Chapitre 5

Vision Informatique Logique
Architectures Applicative et Logicielle

NFE107

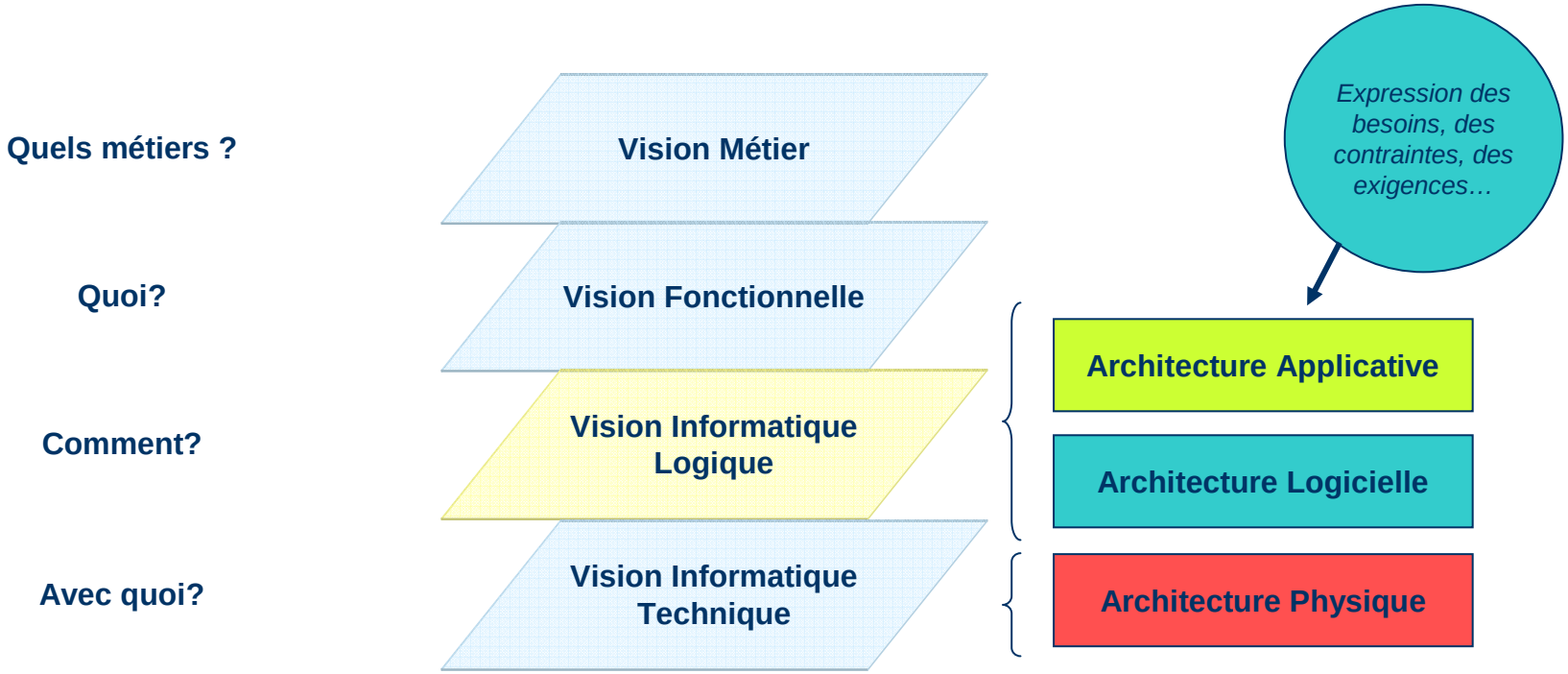
Chapitre 5

Vision Informatique Logique Architectures Applicative et Logicielle

5.1 Introduction

Positionnement de la Vision Informatique

- Urbanisation / Architecture



Introduction

- Objectifs :

- Architecture Applicative
  - elle structure le SI en blocs applicatifs communicants
  - elle décrit sous l'angle technique les applications, les flux et les messages échangés entre applications

- Architecture Logicielle
  - elle se consacre à structurer et à concevoir une application à partir de ses spécifications fonctionnelles
  - elle structure et décompose de façon logique chaque application en couches
  - elle introduit les notions et concepts de découpage en couches, composants, framework et design patterns

Elle répond à la question du COMMENT ?

