Formation INRA-ACTA-ICTA

Introduction à la modélisation

Les modèles mathématiques pour l'agronomie et l'élevage

2nde session, du 28 novembre au 1er décembre 2005

- Informatique et modèles -

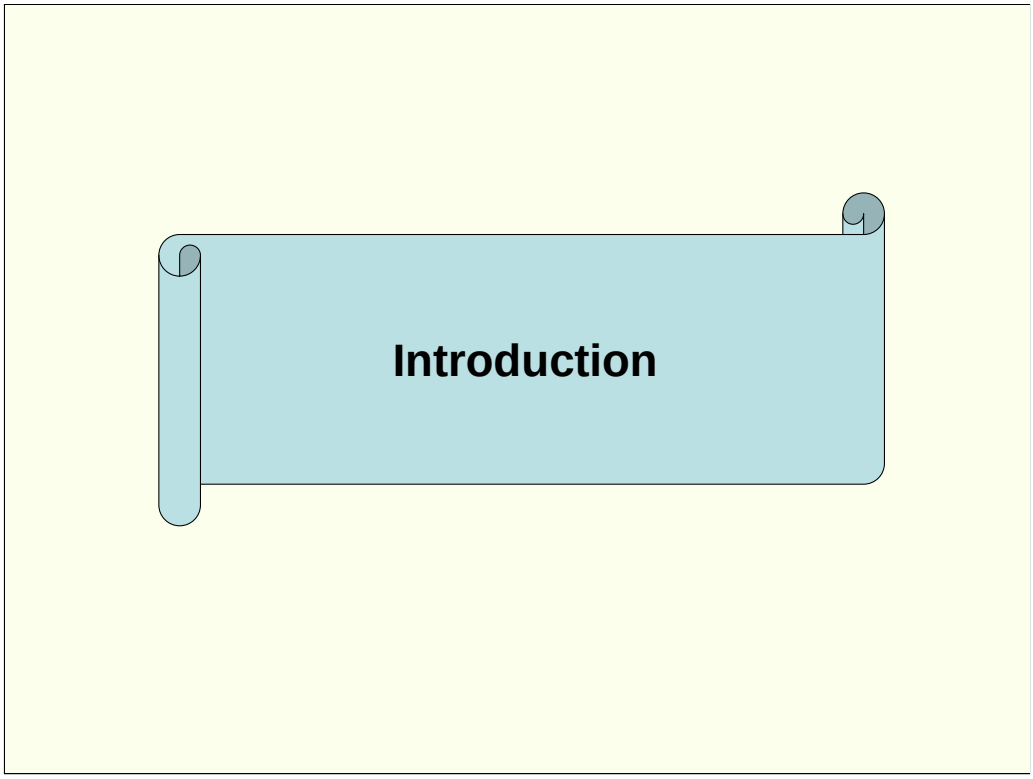
Nathalie Rousse ***Plate-forme INRA-ACTA-ICTA*** ***Décembre 2005***

http://www.modelia.org

Les choix informatiques à faire pour développer un modèle

Nathalie Rousse ***Plate-forme INRA-ACTA-ICTA*** ***Décembre 2005***

http://www.modelia.org


Introduction

Introduction

Méthode

→ **Démarche de développement**
Manière de mener le processus de développement

Technique

→ **Choix techniques informatiques**
Langages, outils, environnements de développement, architecture

Parmi les facteurs informatiques contribuant à la réussite d'un projet, il y a des aspects méthode et des aspects techniques.

La zone des commentaires apporte des précisions et des compléments par rapport au contenu des diapositives.

Introduction

Choix informatiques

Il n'y a pas de bons choix dans l'absolu mais
des choix bien adaptés aux besoins.

Les choix informatiques
dépendent des objectifs et des moyens du projet.

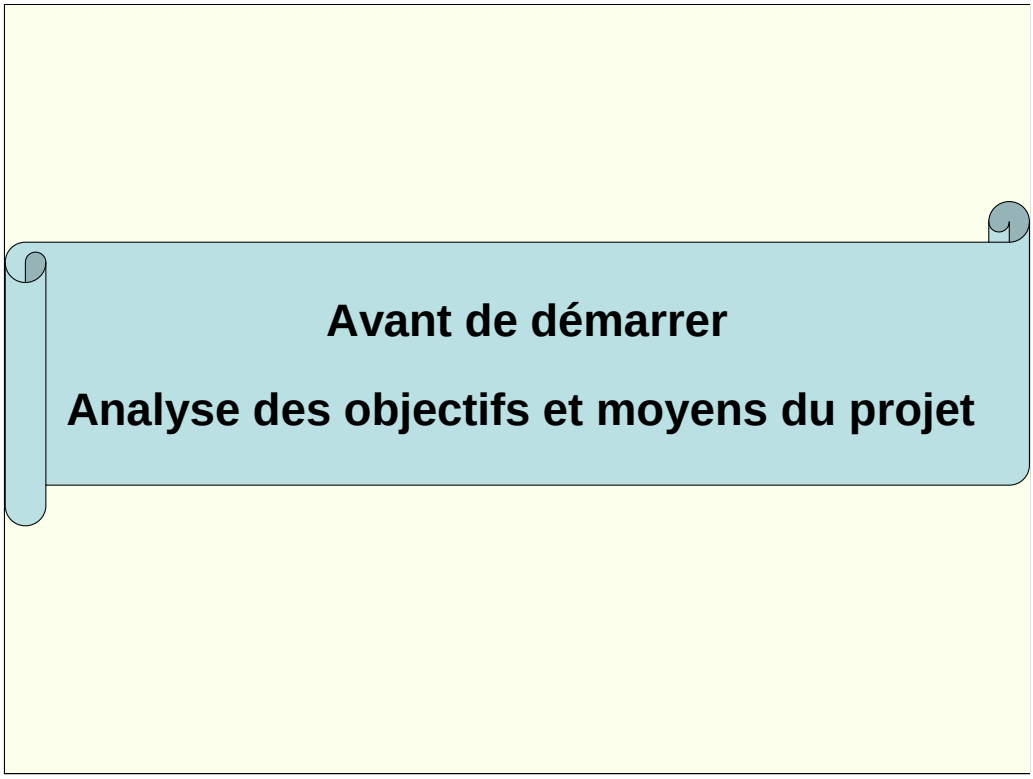
Avant de « se lancer » dans le développement,

Analyser les objectifs et moyens du projet

Les questions avant de démarrer le projet

Avant de se lancer dans le développement, de manière à faire de bons choix informatiques il est nécessaire d'analyser les objectifs et moyens du projet.

Ceci est utile quel que soit le projet, que la durée en soit de 15 jours ou 3 ans, la taille de l'équipe d'une ou 10 personnes. Seulement, l'analyse sera plus ou moins complexe, approfondie, longue (de quelques heures à plusieurs mois) en fonction de la taille, des ambitions et perspectives du projet.


Avant de démarrer
Analyse des objectifs et moyens du projet

Eléments d'analyse des objectifs et moyens du projet

1) Utilisateurs, utilisations

Qui sont les utilisateurs ? Quels sont leur niveau informatique, leur maîtrise du sujet scientifique, leurs préoccupations ?

2) Les suites potentielles

Evolutions potentielles du logiciel dans le futur (couplages, applications logicielles dérivées) ?

Des éléments d'analyse des objectifs et moyens du projet

Utilisateurs, utilisations

Qui doit pouvoir utiliser le logiciel (le concepteur lui-même, un novice …) ? Dans quelles mesures l'utilisateur maîtrise-t-il l'informatique, le sujet scientifique ? Quelles sont les préoccupations de l'utilisateur, sous quelles forme et conditions distribuer le logiciel ?

