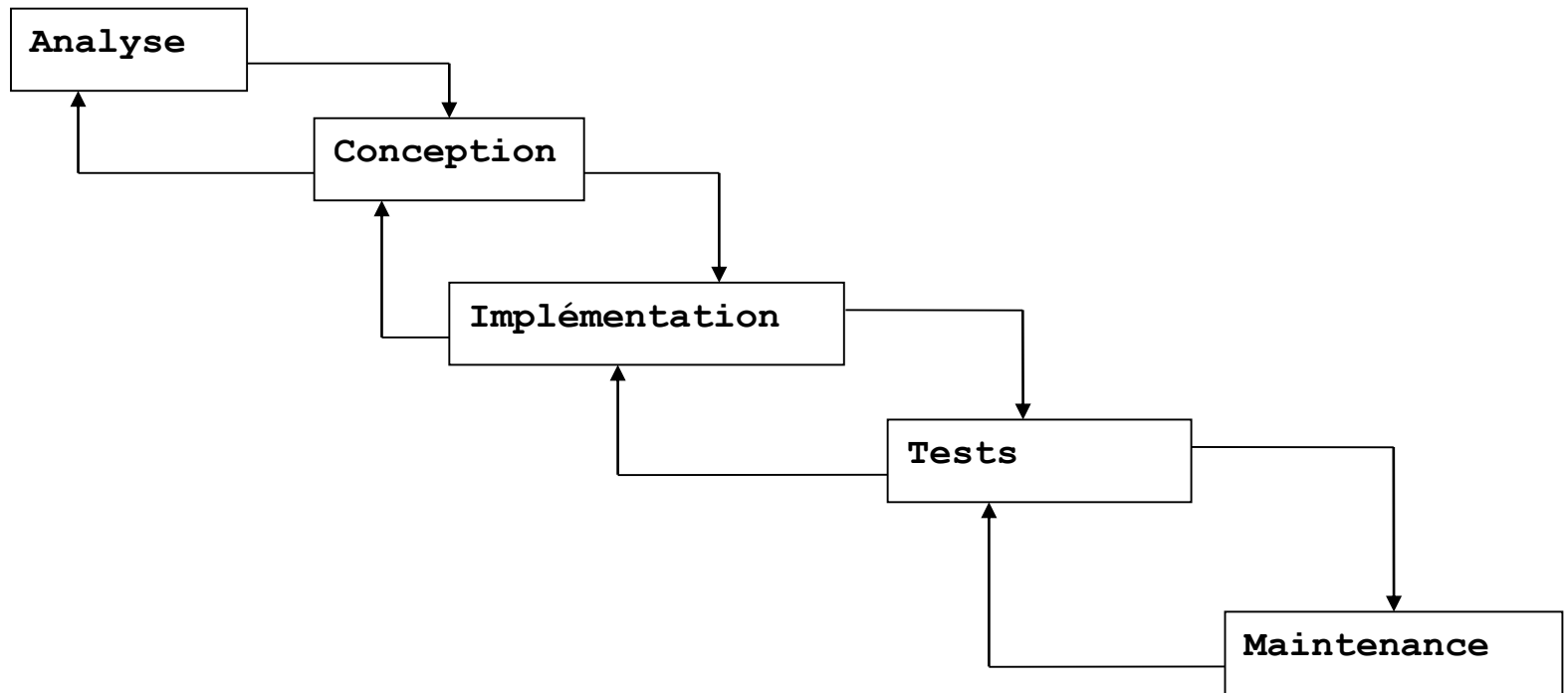
Étapes nécessaires à la réalisation d'un logiciel:

- ▣ Analyse
- ▣ Conception
- ▣ Codage (Implémentation)
- ▣ Tests
- ▣ Livraison
- ▣ Maintenance

# Cycle de vie d'un logiciel

## Modèle en Cascade (WaterFall)

35



# Cycle de vie d'un logiciel

## Analyse

36

- Elle a pour but de dégager le problème à étudier.
- Le résultat de l'analyse est le cahier de charges (exprimé dans une langue naturelle) contenant les besoins du futur système.
- Cette spécification est informelle.
- 3 phases:(Faisabilité, Spécifications des besoins, Organisation du projet)

# Cycle de vie d'un logiciel

## Faisabilité

37

- Première étape du cycle de vie d'un logiciel
- Permet de déterminer si le développement proposé vaut la peine d'être mis en œuvre.
- ▶ ▶ Etudier le marché pour déterminer s'il existe un marché potentiel pour le produit.

# Cycle de vie d'un logiciel

## Spécification des besoins

38

- Permet de définir ce que doit faire le logiciel et non comment il le fait
- Quatre types de spécifications
  - ▣ Spécifications générales
  - ▣ Spécifications fonctionnelles
  - ▣ Spécifications d'interface
  - ▣ Spécifications techniques

# Cycle de vie d'un logiciel

## Spécification des besoins

39

La spécification générale consiste à identifier :

- Objectifs à atteindre
- Contraintes (utilisation du matériel et outils existants)
- Règles de gestion à respecter

# Cycle de vie d'un logiciel

## Spécification des besoins

40

Spécification fonctionnelles est la description des fonctionnalités du futur logiciel de manière détaillée que possible

Spécification d'interface décrit les interfaces du logiciel avec le monde extérieur : homme, autres logiciel, machines



# Cycle de vie d'un logiciel

## Spécification des besoins

41

### Spécification technique :(Etude de l'existant)

- ▣ Moyens d'accès (local, distant, Internet, ...)
- ▣ Temps de réponse acceptable (gestion des GAB, gestion des emplois de temps, ...)
- ▣ Plateforme (Windows, Unix, ...)
- ▣ Quantité d'informations à stocker (choix du SGBDR, ...)

# Cycle de vie d'un logiciel

## Spécification des besoins

42

### Méthodes de spécification

- ▣ SADT : décomposition du système en sous systèmes
- ▣ MERISE (Recueil des informations avec :
  - des interviews des différents postes de travail,
  - documents utilisés,
  - connaissance du problème,
  - expérience, ...)
- ▣ Diagramme des cas d'utilisation de UML
- ▣ ...

### Divers

Cette phase occupe environ 15% du temps du développement

# Cycle de vie d'un logiciel

## Organisation du projet

43

- Appelée aussi **Planification et gestion de projet**
- Permet de déterminer la manière de développer le logiciel
- La phase de planification permet de :
  - ▣ **découper le projet en tâches**
  - ▣ **décrire leur enchaînement** dans le temps,
  - ▣ **affecter à chacune une durée et un effort** calculé en homme\*mois.

# Cycle de vie d'un logiciel

## Organisation du projet

44

### Plusieurs étapes :

- Analyse des coûts: estimation du prix du projet
- Planification: calendrier pour le développement (jours ouvrables, jours fériés, nombre d'heures de travail par jours, ...)
- Répartition des tâches: distribuer les tâches et les sous tâches (affectation des ressources aux tâches)

# Cycle de vie d'un logiciel

## Conception

45

- Permet de fournir une description fonctionnelle (formelle) du système en utilisant une méthode.
- Les méthodes proposent des formalismes (dans la plupart des temps graphiques) pour concevoir le système.
- Deux types de conception :
  - ▣ Conception générale
  - ▣ Conception détaillée

# Cycle de vie d'un logiciel

## Conception générale

46

- Appelée aussi : 'Conception architecturale'
- Se focaliser sur l'architecture générale du système :  
décomposition du système en sous système

Exemple, pour la gestion d'une entreprise :

- ▣ Employés
- ▣ Départements
- ▣ ...