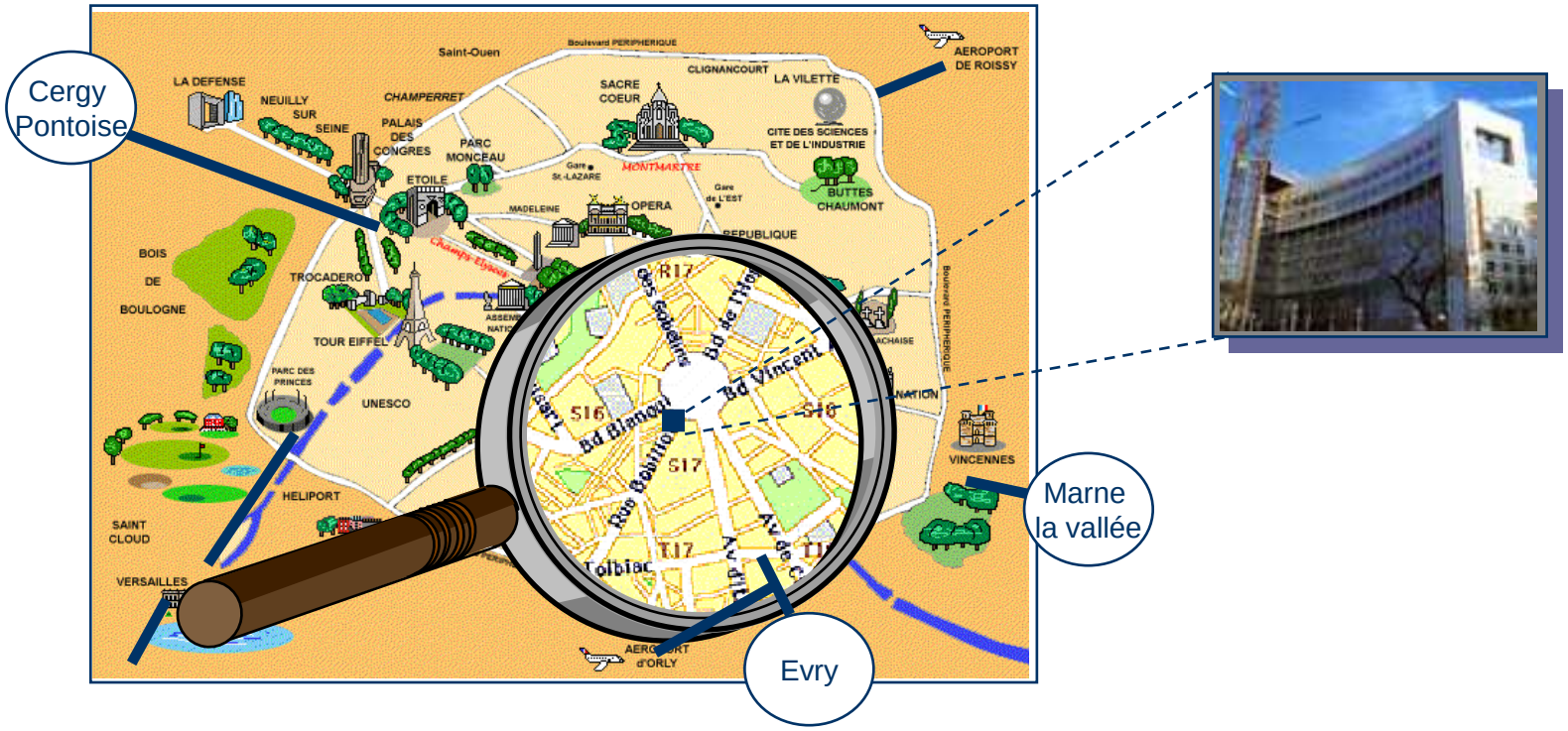
Introduction

- Selon « Le Larousse », l'architecture est « l'art de concevoir et de construire un bâtiment selon des partis esthétiques et des règles techniques déterminés »
- A l'instar d'un bâtiment, une application informatique est construite dans le but de remplir une fonction bien précise
- Ce terme du génie civil, les informaticiens se le sont appropriés pour décrire leurs activités liées à la construction de systèmes informatiques

La métaphore de l’urbanisme
Exemple de la ville

1 - se donner une vision globale et cohérente
2 - décomposer la complexité
3 - normer les découpages
4 - rapprocher les centres de décision du terrain

POS ville/agglomération	⟷	Cadre du système d ’information Règles d ’urbanisme
Arrondissements Quartiers/îlots	⟷	Découpage en sous ensembles
Mairie	⟷	MOA / MOE d’un sous-ensemble



Parallèle entre le génie civil et l'architecture logicielle

Génie civil	Architecture logicielle
Structure	Couches
Plan	Diagrammes (matérialisation des différentes structures, plans de l'application)
Matériaux	Briques informatiques – Librairies – Composants sur étagères (Composants et Frameworks) – Design Patterns (motifs de conception)
Outil	Outils de conception et de développement
Procédé	Méthodes de développement informatiques L'architecte doit les recommander ou les intégrer de façon à conserver une cohérence dont il se porte garant

Méthodologies d'Architecture (1/2)