Certains auront besoin d'un outil convivial intégré à leur système habituel (par exemple Excel), d'autres auront besoin d'un outil en ligne de commandes, de manière à pouvoir automatiser facilement de multiples simulations. Certains seront intéressés par un applicatif, d'autres par du code source …

La suite

Sans qu'il en soit encore question à ce stade, de quelle(s) manière(s) le logiciel est-il susceptible d'évoluer (des possibilités de couplage à d'autres modèles, des idées d'applications logicielles qui pourraient être dérivées du modèle pour répondre à un besoin applicatif particulier …) ?

… suite page suivante

Eléments d'analyse des objectifs et moyens du projet

3) Environnement technologique

Plate(s)-forme(s) (Windows, Linux …), compatibilité de langages ?

4) Ressources humaines

Taille et répartition de l'équipe ? Compétences, disponibilité, statut des personnes ?

5) Calendrier

Durée de développement, échéances de livraisons, jalons, contraintes ?

<u>Des éléments d'analyse des objectifs et moyens du projet</u> *(…suite)*

Environnement technologique

Sur quelle(s) plate(s)-forme(s) le logiciel doit-il tourner (Windows, Linux …) ? Quels sont les langages avec lesquels il faut être compatible (pour pouvoir récupérer/appeler du code/un logiciel existant, en vue de se coupler à un autre modèle plus tard …) ?

Ressources humaines

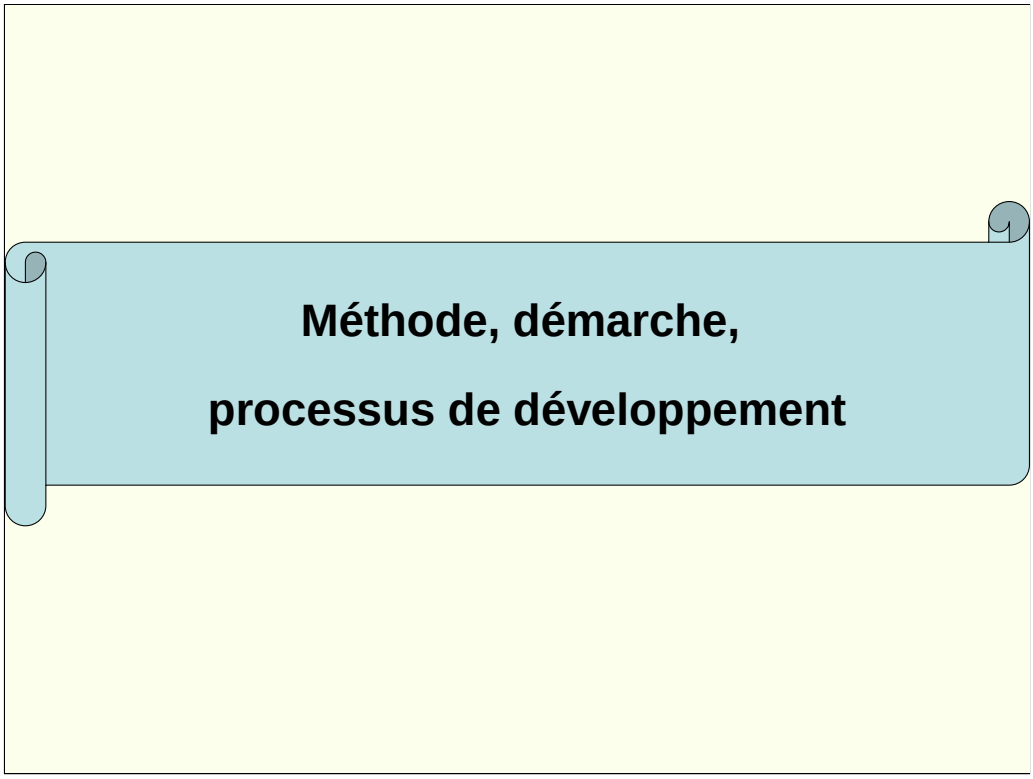
Quelles sont la taille et la répartition de l'équipe ? Quelles sont les compétences (informatiques …), les disponibilités et le statut des personnes (permanent, temporaire) ?

Calendrier

Quels sont la durée prévue du développement, les dates de début et de fin, les échéances de livraisons, jalons et contraintes ?

Conclusion

Quand on commence à réfléchir à ce qu'on veut faire, il arrive souvent de finir par souhaiter un logiciel qui couvre tous les cas, toutes les utilisations … Afin de faire les choix informatiques en tenant compte des futurs potentiels, il est conseillé de lister le plus exhaustivement possible ce qui est visé puis de hiérarchiser les priorités.


Méthode, démarche,
processus de développement

Processus de développement

Une démarche **méthodique**
de **développement informatique**
au profit des
viabilité et fiabilité du logiciel

Différentes approches du développement logiciel :
→ Processus **en cascade** (méthode traditionnelle).
→ Processus **itératifs** (méthodes agiles).

Processus de développement

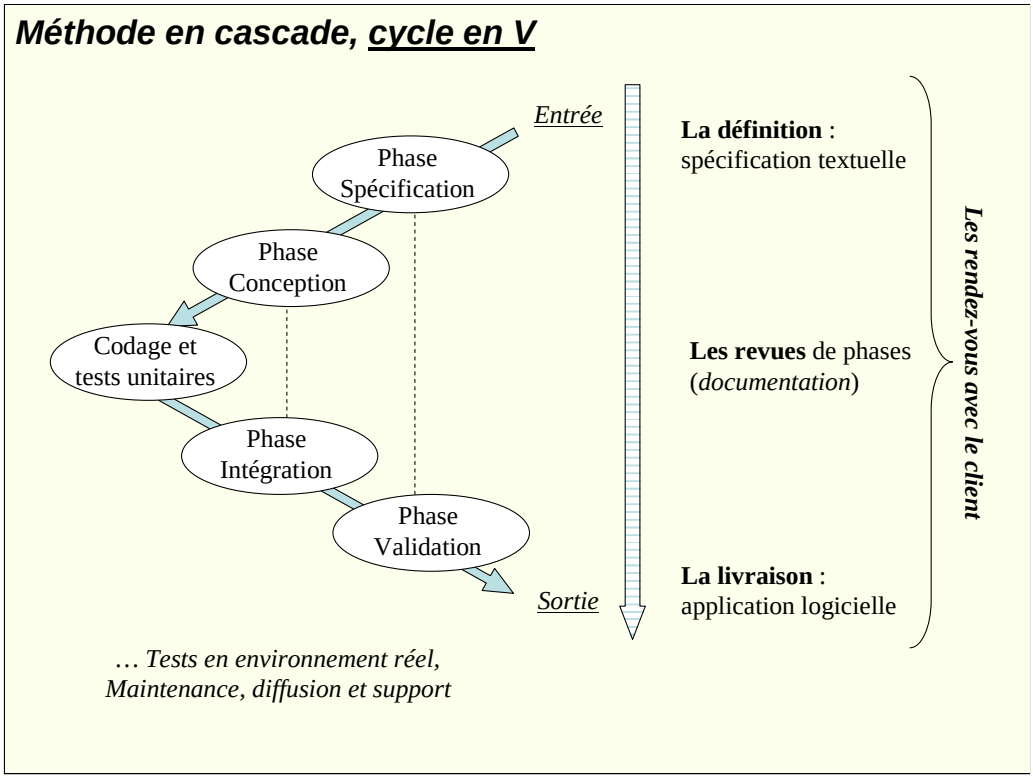
Puisqu'il est voué à subir des corrections et des évolutions, le logiciel est à penser dans la durée, non figé, en transformation constante. L'adoption d'une démarche de développement informatique méthodique contribue à la viabilité et la fiabilité du logiciel.

Il existe plusieurs méthodes de développement informatique :

- processus en cascade (méthode traditionnelle, à partir des années 70).
- processus itératifs (méthodes agiles, à partir des années 90).