# Cycle de vie d'un logiciel

## Conception générale

47

### Méthodes de conception générale

- ▣ OOA/OOD (Y. CODD) : conception orientée objet
- ▣ ...

### Divers

Cette phase représente environ 10% du temps du développement d'un logiciel

# Cycle de vie d'un logiciel

## Conception détaillée

48

- Produire une description externe de chacune des procédures, fonctions et des structures de données (conception classique)
- Produire de manière précise les contenus des objets en terme de propriétés et de méthodes (conception Orientée Objet)



# Cycle de vie d'un logiciel

## Conception détaillée

49

### Méthodes de conception détaillée

- ▣ Modèle Entités – Associations
- ▣ OMT, BOOCH (OOD), UML,
- ▣ ...

### Divers

- ▣ 25% du temps de développement
- ▣ On peut utiliser le pseudo code pour décrire les fonctions, procédures, méthodes, ...

# Cycle de vie d'un logiciel

## implémentation (Réalisation)

50

- Des programmes seront élaborés afin de mettre en œuvre des solutions techniques précédemment retenues
- Plusieurs types langages sont utilisés:
  - ▣ Langages classiques (C, Pascal, Fortran, ...)
  - ▣ Langages orientés objets (C++, Java, C#, ...)

# Cycle de vie d'un logiciel

## Codage et Tests

51

On distingue plusieurs types de tests :

- ▣ Test unitaire: tester les parties du logiciel par les développeurs
- ▣ Test d'intégration: tester pendant l'intégration des modules
- ▣ Test système: tester dans un environnement proche à celui de production
- ▣ Test alpha: tests faits par le client sur le site de développement
- ▣ Test bêta: tests faits par le client sur le site de production
- ▣ Test de régression: enregistrer les résultats des tests et les comparer avec ceux des anciens versions pour déterminer si la nouvelle n'a pas apporté de dégradation de performance.

# Cycle de vie d'un logiciel

## Livraison

52

- Fournir au client une solution logiciel qui fonctionne correctement.
- Plusieurs tâches :
  - ▣ Installation: rendre le logiciel opérationnel sur le site du client
  - ▣ Formation: enseigner aux utilisateurs à se servir de ce logiciel
  - ▣ Assistance: répondre aux questions de l'utilisateur

# Cycle de vie d'un logiciel

## Maintenance

53

- Mettre à jour et améliorer le logiciel
- La maintenance comprend :
  - ▣ Corrective : correction des erreurs et anomalies
  - ▣ Adaptative : adaptation à de nouveaux environnements
  - ▣ Évolutive ou Perfective : ajout de nouvelles fonctionnalités

# Cycle de vie d'un logiciel

## Documents

54

- Cahier des charges: description des fonctionnalités désirées (décrites par l'utilisateur)
- Spécifications: décrit ce que doit remplir le logiciel (décrites par le développeurs) :
  - ▣ Modèle Objet: Classes et objets
  - ▣ Scénarios des cas d'utilisation: différents enchaînements possibles du point de vue de l'utilisateur
  - ▣ ...

# Cycle de vie d'un logiciel

## Documents

55

- Calendrier du projet: indique les tâches, les délais et les ressources
- Plan de test: indiquent les procédures de tests à appliquer
- Manuel d'utilisateur: mode d'emploi du logiciel dans sa version finale

# Cycle de vie d'un logiciel

## Documents

56

- Code source: code complet du produit fini
- Rapport des tests: décrit les tests effectués et les réactions du système
- Rapport des défauts: décrit les comportements du système qui n'ont pas satisfait le client.



Outils de logiciel – Modèle de  
développement d'un logiciel

GÉNIE LOGICIEL

# Modèles de développements d'applications

58

- Le cycle de vie est l'ensemble des étapes par lesquelles nous devons passer pour développer un logiciel
- Un modèle est la manière d'enchaîner les étapes du cycle de vie

# Modèles de développements d'applications

59

Plusieurs approches ont été abordées :

- ▣ Approches classiques
- ▣ Approches faisant intervenir l'utilisateur
- ▣ Approches orientées objet

# Approches classiques

60

- Suivent le cycle de vie traditionnel : spécification, conception, implémentation, tests et maintenance
- Basées sur un processus séquentiel, linéaire (on ne passe à une étape qu'après avoir validé l'étape précédente)
- Utilisées dans la plupart des applications traditionnelles

Exemples :

- ▣ Cycle en cascade
- ▣ Cycle en V
- ▣ Etc.