- L'architecture a ses écoles, ses styles, ses courants …

- Dans le domaine de l'architecture de SI aucune méthodologie n'a réussi à s'affirmer avec succès. Ces méthodologies se sont le plus souvent limitées à des projets spécifiques sans parvenir à se généraliser à l'échelle du SI
  - Approche « top down » (du processus au code), avec deux courants principaux:
    - Approche « Données/Traitements » (Zachman, Merise…) : l'approche « Données/Traitements » centre l'analyse d'un problème sur la donnée manipulée;
    - Approche « Composants » (RM-ODP, Catalysis…) qui adresse plus spécifiquement les architectures des systèmes distribués : ce modèle a été élaboré sous l'influence du framework Zachman mais guidé par le paradigme Orienté Objet.
    - Ces deux courants n'ont pas réussi à s'imposer pour deux principaux griefs :
      - le dogmatisme : la croyance dans une démarche top-down séquentielle (de la stratégie au code)
      - la lourdeur de ces méthodologies : méthodologies verbeuses, manquant de pragmatisme
  - Urbanisation des fonctions du SI
    - Les méthodologies de modélisation des processus métier sont le plus souvent basées sur des outils BPM (Business Process Modeling) ou BPR(Business Process Re-engineering) du marché (Mega, Casewise…), voire sur des outils de type Visio ou Word…
    - Il n'existe pas de standard en matière de méthodologie, chaque société de conseil ou chaque éditeur spécialisé dans le domaine propose généralement sa propre méthodologie de modélisation, qu'elle soit basée sur un formalisme UML ou un formalisme propriétaire…

Méthodologies d'Architecture (2/2)

- Dans le domaine de l'architecture logicielle un consensus s'est créé ces dernières années autour du paradigme objet et des méthodologies basées sur UML (Unified Process, RUP) ou eXtreme Programming (XP) principalement pour les raisons suivantes :
  - Utilisation d'un langage de modélisation formel et standardisé : UML (Unified Modeling Language)
  - Puissance et adéquation du paradigme objet (abstraction, encapsulation) pour les activités d'analyse et de conception qui permet la modélisation à des niveaux successifs d'abstraction
  - Démarche itérative, et non séquentielle, entre les phases de recueil des besoins, analyse, conception, grâce notamment aux niveaux d'abstraction proposés par les modèles
  - Unification du langage de modélisation UML et des langages de développement (Java, C#, etc.) autour d'un même paradigme (l'objet), ce qui favorise la continuité entre les phases de conception et les phases d'implémentation
  - Large utilisation de patterns dans les phases d'analyse et de conception (Analysis Patterns, Design Patterns)

Chapitre 5

Vision Informatique Logique
Architectures Applicative et Logicielle

5.2 Principes directeurs

L’urbanisation du système d ’information sert de guide à l’architecture applicative

- L’urbanisme du système d’information décrit l’agencement des fonctions et des informations et leurs mises en commun, indépendamment de la façon dont elles sont implémentées par le système informatique

- Elle met en évidence :
  - des choix de modularité

    But : assurer l’évolutivité du système d’information face aux évolutions du métier
  - des choix de mise en commun

    But : assurer le réemploi et la cohérence d’ensemble

- L’architecture fonctionnelle lui sert de guide, mais elle obéit à d’autres contraintes :
  - pour tenir compte des architectures techniques et des performances
  - pour tenir compte de l’existence de progiciels du marché

- L’architecture applicative décrit l’agencement des traitements et des données

Lien avec les étapes précédentes

- **Architecture Applicative « Fonctionnelle »**
  - Blocs Applicatifs « fonctionnels »
  - Flux « fonctionnels », messages
- **Architecture Applicative « Technique »**
  - Blocs Applicatifs (fonctionnalités, logiciels)
  - Flux « techniques » (protocoles techniques supportés, synchrones/asynchrones, TP/batch, Web Services, ...)
  - Messages (XML, EDIFACT, ASCII, SWIFT, ebXML, …)
  - Cinématique représentatives de l'utilisation du système (à partir des cas d'utilisation)
- **Architecture Logicielle**
  - Modèle en 5 couches
  - Préconisation de Design Patterns
  - Préconisation de Framework (« cadre de travail ») et de services Techniques
  - (gestion des transactions, logs, traces, gestion des fichiers de configuration, ...)
- **Architecture Physique**
  - Moyens matériels, logiciels de base, réseau, infrastructure
  - Dimensionnement (matériel, OS, SGBDR, …)
  - Load-balancing, Fail-over, Scalabilité, Qualité de Service (QoS), Sécurité
  - Performance

Livrables attendus

- Les livrables des phases d'architecture (logique et technique) sont constitués d'une partie documentaire et d'une partie opérationnelle :
  - Le **dossier d'Architecture** correspondant, organisé en trois parties principales :
    - architecture applicative,
    - architecture logicielle,
    - Architecture physique
  - Une version opérationnelle du **socle technique et méthodologique**:
    - Description des environnements de développement, de tests et d'intégration, frameworks et services techniques communs (IHM, logs…), services « métier » communs (parseur de messages, machine à états, …), guides méthodologiques

- Un prototype opérationnel implémentant un ou plusieurs cas d'utilisation (use-case) significatifs, permet de:
  - valider les choix d'architecture définis dans le dossier d'Architecture (validation de la faisabilité technique)
  - éprouver le socle technique (outils, frameworks, …) sur une réalisation concrète
  - valider la méthodologie sur l'ensemble de la chaîne de production (des spécifications techniques aux procédures de déploiement).