Les deux approches se distinguent essentiellement dans la manière de décomposer le projet :

- Dans l'approche en cascade, le processus est découpé séquentiellement selon les activités intrinsèques du cycle de vie du développement logiciel : l'analyse des besoins, la conception, le codage et les tests.
- Dans l'approche itérative, le processus est découpé en itérations. Sur une itération, il est sélectionné un sous-ensemble des fonctionnalités, et déroulé les activités du développement logiciel d'implémentation de ces fonctionnalités (spécification, conception, codage et tests).



La méthode en cascade (cycle en V)

Description, caractéristiques de la méthode en cascade

En début de processus, il est élaboré le plan de déroulement des phases (planification prédictive). Le processus est ensuite déroulé linéairement, les phases se succèdent avec acceptation d'une phase, lors de la revue de phase, pour passer à la suivante. Dans les premières phases (1ère partie du V), sont préparées les phases de tests correspondantes (2nde partie du V).

En cours de cycle, les échanges avec le client reposent fortement sur la documentation, même si d'autres moyens de contrôle, d'échanges sont possibles (audits, maquettes …).

Hypothèses et application de la vision en cascade

L'approche en cascade part de l'hypothèse d'une expression de besoin stable et précise. Le fait que la livraison de l'application logicielle ait lieu en toute fin de cycle risque de générer des « malentendus » par rapport à l'expression de besoin : pour des questions de mauvaise compréhension et de changement d'avis au vu du résultat.

L'approche en cascade part de l'hypothèse de prévisibilité du processus de développement. Le fait que le processus soit composé d'une suite de phases difficilement mesurables risque de générer des dérives, un glissement de planning.

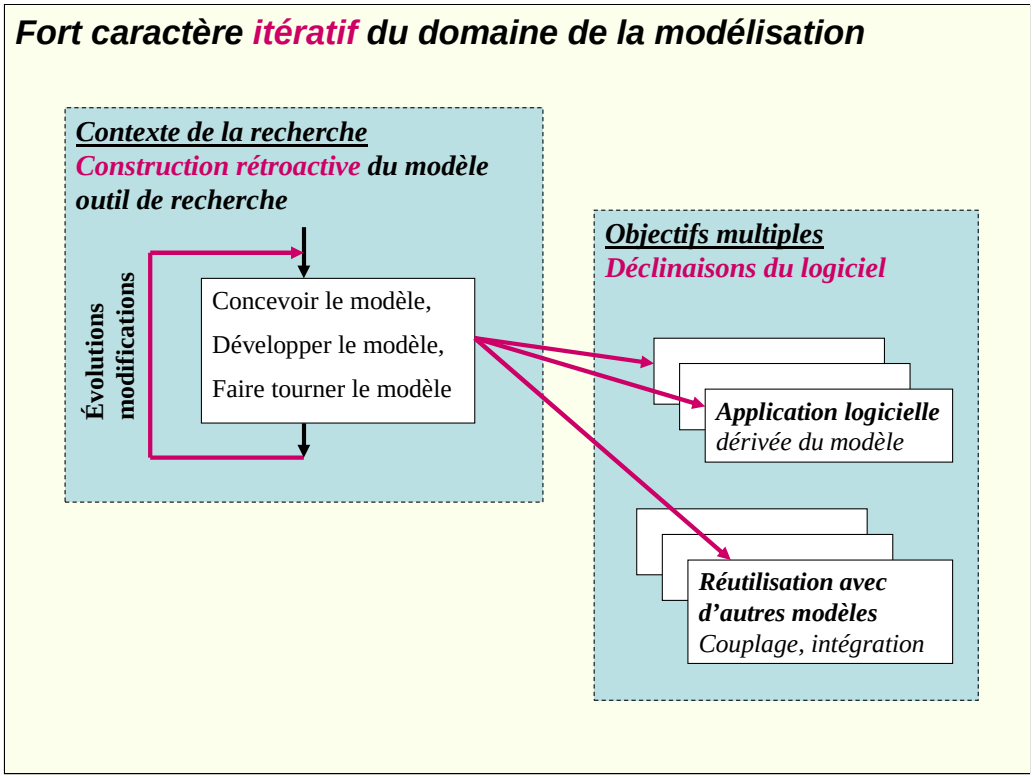
La méthode en cascade est adaptée aux cas où les exigences sont précises initialement, et ne sont pas sensées connaître de changements significatifs.

Cependant dans la pratique, les phases ne se succèdent pas complètement linéairement, il y a des retours en arrière. Il arrive qu'un problème détecté lors d'une phase (par exemple codage) nécessite de reconsidérer une des phases précédentes (par exemple revenir à la spécification et/ou la conception).

Le cycle en V

Le cycle en V est à la base de tout développement informatique, il en représente les activités intrinsèques.

Aussi le cycle en V est « présent » dans le développement logiciel, quelle que soit la méthode employée (méthode en cascade ou une autre).



Caractère itératif du domaine de la modélisation

Spécificités du développement de modèles

→ **Contexte de la recherche :**

Le modèle est défini de manière itérative, il est construit rétroactivement. Le fait d'en faire tourner une version inspire des modifications qui vont conduire à la version suivante, génère des idées d'évolutions.

→ **Objectifs multiples, déclinaisons futures du logiciel** :

Le modèle, outil de recherche en soi, est susceptible d'être couplé à d'autres modèles. Il peut aussi être à l'origine de multiples utilisations/outils/applications logicielles dérivées du modèle (développées à partir du modèle pour un besoin applicatif particulier).

L'approche itérative

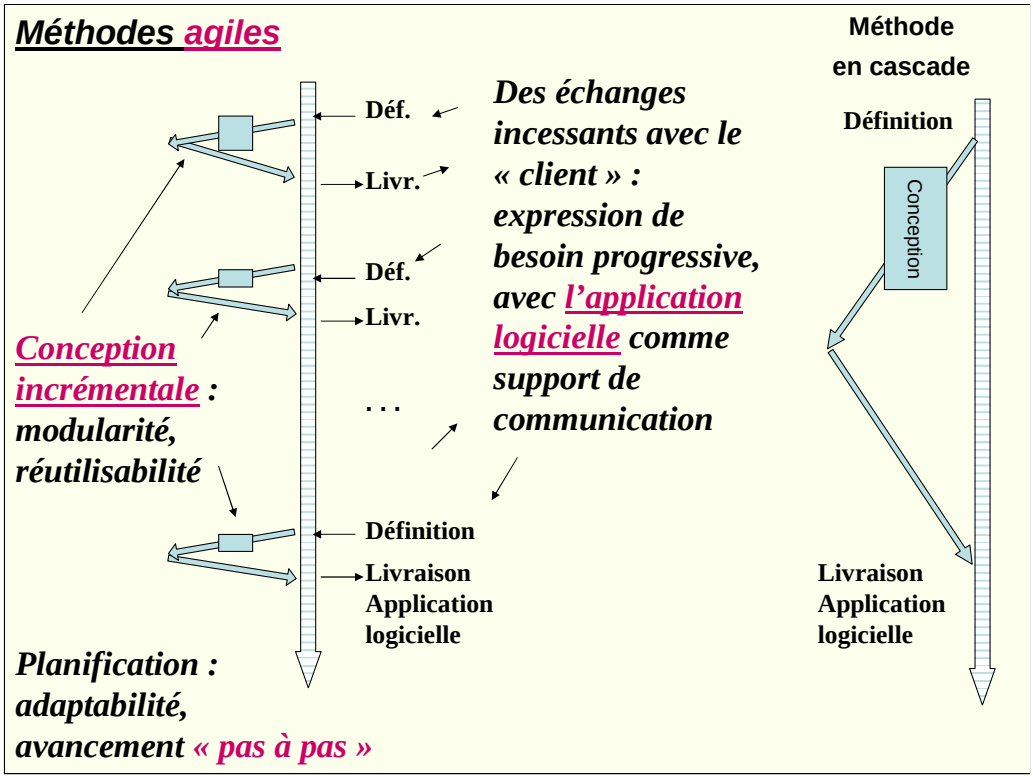
Le développement de modèles est souvent réalisé dans un cadre où les conditions requises pour appliquer la méthode en cascade ne sont pas vraiment remplies (instabilité des exigences, planification difficilement prédictive). D'autres approches s'avèrent plus appropriées : le développement itératif, adopté par les méthodes agiles.