# Approches classiques

## Modèle en Cascade

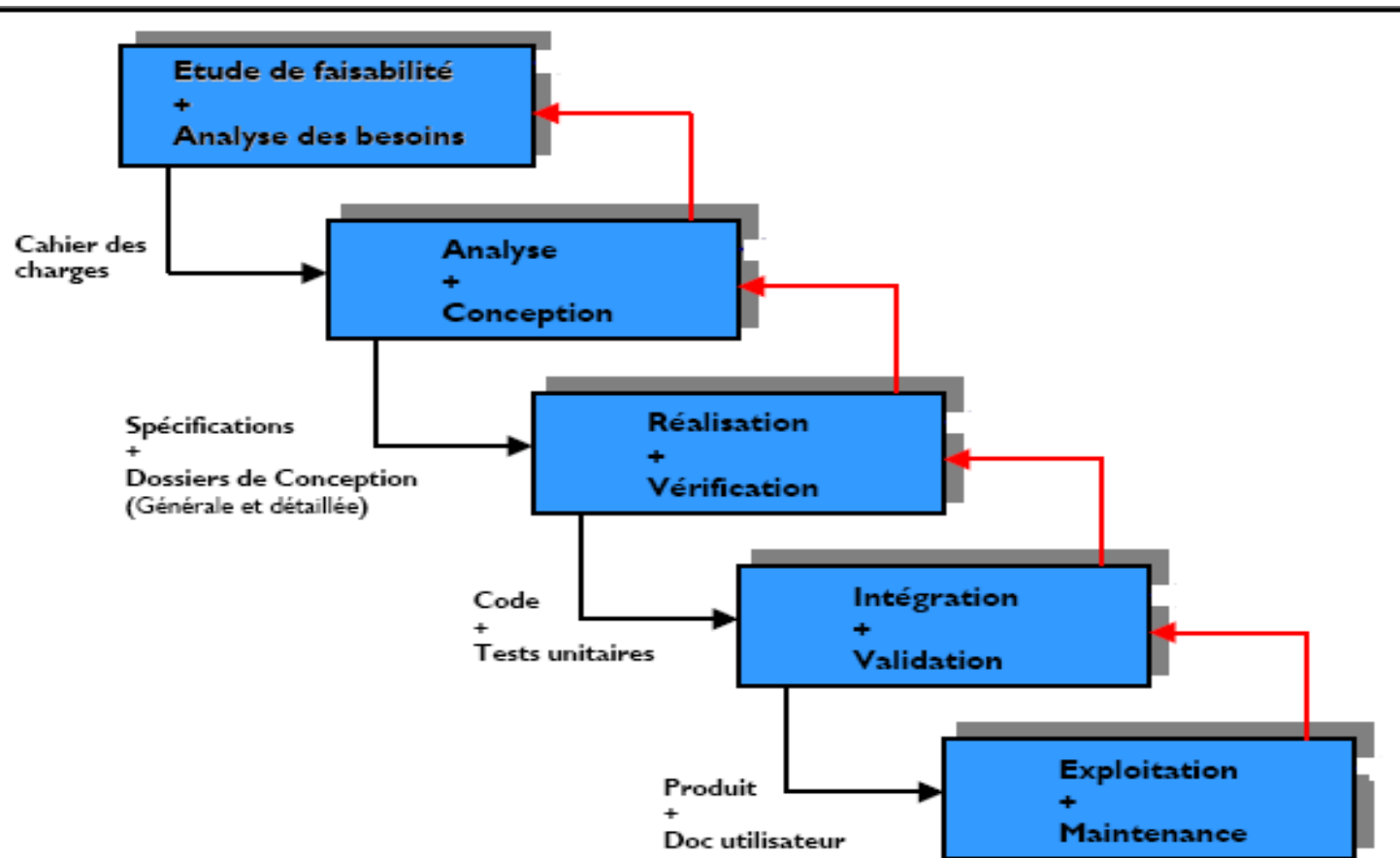
61

- Inventé en 1970 par Royce
- Constitué de plusieurs étapes (phases) : à chaque phase correspond une **activité principale** bien précise produisant un certain nombre de **livrables**

# Approches classiques

## Modèle en Cascade (WaterFall)

62



# Approches classiques

## Modèle en Cascade

63

Approche adaptée pour traiter l'aspect informatique des applications classiques qui sont :

- ▣ Pleinement spécifiées
- ▣ Pleinement analysées
- ▣ Pleinement conçues
- ▣ Pleinement implémentées
- ▣ Pleinement testées et validées

# Modèle en Cascade

## Inconvénients

64

- ❑ Aucun aspect utilisateur n'est pris en considération (sauf dans l'expression des besoins)
- ❑ Considère que tous les utilisateurs sont égaux, éprouvant les mêmes besoins
- ❑ Obligation de définir la totalité des besoins au départ
- ❑ Obtiennent tardivement une version exécutable



# Modèle en Cascade

## Inconvénients

65

- Développent n'importe quel projet de manière séquentiel, alors que les projets sont rarement séquentiels
- Validation tardive : les erreurs d'analyse, de conception et d'implémentation ne seront détectées qu'à la fin
  - ▶ Erreurs coûtent très chères
  - ▶ Délais peuvent être dépassées

# Approches classiques

## Modèle en V

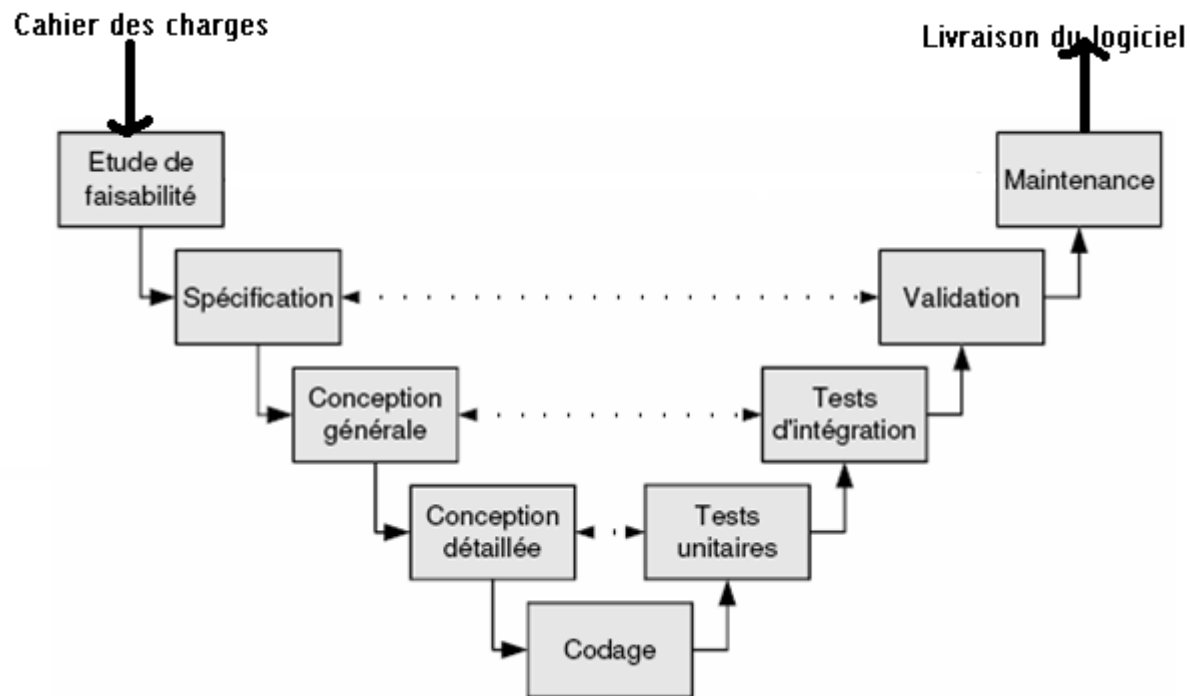
66

- Enchaînement des phases est représenté graphiquement par un «V»
- Ce modèle met en correspondance le moment où un besoin s'exprime et le moment où ce besoin devient approuvable et validable.

# Approches classiques

## Modèle en V

67



# Approches classiques

## Modèle en V

68

- **Avantage :**
  - ▣ Validation de chaque étape de manière séparée
- **Inconvénients**
  - ▣ Mêmes inconvénients du modèle en cascade

# Approches faisant intervenir l'utilisateur

69

Ces approches consistent à :

1. Réaliser une première partie (maquette)
2. Celle-ci sera testée auprès des utilisateurs
3. Des améliorations et des corrections sont alors apportées
4. Ainsi de suite, jusqu'à obtention d'une version finale stable

# Approches faisant intervenir l'utilisateur

70

Suggère de réaliser le logiciel de manière :

- Interactive
- Itérative (plusieurs passes)
- Incrémental

En d'autres termes :

- Modéliser une partie du système
- La valider auprès de l'utilisateur final
- La construire entièrement