- Un benchmark sur plate-forme de référence permet de :
  - valider le modèle de dimensionnement (principalement des serveurs et du réseau)
  - valider les exigences de production (qualité de service, montée en charge, déploiement, …)

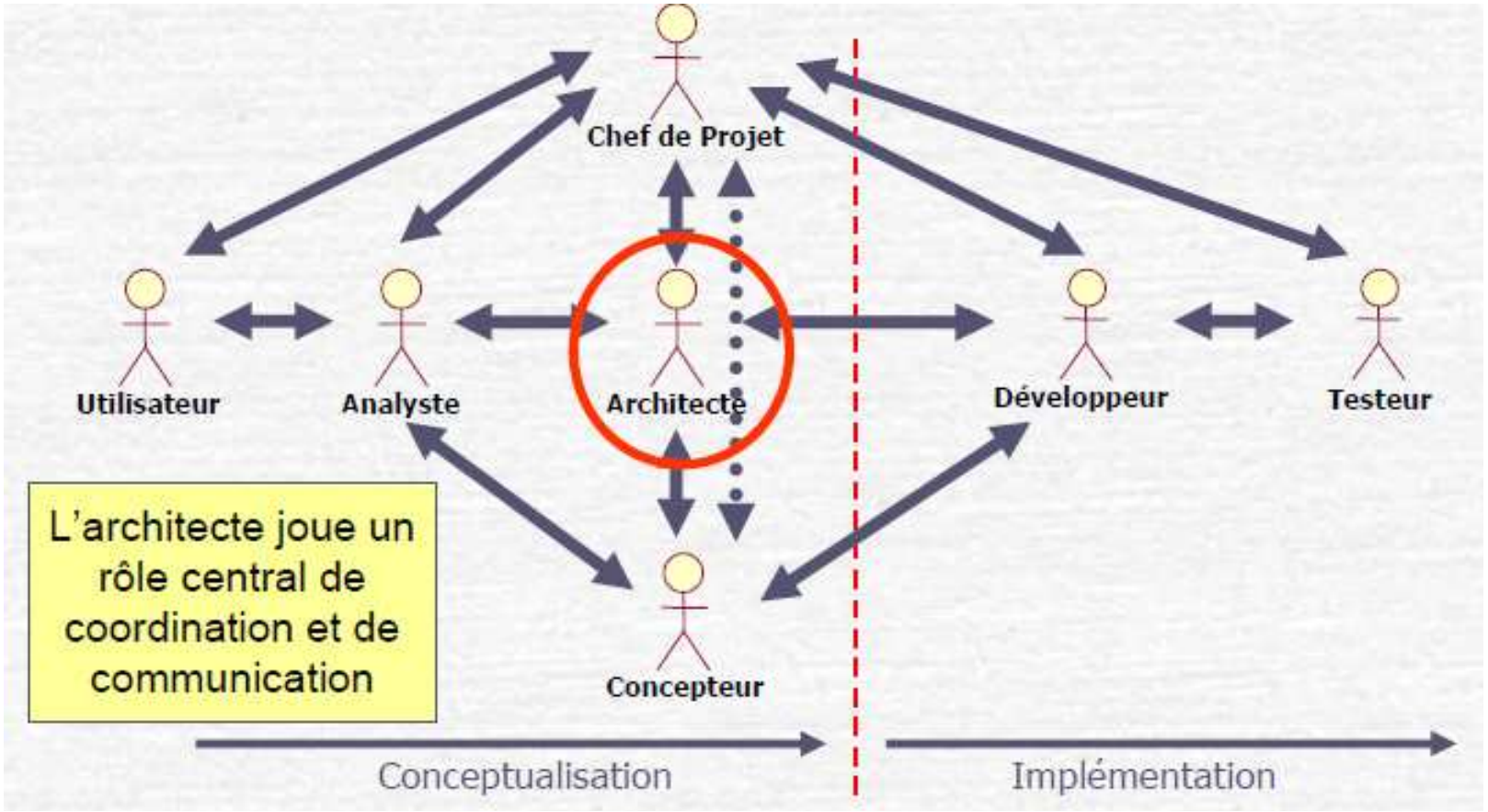
Rôle de l'architecte

- De par sa position centrale, ayant une vision globale de l'architecture et des contraintes associées (fonctionnelles, applicatives et techniques), l'architecte **fournit la première ébauche de l'architecture**. C'est lui également qui **la modifie et l'affine par itérations successives** : re-découpage / fusion de briques applicatives, application d'un pattern, mutualisation de flux…

- Il a pour rôle de:
  - Recenser les besoins techniques (analyse de l'existant, contraintes, besoins exprimés)
  - Définir les principes directeurs de l'architecture
  - Elaborer l'architecture applicative, logicielle et physique
  - Argumenter ses choix technologiques
  - Identifier les besoins en produits tiers et frameworks techniques

- Best Practices
  - Prendre connaissance du fonctionnel en récupérant les informations des analystes
  - Assister le concepteur sur les premières modélisations
    - L'architecte définit avec les équipes de conception :
      - La modélisation du découpage du modèle en couches
      - La définition des stéréotypes et des contrats entre les couches…
  - Assister le développeur sur le codage des premiers modules applicatifs
    - L'architecte définit avec les équipes de développement :
      - Le découpage du modèle en couches
      - Les framework à mettre en œuvre
      - Les design patterns à utiliser
      - Les règles de nommages des packages…
  - Participer / piloter les phases d'intégration
  - Participer / piloter les benchmarks

