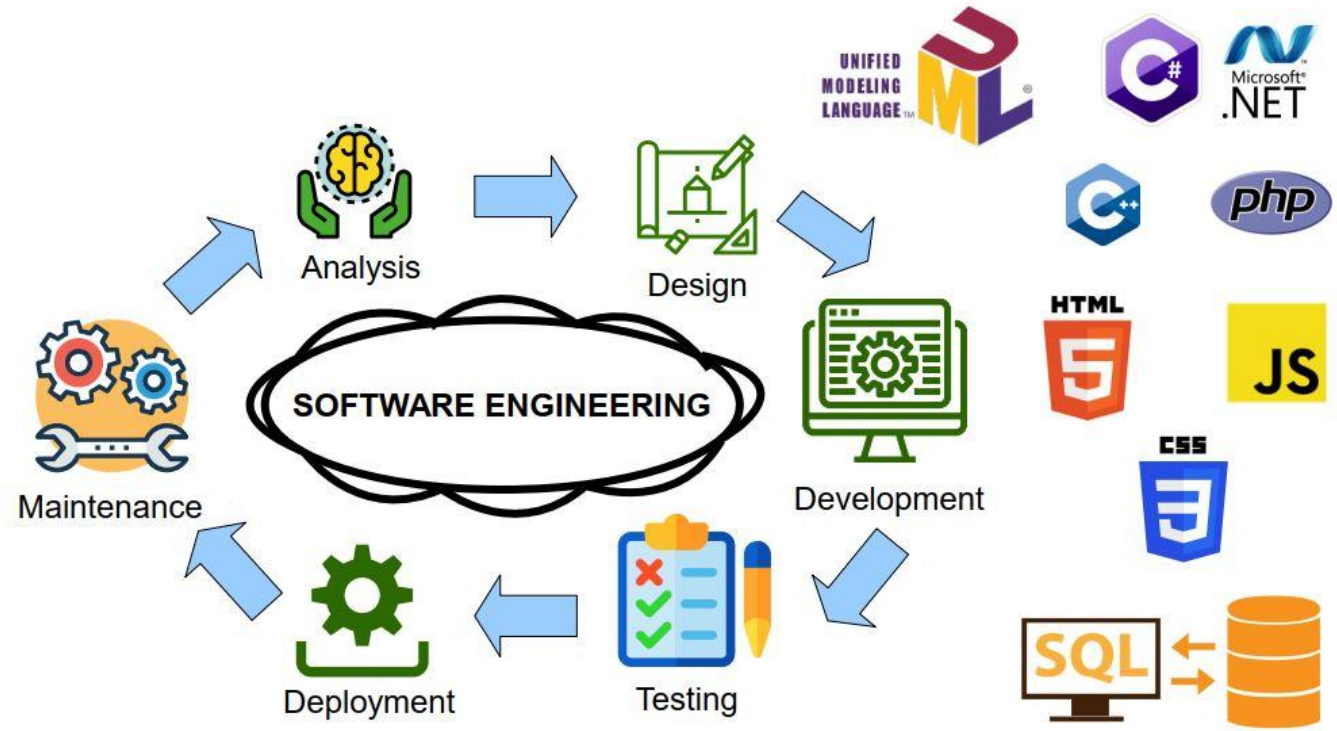
Architecture Logicielle



Dr El Hadji Bassirou TOURE

Université Cheikh Anta Diop de Dakar

2024 - 2025

Architecture Logicielle

- Pour créer un produit fiable, sûr et efficace, vous devez prêter attention à la conception architecturale, qui comprend :
  - son organisation générale,
  - la manière dont le logiciel est décomposé en composants,
  - l'organisation du serveur
  - L'architecture d'un produit logiciel influe sur ses performances, sa facilité d'utilisation, sa sécurité, sa fiabilité et sa maintenabilité.
- Il existe de nombreuses interprétations différentes du terme « architecture logicielle ».
  - Certains considèrent l'architecture comme statique - la structure d'un système - et d'autres comme dynamique - le processus de définition de ces structures.