# Approches faisant intervenir l'utilisateur

71

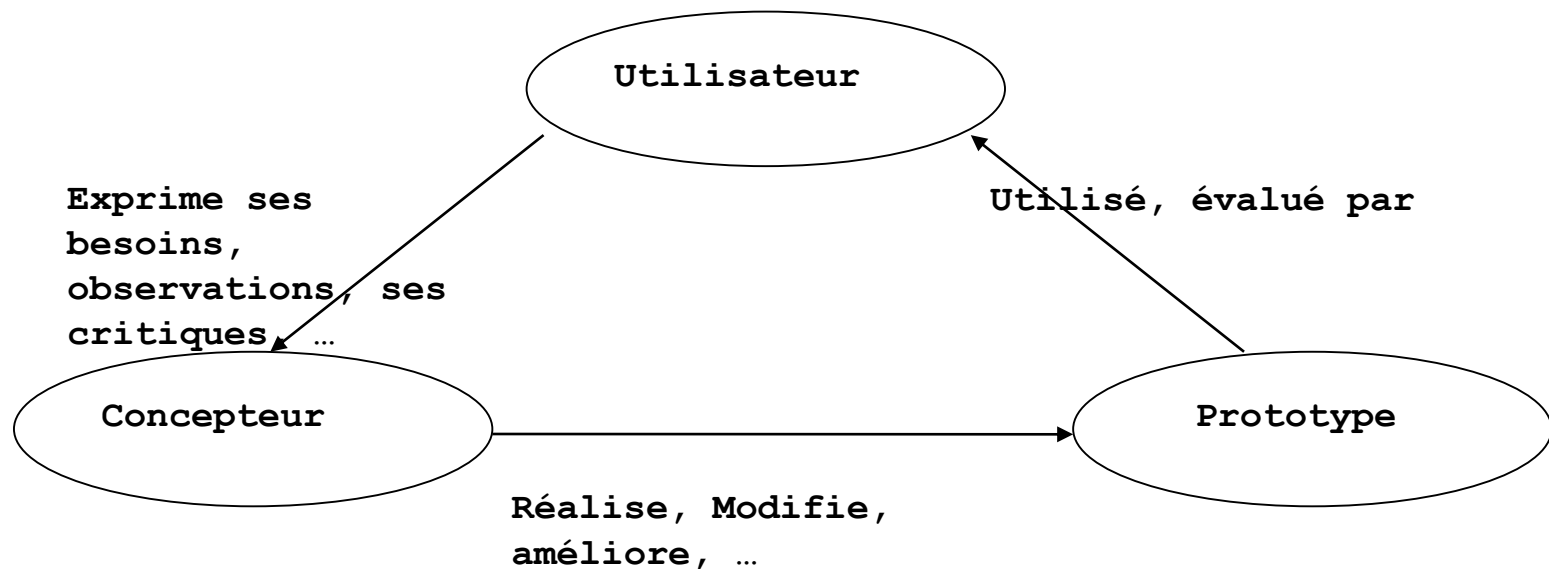
Exemple d'un processus incrémental :

- Livraison 1 = maquette IHM
- Livraison 2 = version de démonstration
- Livraison 3 = version préliminaire
- Livraison 4 = version complète

# Approches faisant intervenir l'utilisateur

72

Le cycle de vie repose sur les caractères itératif et interactif dans la relation utilisateur-concepteur-prototype





# Approches faisant intervenir l'utilisateur

73

On distingue plusieurs méthodes de ce type, la plus utilisée est :

**Modèle en spirale**

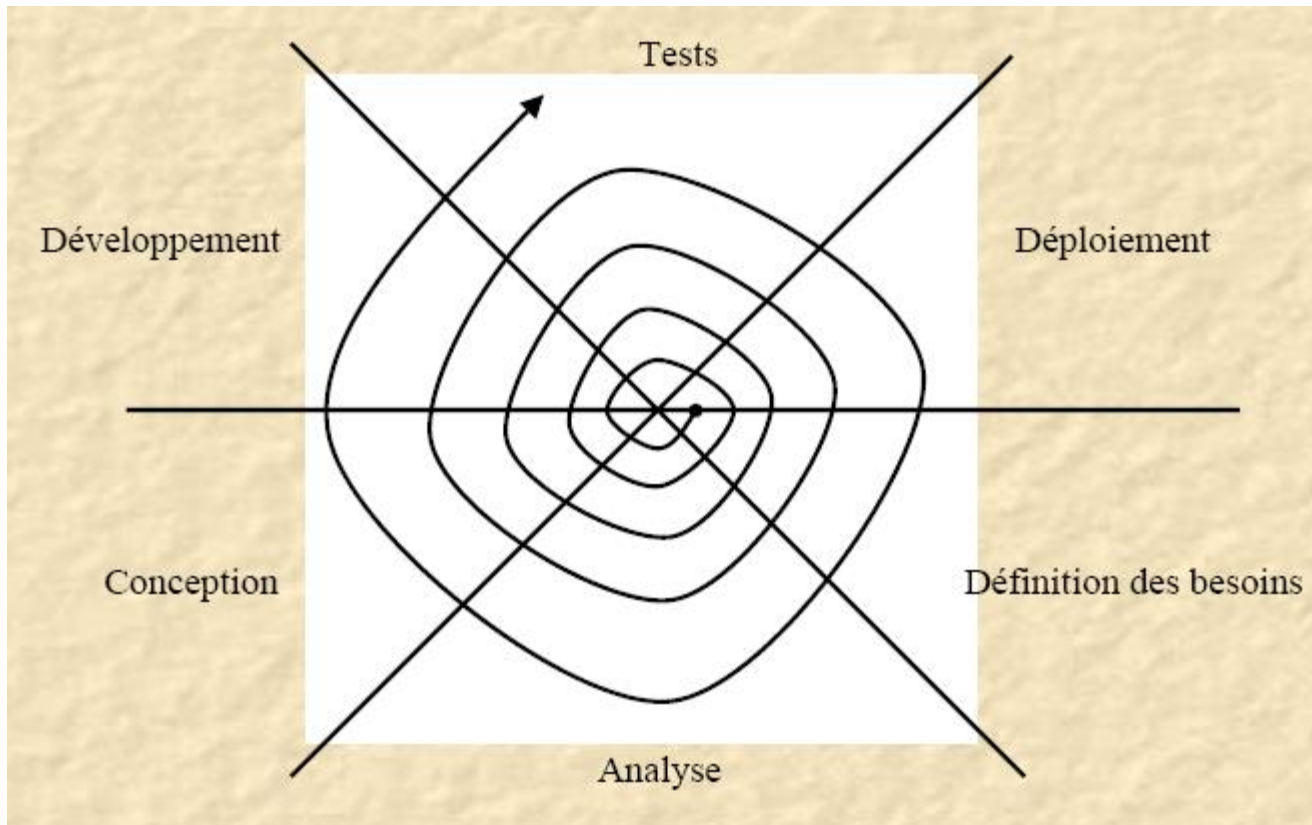
# Modèle en spirale

74

- Inventé par Boehm en 1998
- Consiste à étendre progressivement la couverture du champ du logiciel après chaque validation

# Modèle en spirale

75



# Approches faisant intervenir l'utilisateur (Conclusion)

76

Ces approches permettent :

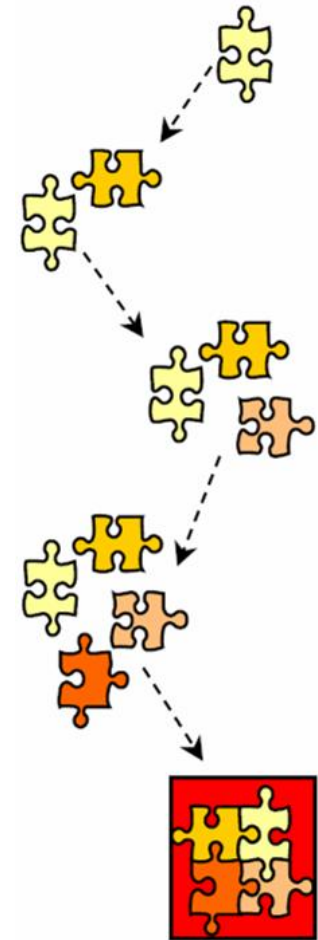
- Évaluation à chaque étape de développement
- Intégration du facteur humain dans toutes les étapes du cycle de vie
- Dialogue continu avec l'utilisateur dès les phases en amont, ce qui évite des remarques à posteriori beaucoup plus coûteuses.