Processus de développement itératif

Unified Process ou Processus Unifié (pour lequel outil d'IBM/Rational : **RUP**, Rational Unified Process) est un processus de développement de type itératif.

Il existe plusieurs processus de développement relevant des méthodes agiles : **XP** (eXtreme Programming), **FDD** (Feature Driven Development ou développement piloté par les fonctionnalités), **DSDM** (Dynamic Systems Development Method), **ASD** (Adaptive Software Development) ... etc

Vocabulaire associé au développement itératif : incrémental, en spirale, évolutif, …



Méthodes agiles

Une méthode agile est une méthode de développement informatique basée sur **4 valeurs fondamentales** :

- Equipe et communication : priorité aux personnes et interactions, sur les procédures et outils.
- Priorité aux applications fonctionnelles, sur une documentation détaillée.
- Priorité de la collaboration avec le client, sur la négociation de contrat.
- Acceptation du changement : réagir au changement, plutôt que suivre un plan.

Description, caractéristiques des méthodes agiles

Il y a continuellement des allers-retours avec le « client ». L'application logicielle est livrée par versions incrémentales : tôt et régulièrement, le plus fréquemment possible (toutes les semaines ou mois selon la méthode). Les versions successives ne couvrent qu'une partie des fonctionnalités demandées, mais sont aussi fiables que la livraison finale en terme de tests et validation.

C'est le fonctionnement de l'application (les versions logicielles opérationnelles plutôt que des documents) qui sert d'indicateur de la progression du projet.

Planification adaptative : la planification repose sur des plans à court terme stables à l'échelle de l'itération, et des plans à long terme fluides réadaptés au fur et à mesure de la progression. En quelque sorte le processus est déroulé comme un enchaînement de « mini-cascades ». La préférence est donnée au respect des délais (livrer l'application stable et validée en temps voulu, quitte à ce qu'elle ne remplisse que partiellement sa fonction) plutôt qu'à la complétude de l'application (ne livrer que quand l'application est complète, au risque de devoir pour cela retarder la date de livraison).

Une attention perpétuelle est portée à la conception logicielle ainsi qu'à la qualité du code (soin, robustesse, modularité). A chaque nouvelle itération, l'ensemble de l'architecture et de la conception logicielles est reconsidéré, le code est retravaillé (remaniement du résultat des itérations précédentes). On parle de refactorisation (refactoring).

Concernant le « message » agile

Les méthodes agiles prônent la documentation « utile » et non pas l'absence de documentation. Adaptabilité au lieu de prédictibilité ne signifie pas absence de planification. Les méthodes agiles sont adaptées aux cas où il n'y a pas dès le début des exigences précises et stables.

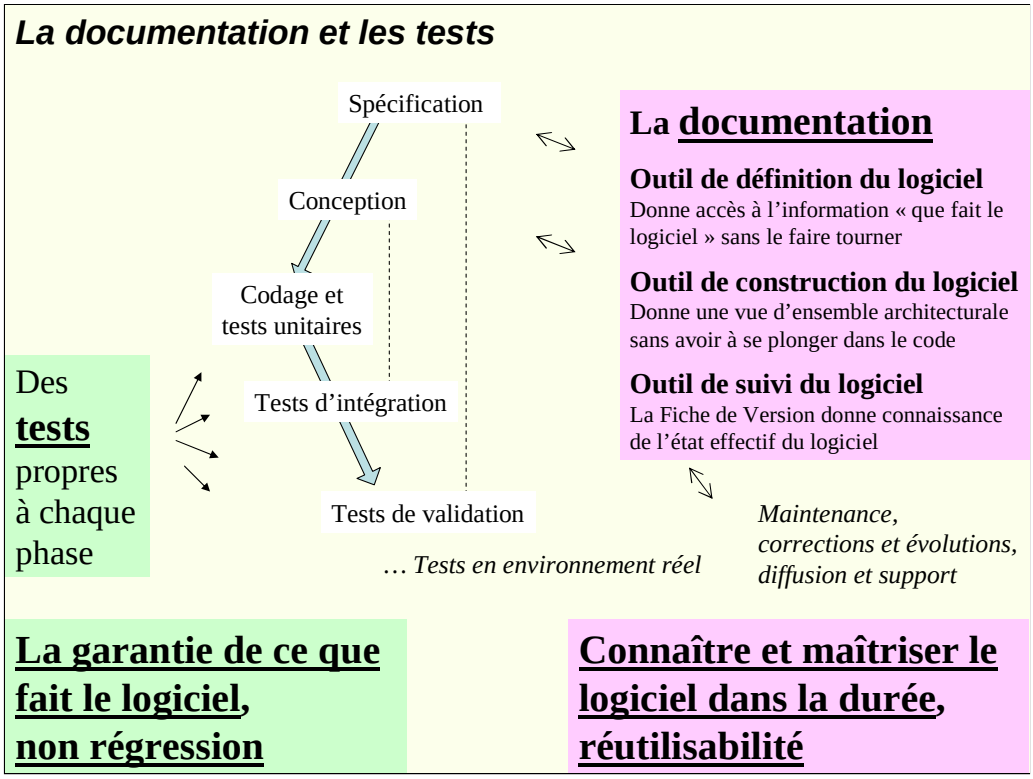
La documentation et les tests

La **documentation** et les **tests**,

des éléments clés des activités logicielles,

qui contribuent à

la **viabilité** et la **fiabilité** du logiciel



La documentation et les tests

La documentation et les tests accompagnent tout le cycle en V, tout le développement.

→ Quelques uns des rôles de la documentation du logiciel

Définition du logiciel

Le document de spécification décrit les exigences que le logiciel doit satisfaire. Il répond à la question « que doit faire le logiciel ? ». Cette documentation donne accès à l'information de ce que fait le logiciel, sans le faire tourner. Elle sert notamment à quelqu'un qui envisage d'appeler, de réutiliser le logiciel dans un autre développement, pour prendre connaissance des services rendus par le logiciel (sans se préoccuper de comment, dans un premier temps).

Construction du logiciel

En conception du logiciel, le travail de formalisation contribue à structurer le logiciel, élaborer la modularité. Cette documentation donne une vue d'ensemble architecturale sans avoir à se plonger dans le code. Elle aide à se repérer, quelqu'un qui doit travailler sur le logiciel, le reprendre. C'est dans ce sens que des informaticiens, pour s'approprier un programme informatique, ont parfois recours à des outils de rétro-ingénierie (« reverse engineering ») permettant de générer des diagrammes de conception à partir du code.