Le rôle de l’architecte au sein du projet



Le recensement des contraintes

- Les contraintes servent de critères de définition de l'architecture et précisent la mission de l'architecte

- Un architecte est rarement amené à créer des architectures « from scratch »

- Les contraintes sont de deux natures :
  1. **Les contraintes exprimées** : elles sont extraites des besoins fonctionnels des utilisateurs, de la charte d'architecture et d'ergonomie de l'entreprise, mais aussi de la stratégie économique du projet (business plan)
  2. **Les contraintes existantes** (induites par le contexte) : nécessité de s'interfacer et de prendre en compte l'existant du SI de l'entreprise en termes notamment :
     - De choix de technologies (par exemple : J2EE, .NET, Open Source, …)
     - De choix d'éditeurs de logiciels et de progiciels
     - De choix de constructeurs de matériels…

Contraintes exprimées

- L'architecte doit répondre à des **besoins techniques** de manière à tirer profit des technologies :
  - Besoins d'ergonomie de l'interface utilisateur (charte d'ergonomie)
  - Charte d'architecture
  - Refonte du poste de travail utilisateur en client léger, proposition d'un mode de saisie de masse, mécanisme d'authentification unique sur le système (SSO), …
  - Exigences de performances : temps de réponse moyens, disponibilité de l'application, volume des échanges, …
  - Exigences sur le procédé de fabrication, afin de garantir une maintenance aisée et plus généralement une intégration dans la gestion du « patrimoine applicatif » de l'entreprise…

- Ainsi qu'a des **besoins métier**. Le business plan par exemple produit aussi ses contraintes :
  - Budget
  - Besoins de montée en charge dans le temps (lotissement du projet)
  - Time to market…

Contraintes existantes

- L'architecte prend en compte une seconde catégorie de contraintes : les contraintes existantes. Elles ne sont pas liées directement à la demande fonctionnelle, mais au contexte.

- Les **contraintes organisationnelles** :
  - La culture de développement des équipes du projet : selon qu'elles sont familières des notions de conception et de développement orientées objet ou bien teintées développement procédural
  - Les habitudes des équipes de production : fil de l'eau versus batchs, culture de supervision
  - Les habitudes des Maîtrises d'Ouvrage pour la production de spécifications…

- Les **contraintes liées à l'environnement technique** :
  - La plateforme physique, l'OS, le réseau, le type de SGBDR, les logiciels et les technologies imposés
  - Une norme d'entreprise pour l'ergonomie des interfaces graphiques
  - La charte d'architecture
  - Un existant applicatif à intégrer avec un mode d'interopérabilité propriétaire
  - Les référentiels et annuaires auxquels il faut s'intégrer…

Exemples de contraintes structurantes

Contrainte	Exemple
Architecture centralisée v/s répartie (décentralisée)	• Un mode de fonctionnement en caisses régionales peut obliger d'avoir une architecture répartie qui pourra cependant être centralisée sur les aspects de la Business Intelligence (datawarehouse, datamart, tableaux de bord, statistiques, ciblage)
Qualité de service	• Le système informatique doit pouvoir continuer à fonctionner en mode déconnecté (avec un niveau de service dégradé) dans les agences et ceci même si la connexion réseau WAN avec le site central est hors service
Choix de technologies	• Plateformes J2EE, .NET, mise en œuvre de solutions Open Source
Contrainte de sécurité	• Les infos doivent être dupliquées (sas étanche) pour des questions de sécurité entre les applications de gestion et les applications accessible par les clients via Internet, Minitel, WAP, SVI…

Règles liées aux contraintes
Définition des principes directeurs de l'architecture

1. La prise en compte des contraintes est primordiale
   - Lors de la construction de l'architecture technique, **plusieurs scénarios d'architecture doivent être envisagés**. Une préconisation de scénario devra être arrêtée et validée.

2. Nécessité parfois, d'écarter des solutions d'architecture qui semblaient s'imposer sur le papier au regard des besoins exprimés
   - Pour un même besoin, un architecte ne pourra pas toujours décliner à la lettre la même architecture.

3. Ne pas hésiter à « challenger » les contraintes dans leur ensemble
   - Le coût financier, l'impact métier d'une panne ont-ils été mesurés et justifient-ils vraiment la contrainte de haute disponibilité ?
   - Certaines contraintes sont parfois incompatibles :
     - Exemple : besoin de performances maximales conjugué à l'utilisation d'un langage ou d'un framework de haut niveau.

4. Les arbitrages sur la priorité des contraintes, l'annulation de certaines d'entre elles sont inévitables
   - La contrainte du budget peut par exemple imposer de se limiter aux documentations les plus pertinentes au meilleur coût, la faible disponibilité des utilisateurs ne sera pas compatible avec une méthodologie de spécifications très itérative...

5. La solution d'architecture doit prendre en compte les contraintes organisationnelles, le patrimoine de l'entreprise et le niveau de compétence des équipes internes
   - Le gap doit être franchissable et les risques encourus sont à mesurer.