# Approches faisant intervenir l'utilisateur (Conclusion)

77

Fabriquer un composant du système global  
à chaque cycle

- ▣ Accepter une définition floue au départ
- ▣ Lever les difficultés par les étapes successives
- ▣ Assembler et lier le tout lors de la dernière étape



# Approches faisant intervenir l'utilisateur (Conclusion)

78

Ces approches sont plus adaptées aux applications interactives pour plusieurs raisons :

- ▣ Les utilisateurs sont incapables de formuler et de quantifier leurs besoins dès le début
- ▣ Les besoins sont mouvants
- ▣ Pour des applications hautement interactives, la convivialité de l'interface est importante, d'où l'importance de l'évaluation de l'utilisateur (seule personne sensée être en contact direct avec l'application) durant toutes les phases de développement

# Approches faisant intervenir l'utilisateur

79

Ces approches présentent des inconvénients, parmi lesquels :

- ▣ Manque d'outils de prototypage
- ▣ Non disponibilité de l'utilisateur final pour la validation durant toutes les étapes de développement
- ▣ Etc.

# Approches orientées objet (Rappel)

80

## Approches classiques

- ➡ Séparation nette entre les données et les procédures de traitement de ces données

## Approches orientées objets

- ➡ Utilise des entités (appelées objets) combinant à la fois des données (partie statique, passive) et méthodes (partie dynamique, active)

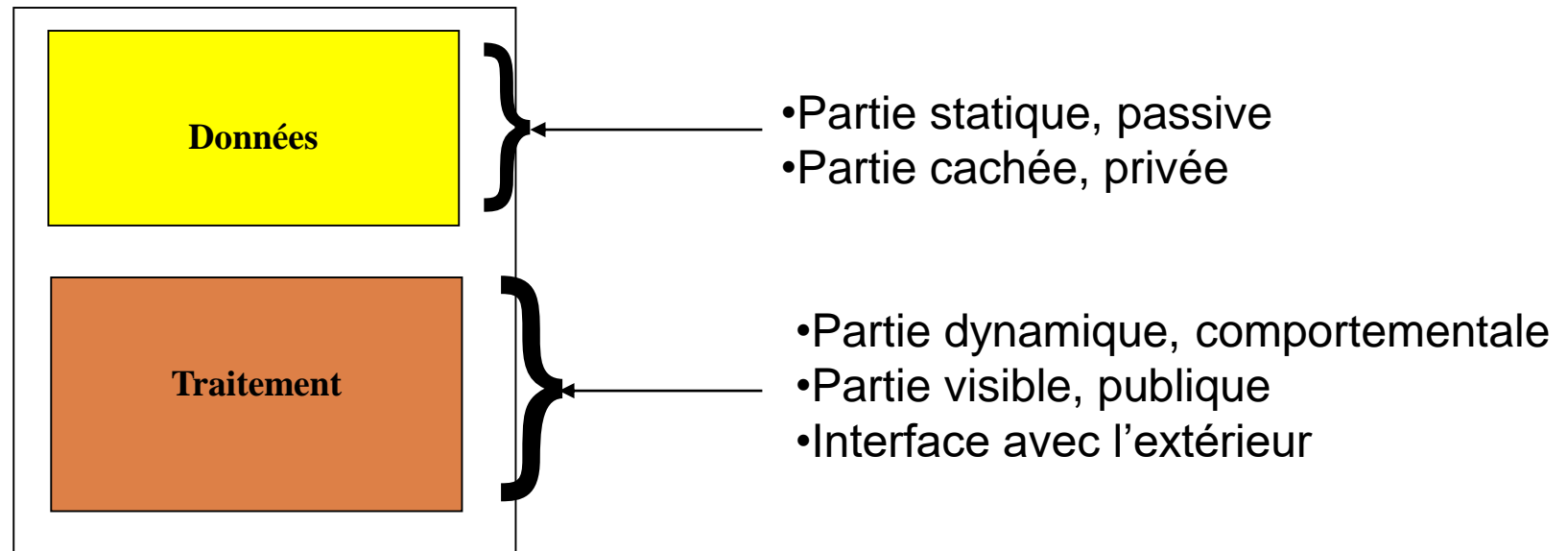


# Approches orientées objet

## (Rappel)

81

Une classe regroupe dans une même entité ses données et ses méthodes



# Approches orientées objet

82

## Plusieurs méthodes orientées objets

- OOD (G. BOOCH)
  - Diagrammes de classes
  - Diagrammes d'instances
  - Diagrammes de transitions
- OMT (J. RUMBAUGH)
  - Modèle Objet
  - Modèle Dynamique
  - Modèle Fonctionnel
- UML
  - Diagrammes des séquences
  - Diagrammes d'activités
  - Diagrammes de classes
  - ...
- OOA/OOD (Y. COAD)
  - ...
- OOSE (I. JACOBSON)
  - ...

# Approches orientées objet

83

## Traçabilité

la possibilité d'effectuer aisément une correspondance entre les éléments définis dans deux phases successives.

# Approches orientées objet

84

## Principal avantage : Bonne Traçabilité

Les mêmes concepts servent depuis l'analyse jusqu'à l'implémentation (les classes implémentées sont découvertes depuis l'analyse)

En effet : ....

# Approches orientées objet

85

## Projet traditionnel

- Analyse/Conception
  - ▣ Entités
  - ▣ Relations
  - ▣ Traitements
- Implémentation
  - ▣ Structure
  - ▣ Type
  - ▣ Fonction/Procédures

## Projet Objet

- Analyse
  - ▣ Classes
  - ▣ ...
- Conception
  - ▣ Classes
  - ▣ ...
- Implémentation
  - ▣ Classes
  - ▣ ...

Ateliers de génie logiciel

GÉNIE LOGICIEL

# Ateliers de génie logiciel

87

- Un AGL (Atelier de Génie Logiciel) ou outil CASE (Computer Aided Software Engineering) est un logiciel aidant à la réalisation de logiciels
- Un AGL intègre des outils adaptés aux différentes phases de la production d'un logiciel (spécification, conception, codage, ...)

# Ateliers de génie logiciel

88

## Exemples d'outils intégrés dans un AGL

- ▣ Éditeurs de texte
- ▣ Diagrammes
- ▣ SGBD
- ▣ Outils de mise en forme
- ▣ Générateurs de code
- ▣ ...