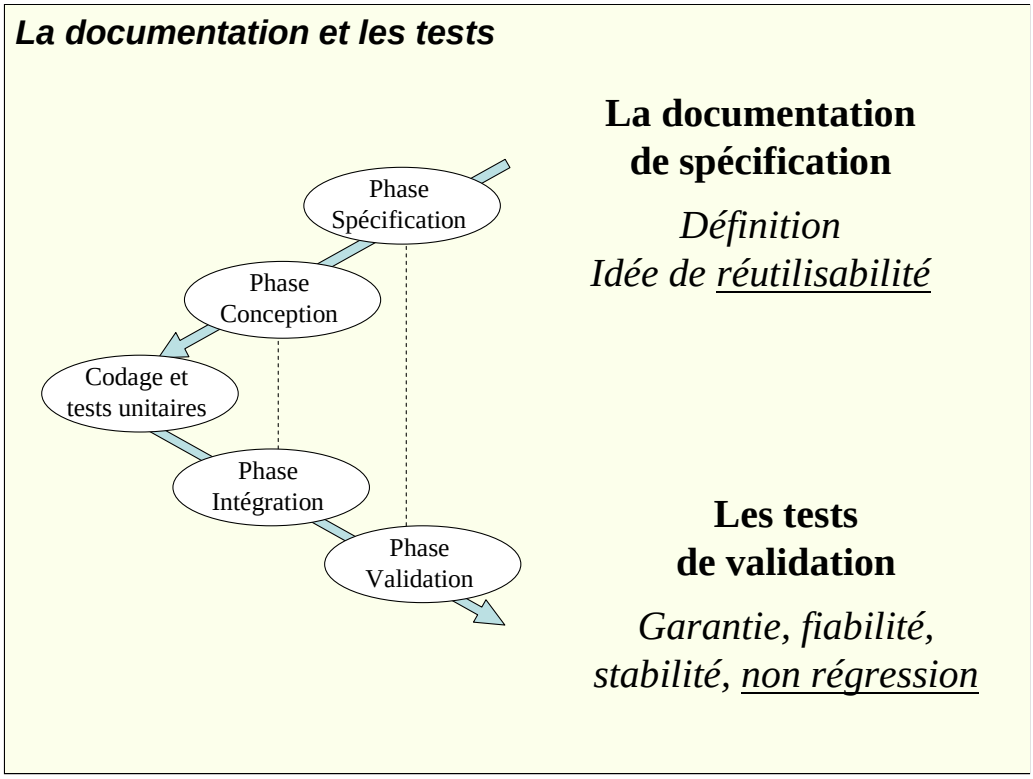
Suivi du logiciel

La fiche de version identifie et décrit une version stable du logiciel. Elle mentionne les modifications apportées par la version, ce qui est prévu mais pas encore implémenté, les anomalies connues … Cette documentation, accompagnant la livraison d'une version logicielle, donne connaissance de l'état effectif, réel du logiciel. De plus, afin de gérer les différentes versions du logiciel (liées/dues aux modifications dans le temps, s'il y a plusieurs développeurs…), les équipes informatiques ont recours à des systèmes de gestion de version tels que CVS (Concurrent Versions System).

Conclusion

Documenter le logiciel sert à le produire/construire, permet de le connaître et le maîtriser dans la durée, permet sa réutilisabilité (envisager de l'utiliser dans d'autres développements).

… suite page suivante



La documentation et les tests *(… suite)*

→ Les tests

La fiabilité et la qualité du logiciel passent par des vérifications propres à chaque phase.

Validation logicielle

En particulier, la validation logicielle (test « en bout de cycle ») vérifie que le logiciel répond bien aux exigences définies dans la spécification. La validation, en accompagnement de la livraison d'une nouvelle version logicielle, garantit ce que fait le logiciel.

La non régression

L'ajout, la modification de code impliquent de mettre en place de nouveaux tests vérifiant les changements apportés. Il est de plus nécessaire de rejouer les tests existants pour vérifier la non régression (s'assurer qu'on n'a pas dégradé l'existant).

Formalisation des tests

Un document de description des tests sert de support à l'exécution des tests, il décrit les procédures et les outils à mettre en œuvre, les scénarios de tests à passer. Un tel support trouve tout son intérêt dans la durée, quand il s'agit de vérifier la non régression (le « confort » maximal consistant à automatiser les tests).

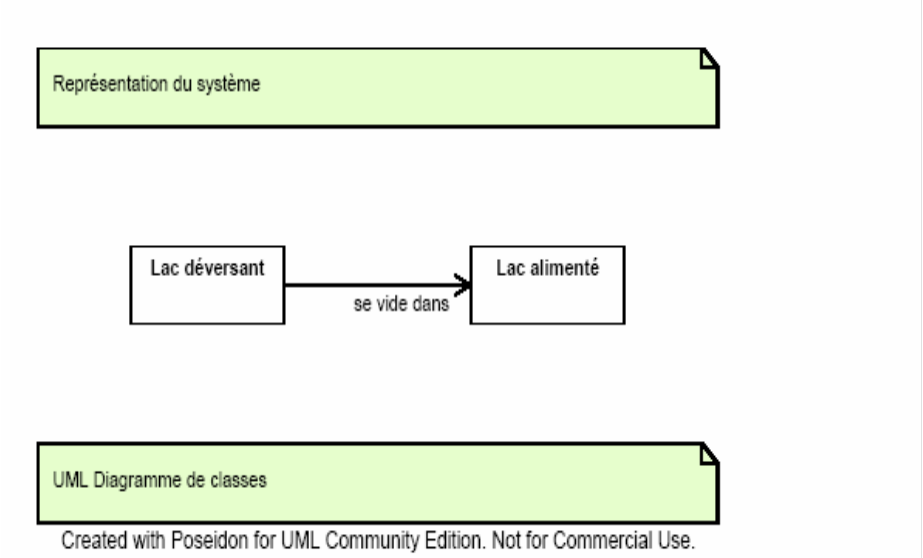
Quelques types de tests

Tests fonctionnels, tests « boite noire », tests d'interface, tests nominaux, tests aux limites, tests de performances, tests de robustesse, tests de sécurité …

→ Rôle majeur des documents de spécification et de validation

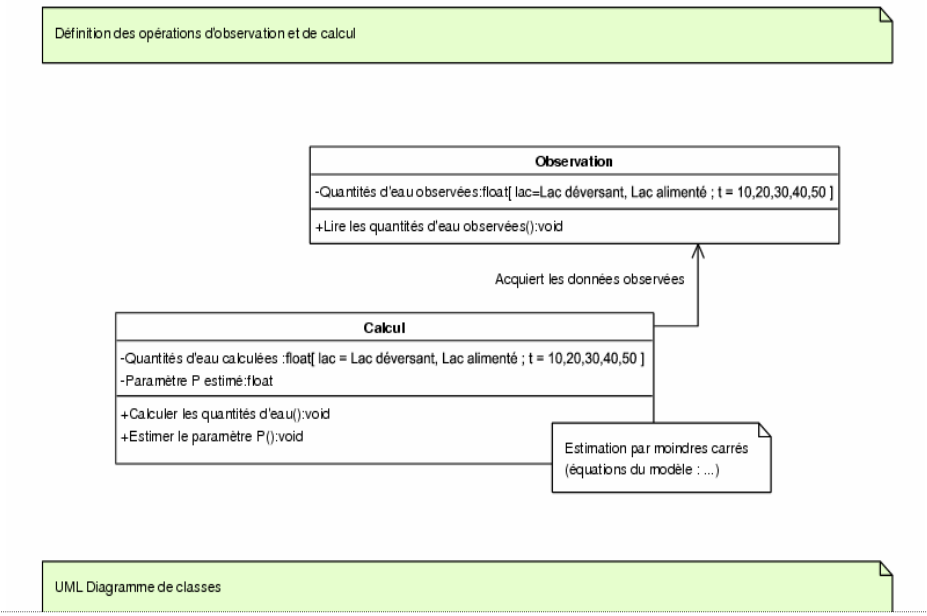
Les documents de spécification et de validation, en tant qu' « entrée / sortie » du développement, donnent une vue globale et externe du logiciel. Ils sont à privilégier en terme de formalisation, dans le sens où ils constituent des « points d'accès » au logiciel pour les personnes externes.