Migration vers l'architecture cible

- Lorsque l'on fait évoluer ou que l'on refond un système ou une application existante il est primordial de gérer les phases de migration vers le système / application cible :
  - Migration / refonte des systèmes et des applications
  - Migration des données (problématique des initialisations et des mises à jour)

- Il faut toujours garder à l'esprit l'architecture cible et définir les différentes étapes pour y parvenir

- Plusieurs méthodes sont envisageables :
  - **L'approche itérative / progressive** qui consiste à définir des étapes intermédiaires pendant lesquels l'ancien et le nouveau système cohabiteront pendant un temps donné (de quelques mois à plusieurs années !). Cette méthode permet de mieux maîtriser la complexité par une mise en œuvre progressive du nouveau système. Son inconvénient majeur réside souvent dans la complexité à intégrer et à gérer deux systèmes de façon concomitante (notamment pose les problématiques de double alimentation TP/batch, et la complexité des flux à gérer)
  - **L'approche « big-bang »**, qui consiste à basculer de l'ancien système vers le nouveau système en une étape. Ce scénario comporte beaucoup plus de risque que le précédent (problématique de tenue à la charge, risque de rejet des utilisateurs, complexité de mise en oeuvre et d'administration, formation des équipes, …). Son avantage majeur est qu'il règle les problématiques de double alimentation et de complexité de gestion des flux)

Métriques à prendre en compte pour valider une architecture

- Il existe de nombreuses façons de valider une architecture technique (prototype, benchmark de performance, etc.) mais il est plus difficile de trouver des méthodes pour valider une architecture sous toutes ses dimensions

- Il existe de nombreuses façons de caractériser un SI. Voici une liste non exhaustive de métriques à prendre en compte lors de la validation de l'architecture :
  - **Agilité / Extensibilité** :
    - Capacité à intégrer rapidement de nouveaux flux et/ou de nouveaux applicatifs : intégration d'un nouveau canal de distribution, sous-traitance d'une activité, etc.
    - Capacité à supporter plusieurs modes d'organisations
  - **Respect des standards** applicatifs et techniques de l'entreprise
    - L'architecture utilise-t-elle les fonctions transverses mises en place au niveau SI, comme les référentiels, la sécurité, les plates-formes d'échanges, les services communs ?
    - L'architecture technique s'insère-t-elle dans l'univers d'exploitation connu ou nécessite-t-elle de nouvelles technologies ?
  - **Evolutivité** : Capacité à intégrer de nouveaux utilisateurs, des référentiels plus larges, montée en charge, etc.
  - **Sécurité** : Mesure de la sécurité globale du système : sécurisation des échanges, difficulté d'intrusion, authentification des acteurs, etc.
  - **Coûts** : Mesure du rapport qualité/prix, benchmark par rapport à d'autres constructions

Chapitre 5

Vision Informatique Logique Architectures Applicative et Logicielle

5.3 Architecture Applicative

Définitions

- Architecture Applicative
  - Elle structure le SI en blocs applicatifs communicants
  - Elle décrit sous l'angle technique les applications, les flux et les messages échangés entre applications
- Bloc applicatif
  - Module logiciel exécutable ayant une identité, proposant des services et ayant une interface (prise) bien définie
- Approche « boîte noire »
  - Connaissance des entrées et sorties du bloc applicatif
  - Connaissance des fonctionnalités et des technologies
  - Dans cette phase les détails du découpage interne du bloc applicatif en couches n'est pas étudié

Principes directeurs de l'Architecture Applicative

- Pour définir les flux échangés, nous allons nous baser sur les principes directeurs de l'architecture

- Exemples :
  - Toujours privilégier l'utilisation des standards techniques du marché HTTP, XML, XSL, HTML, Javascript, DOM/DHTML, Web Services (SOAP, WSDL, UDDI), J2EE, .NET, FTP, …
  - Utilisation des langages XML (XML, XML Schema, SOAP, WSDL, ebXML, …) comme format pivot pour les messages échangés
  - Les échanges synchrones entre blocs applicatifs sont réalisés en utilisant SOAP/HTTP
  - Les échanges asynchrones entre blocs applicatifs sont réalisés en utilisant un MOM…

Une démarche en 4 étapes

1. Décrire de façon détaillée (fonctionnelle, applicative et technique) chacun des blocs applicatifs (interne, externe, filiale ou partenaire). De plus il faut :
   - faire apparaître les blocs applicatifs concernant l'infrastructure (annuaire, SSO, supervision, …) et
   - pour chaque bloc applicatif, il faut préciser les aspects liés aux contraintes de sécurité, aux performances, à la continuité de service et à la géographie

2. Construire une cartographie applicative détaillée présentant tous les flux (synchrones/asynchrones, TP/batch) et messages échangés entre les blocs applicatifs (interne, externe, filiale ou partenaire)
   - **La cartographie applicative des flux donne une vision statique du système**

3. Construire la matrice des flux à partir de la cartographie applicative des flux
   - Lister et numéroter l'ensemble des flux identifier sur la cartographie applicative
   - Définir le sens du flux (depuis le bloc applicatif A vers le bloc applicatif B)
   - Définir le support physique : LAN, WAN, VPN, bande magnétique, CD, DVD, …
   - Définir le type de traitement : TP, batch
   - Définir le format de message : XML, fichiers ASCII, EDIFACT, SWIFT, ebXML, …
   - Définir le protocole de transport : HTTP, FTP, SMTP, …