# Ateliers de génie logiciel

89

Ces outils interviennent lors d'une ou de plusieurs phases du cycle de vie d'un logiciel

- ▣ Conception (éditeurs de texte, diagrammes, ...)
- ▣ Programmation (éditeurs de texte, compilateurs, générateurs de code, ...)
- ▣ Tests (débuggers, vérificateurs, ...)
- ▣ Etc...

# Ateliers de génie logiciel

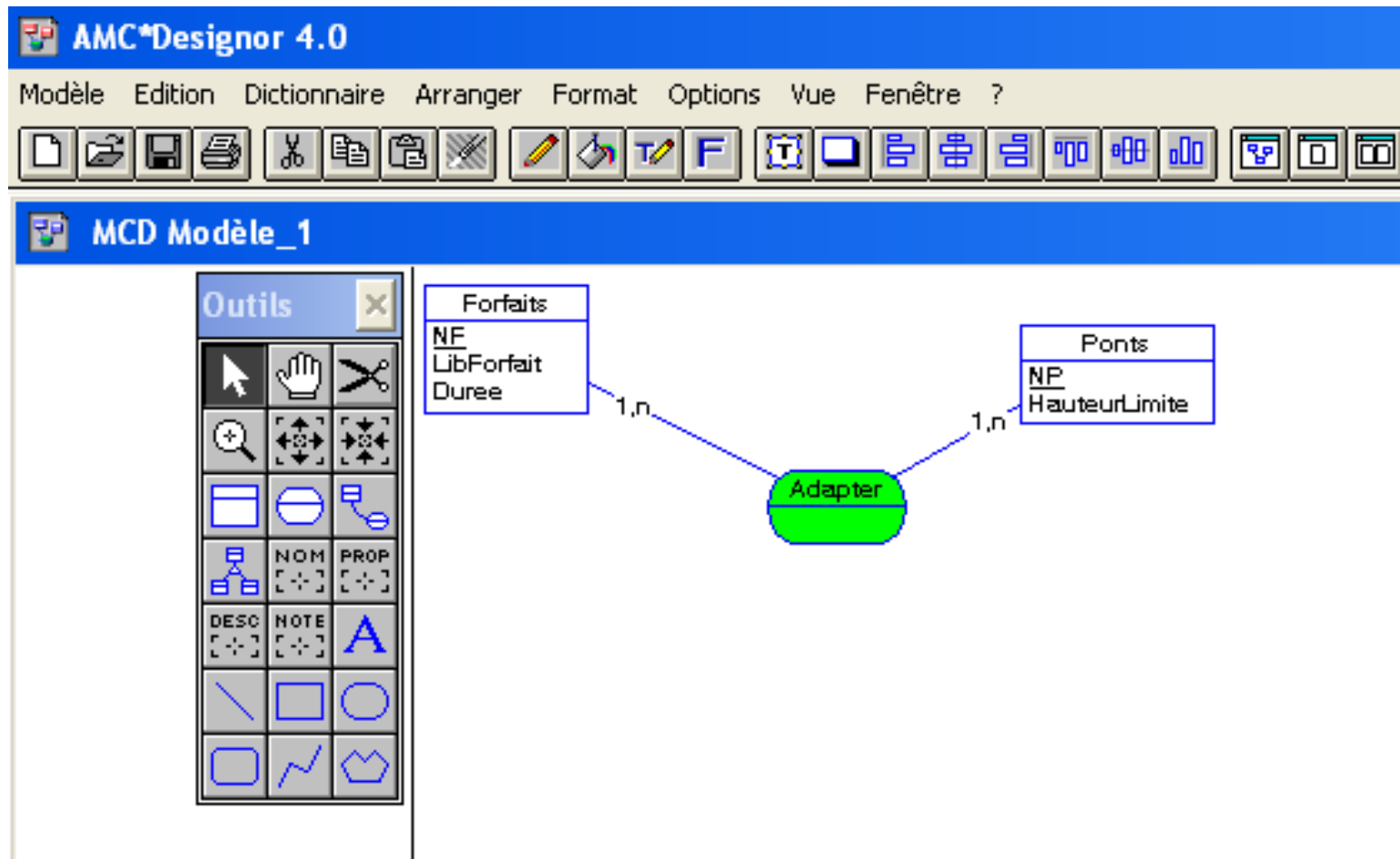
90

On distingue deux types d'AGL

- Environnement de conception (UpperCase): interviennent lors des étapes d'analyse et de conception d'un logiciel  
Exemples, Tramis, AMC\*Designor, Power AMC, ...
- Environnement de développement (LowerCase): interviennent lors des étapes de codage et de tests d'un système  
Exemple, WinDev, WevDev,...