Documentation de définition – illustration –



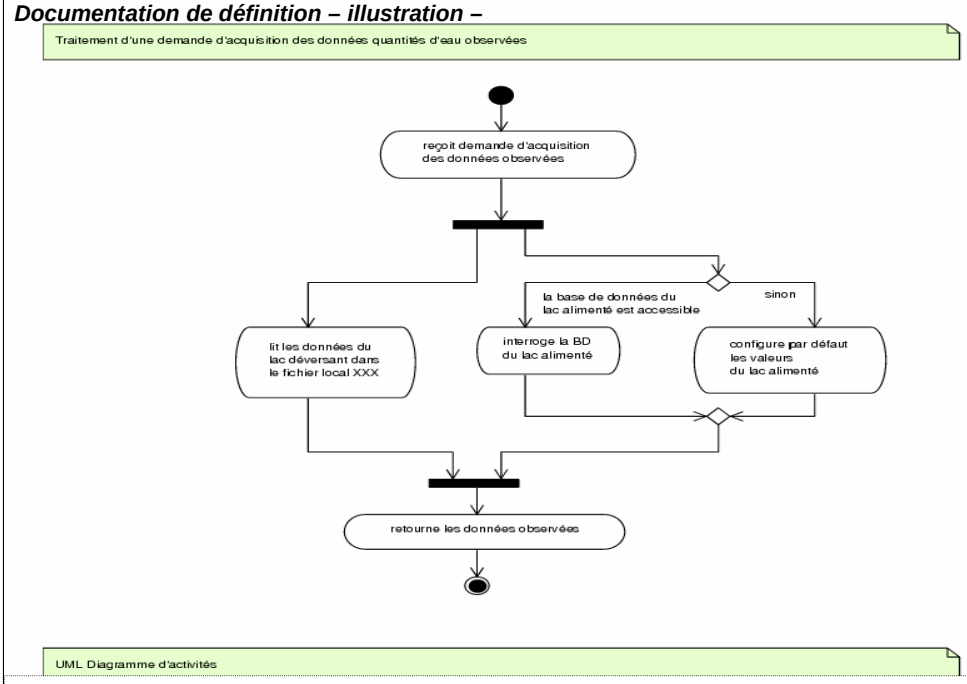
Modélisation conceptuelle du système, conceptualisation du modèle.

Documentation de définition – illustration -



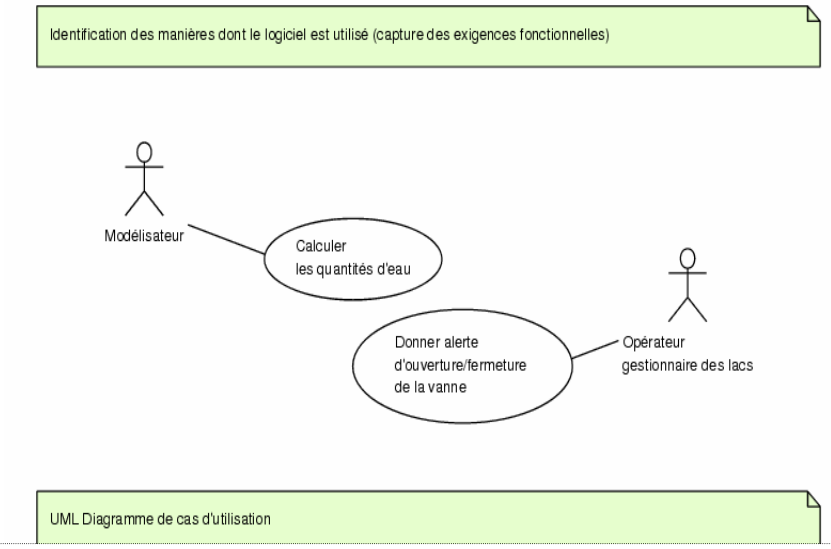
Identification des exigences, des fonctionnalités du logiciel.

Documentation de définition – illustration –



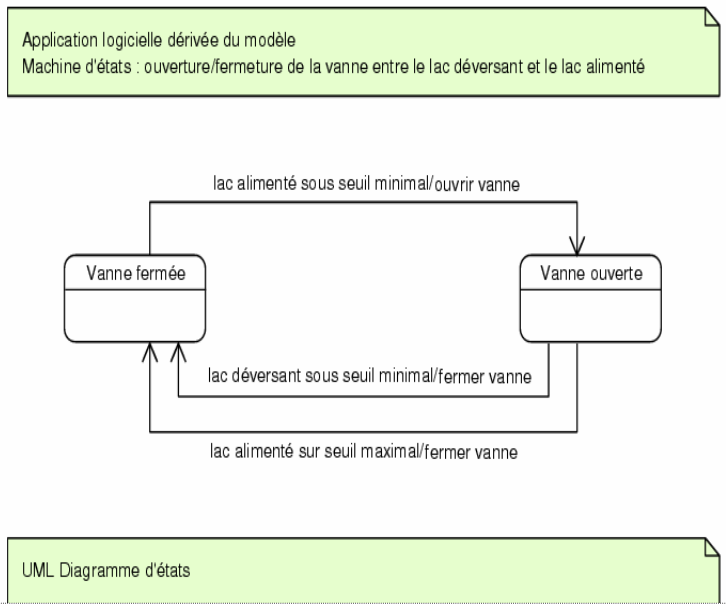
Exigences relatives aux traitements d'acquisition des données d'observation.

Documentation de définition – illustration –



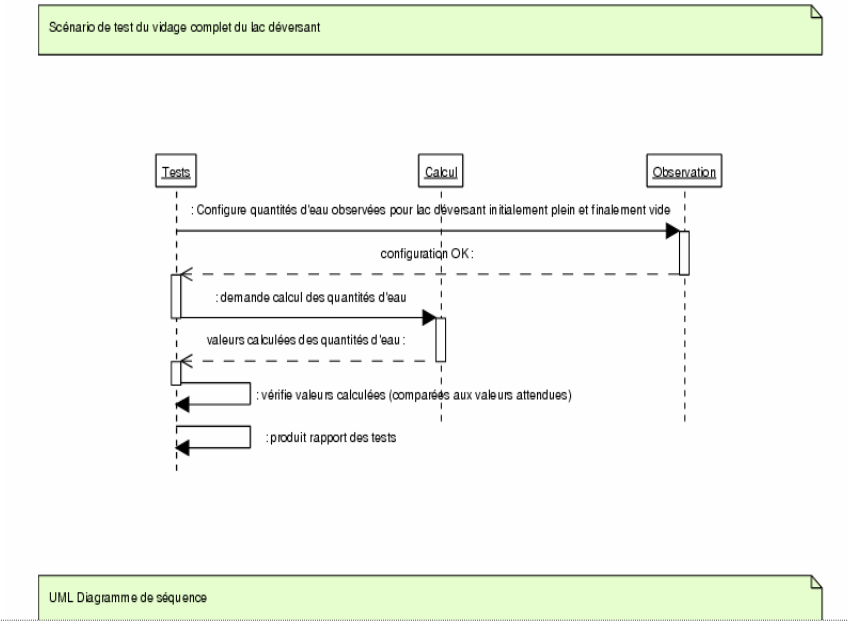
Le premier cas d'utilisation représente le contexte initial de développement du modèle : le modélisateur utilise le modèle dans le cadre de ses travaux de recherche.
Le second cas d'utilisation correspond à une évolution du logiciel : développement d'une application logicielle dérivée du modèle.

Documentation de définition – illustration -



Automate décrivant l'état de la vanne gérée par l'application logicielle qui a été greffée sur le modèle (voir en page précédente le second cas d'utilisation).

Documentation de test – illustration –



Un document de description des tests peut contenir un catalogue de scénarios tels que celui-ci et pour chaque fonctionnalité du logiciel, la liste des scénarios la vérifiant (à rejouer en non régression).

UML « Unified Modeling Language »

UML regroupe **une famille de notations graphiques**

permettant de décrire et de concevoir un système

(en particulier système logiciel orienté objet).

Utilisation d'UML par le modélisateur et par l'informaticien

→ Définition et conception du modèle, conceptualisation du modèle, modélisation conceptuelle de systèmes.