4. A partir des cas d'utilisation identifier un nombre limité de cinématique représentative de l'utilisation du système
   - **Les cinématiques d'utilisation du système donnent une vision dynamique du système**

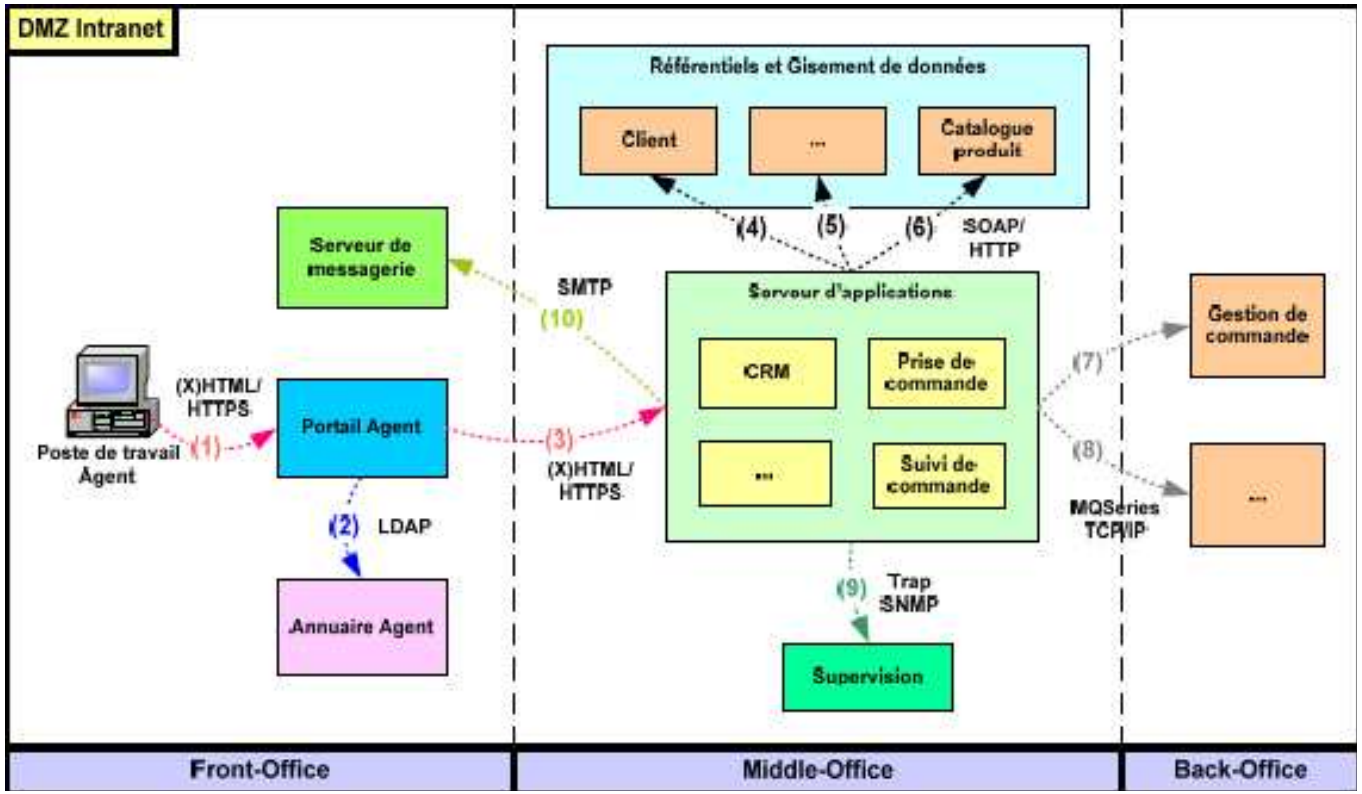
Etape 1

- Description fonctionnelle
  - Objectifs
  - Classification (Critique, Important, Utile)
  - Entrées / Sorties
  - Fonctions
  - Classes concepts gérées (matrice entités gisement de données et référentiel) / Type gestion (création, modification, suppression, lecture)…

- Description applicative et technique
  - Développement spécifique ou progiciel
  - Volume traité (volumétries statiques et dynamiques)
  - Disponibilité (contraintes de disponibilité, plage horaire TP, fenêtre d'exécution des batchs)
  - Fiabilité, niveau de service, performances attendues
  - Matériel
  - Système d'exploitation
  - SGBDR ou SGF (Système de Gestion de Fichiers)
  - Middleware
  - Types de sites concernés (notion de localisation et de géographie)
  - Sécurité (identification, authentification, gestion des habilitations, intégrité, confidentialité et non répudiation (données et échanges)…

Etape 2

- Exemple de cartographie applicative des flux présentant une **vision statique** du système



Etape 3

- Un flux est un échange de données (messages) entre blocs applicatifs. Il peut être continu ou déclenché à certains moments de la journée (prise en compte de mouvements comptables la nuit, …)