# Exemple d'AGL : AMC\*Designor

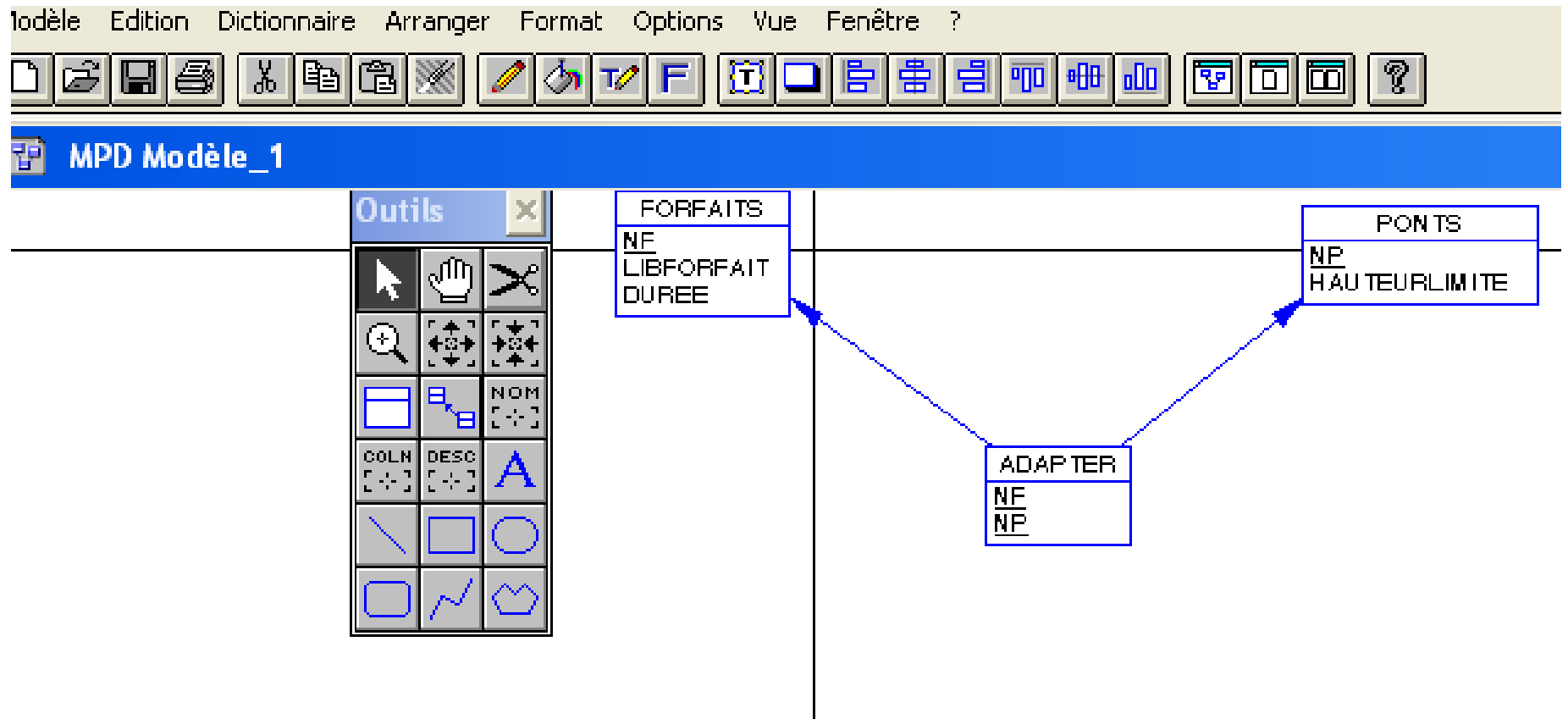
## Modèle Conceptuel de Données



# Exemple d'AGL : AMC\*Designor

## Modèle Physique de Données

92



# Exemple d'AGL : AMC\*Designor

## Script de création de Bases de Données

93

```
-- =====
--   Table : ADAPTER
-- =====
create table ADAPTER
(
    NF                INTEGER                not null,
    NP                INTEGER                not null
);

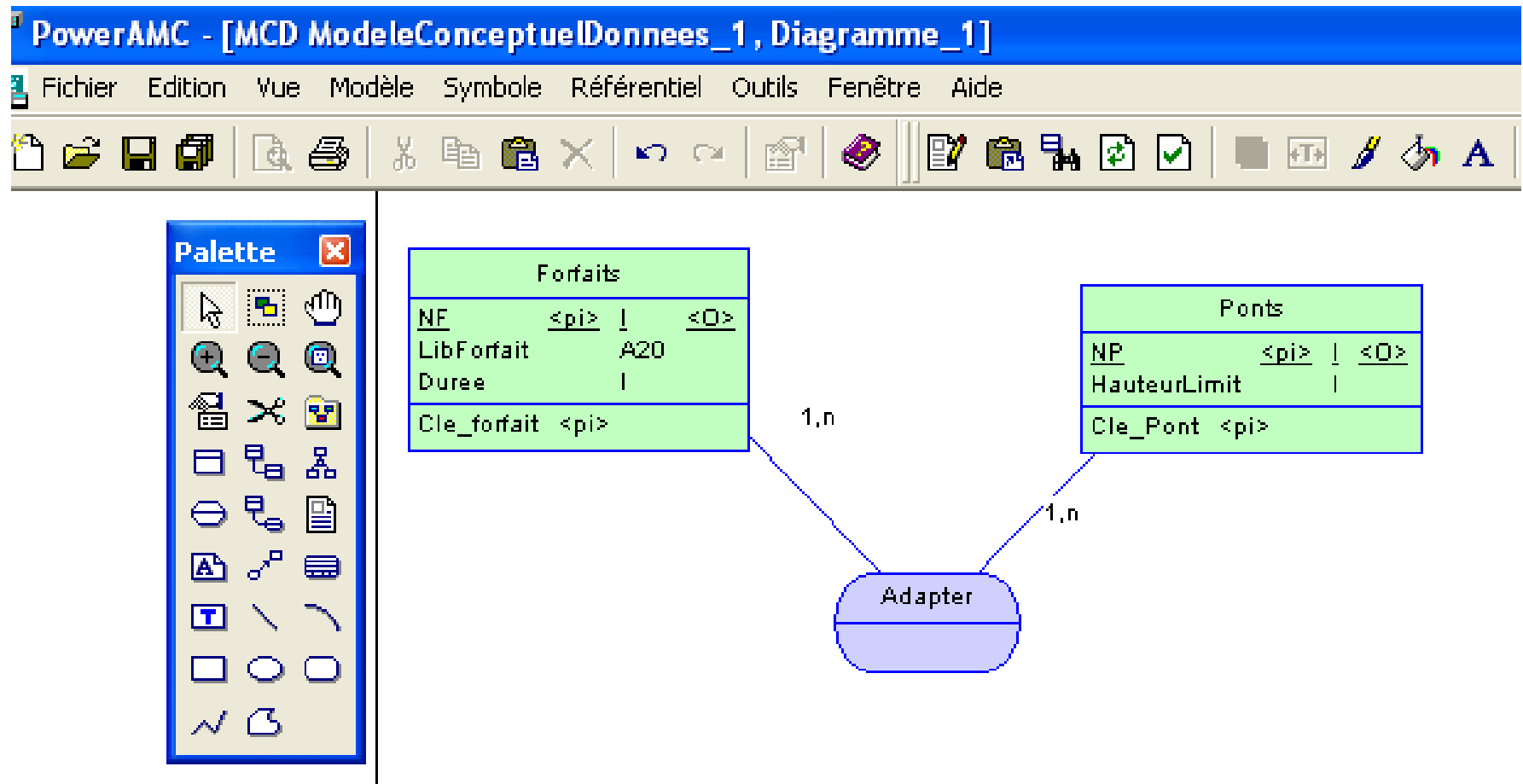
-- =====
--   Index : ADAPTER_PK
-- =====
create unique index ADAPTER_PK on ADAPTER (NF asc, NP asc);

-- =====
--   Table : FORFAITS
-- =====
create table FORFAITS
(
    NF                INTEGER                not null,
    LIEFORFAIT        CHAR(20)              ,
    DUREE             INTEGER
);
```