→ Spécification et conception logicielle.

En tant que formalisme **commun aux différents acteurs**,

UML **facilite les échanges, la communication, la compréhension**.

Intérêt et richesse d'UML

UML est le formalisme qui a été utilisé sur les illustrations précédentes.

Un formalisme commun de modélisation

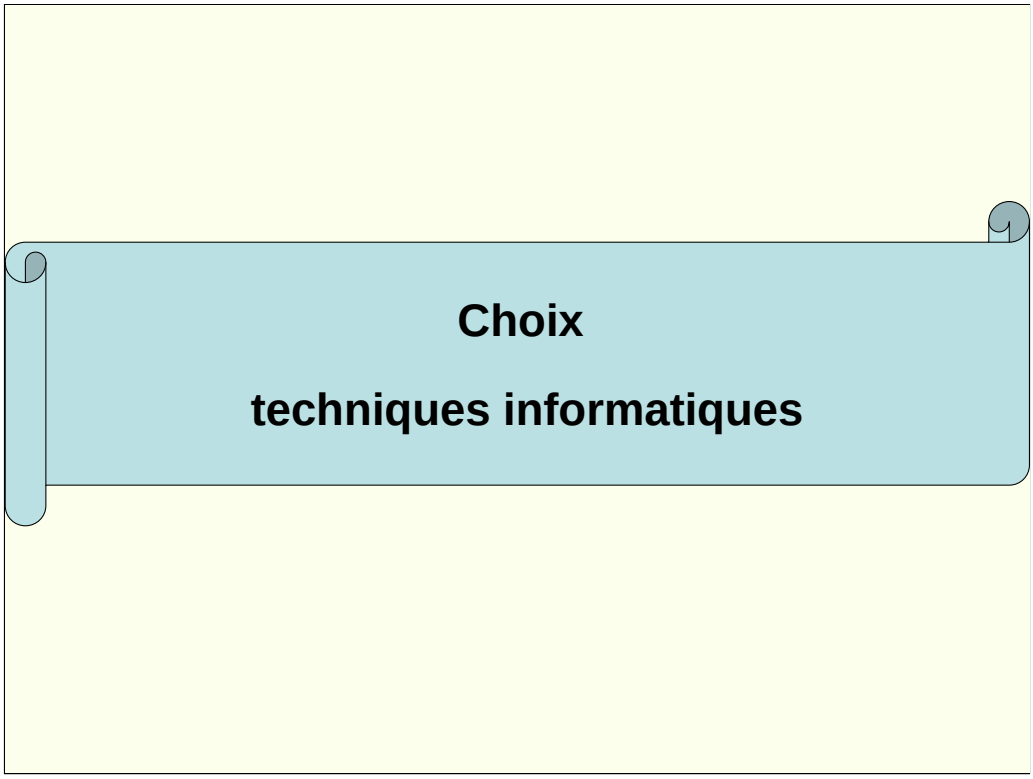
UML est un standard contrôlé par l'OMG (Object Management Group, consortium d'entreprises). UML est issu de l'unification de plusieurs langages de modélisation graphique des années 1980-1990. La première version UML standardisée par l'OMG est sortie en 1997. La dernière version, **UML 2.0**, en cours depuis mi-2001, a commencé à se stabiliser en 2003.

Il y a **13 types de diagrammes** dans **UML 2.0**. Les uns sont des diagrammes de structure servant à la description statique (**diagramme de classes**, diagramme d'objets, diagramme de déploiement, diagramme de package …), les autres sont des diagrammes de comportement servant à la description dynamique (**diagramme de séquences**, diagramme de cas d'utilisation, diagramme de machines d'états, diagramme d'activités …).

Utilisation d'UML par le modélisateur et par l'informaticien

→ UML du point de vue conceptuel (modèle, système) : définition et conception du modèle, conceptualisation du modèle, modélisation conceptuelle de systèmes.

→ UML du point de vue logiciel : spécification et conception logicielle.


Choix
techniques informatiques

Choix techniques informatiques

L'environnement technique informatique
auquel on a recours pour construire un logiciel
couvre des **langages de programmation**,
des **outils** et **environnements de développement**,
des **technologies informatiques** …

Dans la suite de la présentation, un « **objet** » désignera une composante d'environnement technique informatique telle qu'un langage de programmation, un outil ou un environnement de développement, une technologie informatique …

Les « critères de choix techniques informatiques » qui seront présentés dans la suite montrent différents facteurs qui entrent en compte lorsqu'il s'agit de choisir des « objets » pour développer un modèle. Ces critères sont illustrés par quelques exemples *en italique*.

Environnement technique informatique - des exemples -

Langages de programmation Fortran, C, java, C++.

Outils et macro-langages S-plus, R.

Outils Excel, ModelMaker, Stella, Vensim.

Langages interprétés python, perl, php.

Outil de compilation gcc.

Environnements de développement et langages Matlab, Scilab.

EDI (Environnements de développement intégrés) Delphi, Eclipse.

Technologies XML, services Web.

Environnement de programmation de SMA (Systèmes Multi-Agents) Cormas.

Pour un programme écrit dans un **langage interprété**, l'interpréteur (programme auxiliaire installé sur la machine) traduit au processeur de la machine les instructions du programme, au fur et à mesure du déroulement de l'exécution.

Pour un programme écrit dans un **langage compilé**, avec l'opération préalable de **compilation** il est généré en langage machine un fichier exécutable. Ce fichier exécutable est directement intelligible et exécuté par le processeur de la machine.

Un **EDI** (Environnement de Développement Intégré) est un programme regroupant un éditeur de texte, un compilateur, des outils automatiques de fabrication, et souvent un débogueur. On peut également trouver dans un EDI un système de gestion de versions et différents outils pour faciliter la création de l'interface graphique (*GUI* en anglais pour *Graphical User Interface).*

Exemples d'EDI : Visual Basic, Visual C++, Visual Studio .NET (pour C#, VB.NET) de Microsoft. Delphi (pour Pascal / Windows …), Kylix (pour Pascal / Linux), JBuilder (pour Java) de Borland. Eclipse (EDI du monde du logiciel libre pour Java, C++, Python).

Un **service Web** (Web Services) est un ensemble de protocoles et de normes informatiques utilisés pour échanger des données entre les applications. Les logiciels écrits dans divers langages de programmation et sur diverses plates-formes peuvent employer des services Web pour échanger des données à travers des réseaux informatiques comme Internet. Cette interopérabilité est due à l'utilisation de normes ouvertes regroupées au sein du terme générique de SOA (Service Oriented Architecture, ou Architecture Orientée Services). Standards employés : Web Services Protocol Stack, XML, http, ftp …

XML (eXtensible Markup Language, ou Langage de balisage extensible) est un standard défini par le W3C (World Wide Web Consortium), qui sert de base pour créer des langages de structuration de données spécialisés et personnalisés. Le XML est ainsi ce qu'on appelle un « méta langage », mais aussi un format de fichiers. Les langages basés sur XML (dialectes XML) permettent de décrire, manipuler, traiter et communiquer toutes sortes de données et de textes.

Sources principales :

Wikipédia, l'encyclopédie libre (URL http://fr.wikipedia.org/wiki).

Le Dico du Net, dictionnaire collaboratif en ligne (URL http://www.dicodunet.com).

Des critères de choix techniques informatiques

1) Prise en main

Il s'agit d'évaluer la facilité de prise en main,
la praticité, la complexité de l'objet

<u>Critère : Prise en main</u>

Niveau de compétences informatiques requis, accessibilité pour non informaticien

Praticité

Facilité/simplicité d'utilisation et de prise en main, ergonomie, convivialité. Facilité/complexité d'installation.

Appel de l'objet via une interface graphique ou par commandes en ligne.