- Un flux peut être interne au système étudié ou provenir de ou être destiné à un système externe
  - Description textuelle
  - Liste des données échangées
  - Description du mode de transmission
  - Description des conditions de déclenchement
  - Signalisation flux externe ou interne
  - Indications de volume et de fréquence
  - Source, Cible (sens du flux)
  - Traçabilité
  - Sécurité
  - Support (transfert de fichiers, protocole, CD, …)
  - Nature (Synchrone, Asynchrone, TP, Batch, …)
  - Format des données (XML, ASCII propriétaire, …)
  - Propriétaire du Format
  - Phase Projet (Initialisation, Migration, Cible)

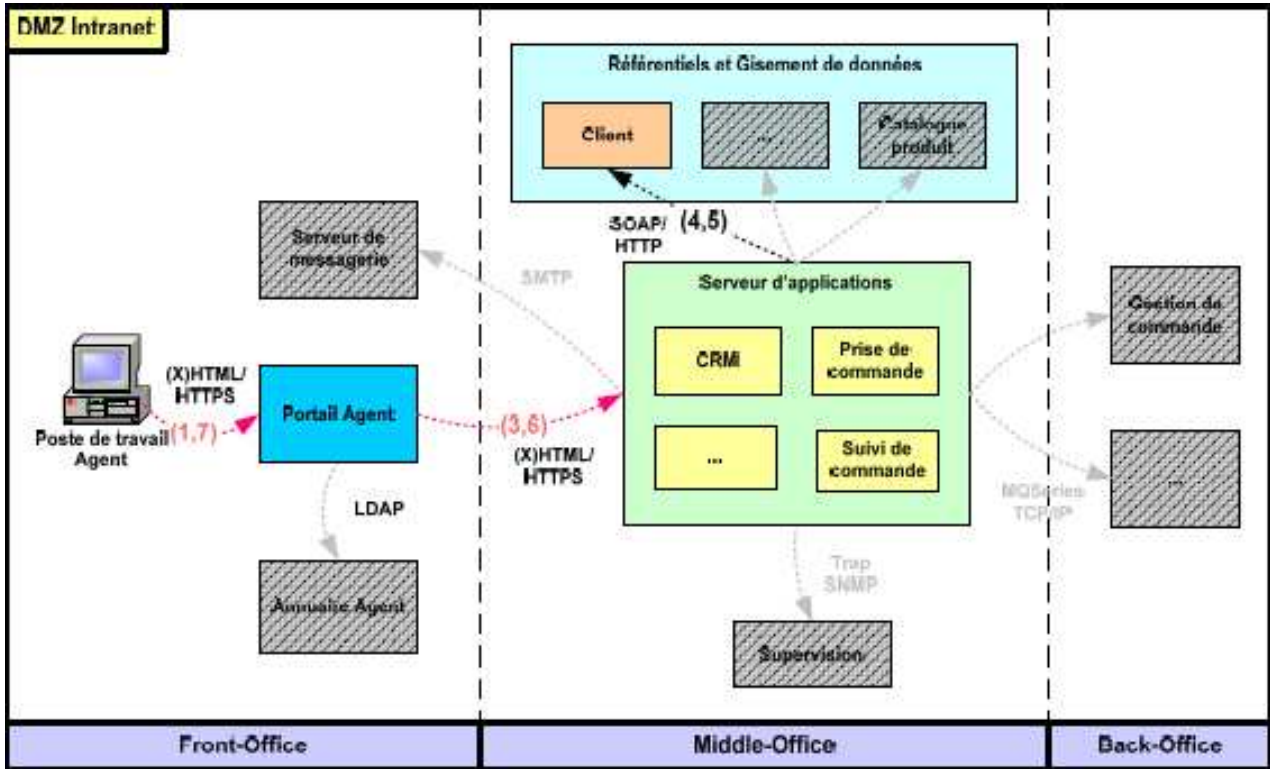
Etape 3 – Matrice des flux

- La matrice des flux est créée à partir de la cartographie applicative des flux
- Exemple :

#	Depuis	Vers	Support	TP / Batch	Format message	Protocole de Transport	Description
1	Navigateur du client	Portail client	Internet	TP	(X)HTML	HTTP	Connexion du client au portail
2	Portail client	Annuaire client	LAN	TP	LDAP	TCP/IP	Authentification du client
...	...	...	...	...	...	...	...

Etape 4 – Cinématique représentative de l'utilisation du système

- Exemple de cinématique présentant une **vision dynamique** du système



Etape 4 – Cinématique représentative de l'utilisation du système (suite)

- Exemple:

Etape	Description
(1)	L'agent via son navigateur internet saisie l'identifiant du client, puis soumet la requête au portail
(2)	Le portail agent transmet (mode proxy) la requête de l'agent au serveur d'applications
(3)	Le serveur d'applications (module applicatif CRM) fait appel à un service de recherche de la synthèse client (référentiel client) en SOAP/HTTP (Services Web XML)
(4)	Le référentiel client traite la demande et renvoi au serveur d'applications la demande (module applicatif CRM)
(5)	La réponse est retournée au portail agent
(6)	Le portail agent retourne la réponse au navigateur pour affichage