# Exemple d'AGL : Power AMC

## Modèle Conceptuel de Données

94

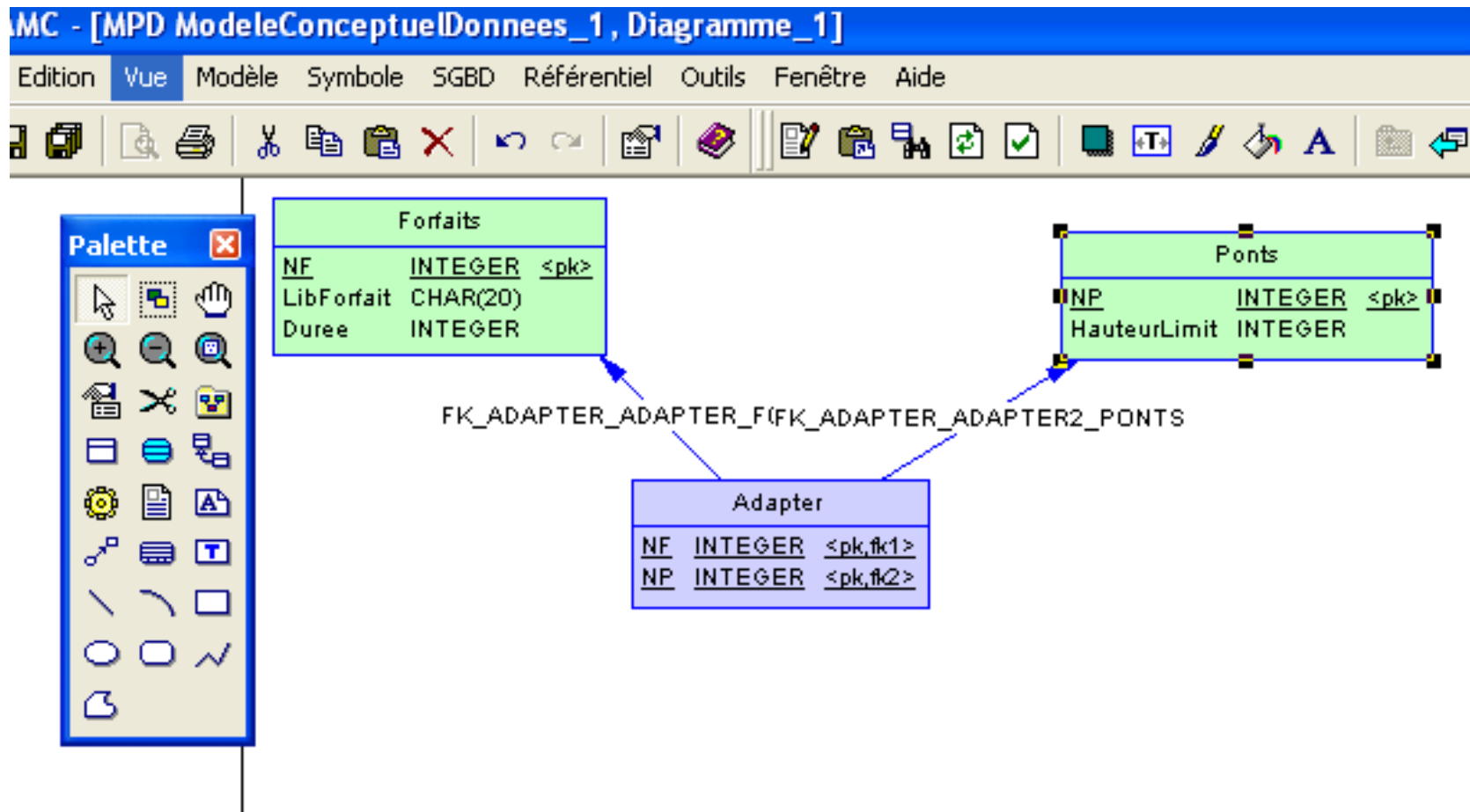


A. Madani (abdellah\_madani@yahoo.fr)

# Exemple d'AGL : Power AMC

## Modèle Physique de Données

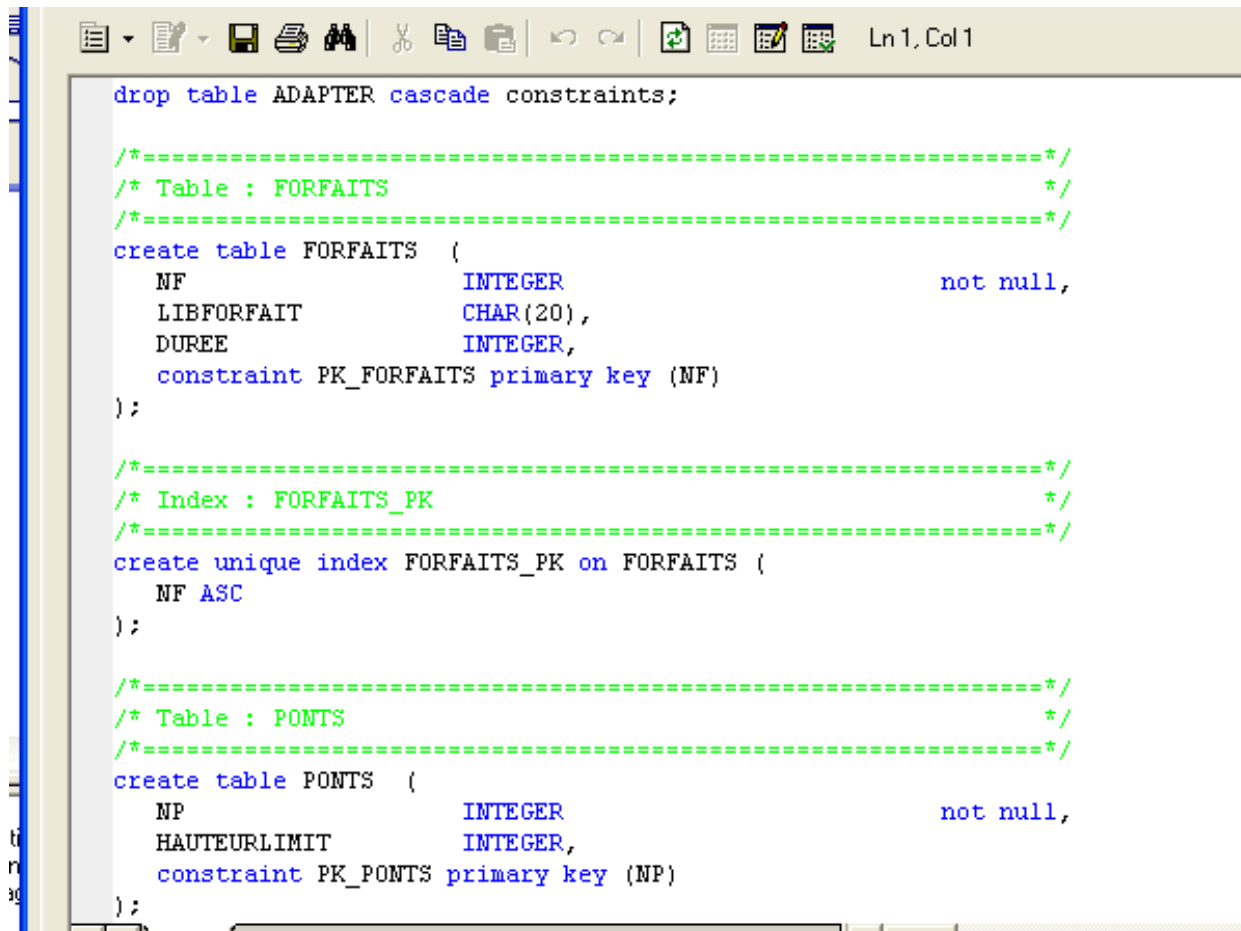
95



# Exemple d'AGL : Power AMC

## Script de création de Bases de Données

96



```
drop table ADAPTER cascade constraints;

/*=====*/
/* Table : FORFAITS */
/*=====*/
create table FORFAITS (
    NF                INTEGER                not null,
    LIBFORFAIT        CHAR(20),
    DUREE             INTEGER,
    constraint PK_FORFAITS primary key (NF)
);

/*=====*/
/* Index : FORFAITS_PK */
/*=====*/
create unique index FORFAITS_PK on FORFAITS (
    NF ASC
);

/*=====*/
/* Table : PONTS */
/*=====*/
create table PONTS (
    NP                INTEGER                not null,
    HAUTEURLIMIT      INTEGER,
    constraint PK_PONTS primary key (NP)
);
```