Connaissance préalable et personnes ressources

Connaissance/expérience de l'objet par l'utilisateur ou des collègues.

Des critères de choix techniques informatiques

2) Adapté aux besoins scientifiques

Il s'agit d'évaluer si l'objet répond aux préoccupations du modélisateur, s'il est adapté au besoin du modélisateur en terme de **construction** et de **mise au point du modèle**.

Critère : Adapté aux besoins scientifiques

Adapté / spécialisé dans la discipline, la thématique scientifique, le type de modèle

Disponibilité de bibliothèques scientifiques spécialisées (*statistique, calcul scientifique, …*).

Excel pour modèles empiriques. CORMAS pour modélisation SMA (Système Multi-Agents).

Moyens de test et mise au point du modèle

Des méthodes/outils de tests, évaluation, validation, mise au point de modèle sont ou non fournis avec l'objet (analyse de sensibilité, estimation de paramètres …).

Moyens de mise en œuvre

Performances techniques au moment de l'utilisation du modèle produit (rapidité d'exécution, procédure de lancement des simulations).

Les langages compilés (C++, Fortran …) plus rapides que les langages interprétés (Matlab, Python, S-plus …).

Des critères de choix techniques informatiques

3) Adapté aux besoins informatiques

Il s'agit d'évaluer si l'objet répond à des préoccupations informatiques en terme de qualité informatique,
évolutivité et **interopérabilité**
du logiciel qui est développé.

Critère : Adapté aux besoins informatiques

Moyens de développement

Services et facilités de développement proposés en terme informatique. Génération automatique de code source, de documentation. Moyens de test et mise au point du logiciel (débogueur).

Richesse des EDI (Environnements de développement intégrés), graphiques Excel, macro-instructions.

Des outils UML permettent de générer du code (java, C++ …) à partir des diagrammes UML de conception logicielle.

Performances et qualité

Puissance, sophistication (richesse syntaxique …), permissivité d'un langage de programmation.

Les langages basés sur la conception orientée objet (Java, C++, Python …) favorisent la modularité.

Les langages compilés garantissent un certain nombre de vérifications (syntaxe) avec l'étape de compilation.

Interopérabilité, ouverture, portabilité, compatibilité, diffusion

→ Aptitude du logiciel produit à être utilisé dans divers contextes, à être appelé par d'autres logiciels et à appeler d'autres logiciels ; compatibilité de l'objet avec d'autres objets (ouverture vers d'autres projets logiciels).

Langages normalisés : C, C++ (norme ISO). Langage propriétaire : Matlab.

Outils propriétaires : Matlab, Excel. Outil ouvert/licence libre : Scilab.

Format fermé : .xls. Format ouvert : XML (recommandation W3C).

Caractère multi-plateforme (Windows, Unix ...) du langage Java.

Appel de code C ou C++ par du code Java (via Java Native Interface).

→ Aptitude à la diffusion des applications produites : outil donnant ou non moyen de générer un exécutable autonome (pour lancer l'application hors outil), ou bien l'exécutable est lié à l'outil de développement. Y a-t-il moyen ou non d'empêcher de modifier le code source de l'application produite ?

Des critères de choix techniques informatiques

Interopérabilité

L'interopérabilité est le fait que plusieurs systèmes, qu'ils soient identiques ou radicalement différents, puissent communiquer sans ambiguïté et opérer (travailler) ensemble. L'interopérabilité nécessite de se conformer à des normes clairement établies et univoques.

Choix d'architecture et conception logicielles

Les choix d'environnement technique informatique s'intègrent plus globalement dans les choix d'architecture et conception logicielles.

En fait, les choix d'environnement technique informatique (langages, outils, environnements de développement, technologies informatiques) s'intègrent plus globalement dans les choix d'architecture et conception logicielles.

A titre d'illustrations

Conception d'un logiciel qui appelle un modèle, ou qui intègre plusieurs modèles

La structure/organisation logicielle suivante est choisie pour tirer le meilleur de chaque langage :

- Coder le cœur du modèle en langage évolué (C++, C, Fortran…) : langage puissant, performant, adapté à des traitements complexes.
- Pour coder une couche d'appel du modèle ou une couche faisant l'interface entre plusieurs modèles, choix d'un langage interprété (Python...) : langage simple/facile d'utilisation, jouant alors le rôle de « glue ».

Une architecture basée sur des services Web

Les services web fournissent l'interopérabilité entre logiciels fonctionnant sur diverses plates-formes. Cette solution évolutive facilite la diffusion et la maintenance.

Des critères de choix techniques informatiques

4) Aspect commercial et confiance dans le produit

Il s'agit d'évaluer si l'objet est de confiance,
pérenne, stable, fiable,
ses conditions et droits d'utilisation.

Critère : Aspect commercial et confiance dans le produit

Fiabilité, pérennité (âge, parts de marché, « réputation »).

Editeurs et distributeurs (santé, stabilité).

Supports (assistance utilisateur, tutoriels existants …).

Conditions d'utilisation (licence, libre/propriétaire, modalités de distribution …).

Coût (tarifs des licences de développement et de redistribution).

Des critères de choix techniques informatiques

Comment évaluer ces critères ?

→ **Etudes techniques**

→ **Maquette d'évaluation**

Réaliser une maquette de petite dimension, simple mais représentative de la problématique, afin d'évaluer les performances, d'étudier la faisabilité.

→ **Retours d'expérience**

Recueillir des retours d'expérience et avis d'utilisateurs, développeurs.

Fiches techniques
diffusées par la plate-forme INRA-ACTA-ICTA
http://www.modelia.org

UML,
Aspects informatiques juridiques,
Interopérabilité, ouvert, norme, standard,
Environnements de développement informatiques, outils, langages, technologies,
Couplage, intégration de modèles,
Méthodologie, processus de développement, méthode en cascade, méthodes agiles,
Documentation : la documentation d'un projet informatique,
Le développement d'un produit logiciel …

Des références

Méthode en cascade, méthodes agiles, UML.

Références méthode en cascade

Ouvrage « Ingénierie et qualité du logiciel et des systèmes », AFNOR, ISBN 2-12-236141-7 : recueil de normes dans le domaine de l'ingénierie du logiciel et des systèmes.

En particulier la norme française : NF ISO/CEI 12207:1995 (Traitement de l'information, Ingénierie du logiciel, processus du cycle de vie du logiciel).

Références méthodes agiles

→ En 2001, les instigateurs des principales méthodes agiles se sont réunis pour former l' « Agile Alliance ».

Ils ont défini un manifeste pour le développement agile d'applications.

Alliance Agile : http://www.agilealliance.org ; « Manifesto for Agile Software Development » : http://agilemanifesto.org

→ Wikipedia « Méthode agile » : http://fr.wikipedia.org/wiki/Méthode_agile

→ Traduction d'un article de Martin Fowler sur les méthodologies agiles : http://www.martinfowler.com/articles/frNewMethodology.html

→ « Extreme Programming, Méthodes Agiles, Tour d'horizon » par Business Interactif

http://www.dsi.cnrs.fr/bureau_qualite/developpement-web/methodologie/BI-methodes-agiles.pdf

→ Dossier de « La Cible n°101 – septembre 2004 » : « les méthodologies informatiques Agiles » par Jean Guillaume Lalanne, Cap Gemini Ernst & Young.

Références UML

Le site officiel par l'OMG : http://www.uml.org

Ouvrage « UML 2.0 » de Martin Fowler. Campus Press. ISBN : 2-7440-1713-2. Ouvrage en français, traduit de l'ouvrage américain « UML Distilled – Third Edition ».

Ouvrage « UML 2 Initiation, exemples et exercices corrigés » de Laurent Debrauwer et Fien Van der Heyde.Collection Ressources Informatiques, éditions ENI. ISBN : 2-7460-1455-6. http://www.guideuml2.fr