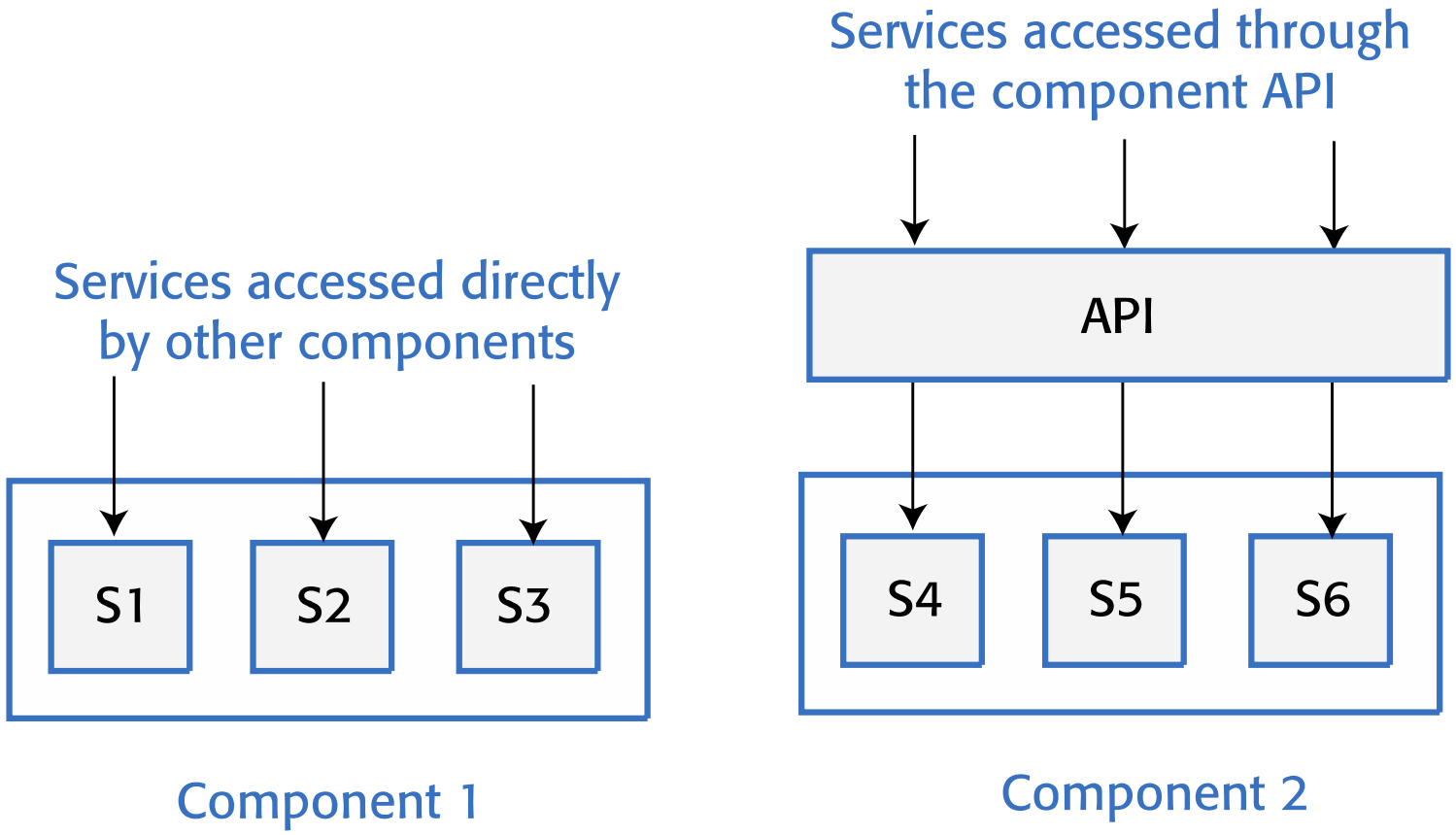
Définition “Architecture Logicielle” de l’IEEE

- *L'architecture est l'organisation fondamentale d'un système logiciel incarnée par ses composants, leurs relations entre eux et avec l'environnement, et les principes qui guident sa conception et son évolution.*

Composant

- Elément qui met en œuvre un ensemble cohérent de fonctionnalités ou de caractéristiques.
- Peut être considéré comme une collection d'un ou plusieurs services qui peuvent être utilisés par d'autres composants.
  - Pas nécessaire de décider comment un composant est mis en œuvre.
  - Plutôt concevoir l'interface du composant
  - Et laisser la mise en œuvre de cette interface à un stade ultérieur.

Figure 4.1. Accès aux services fournis par les composants logiciels.



Pourquoi l'architecture est-elle importante ?

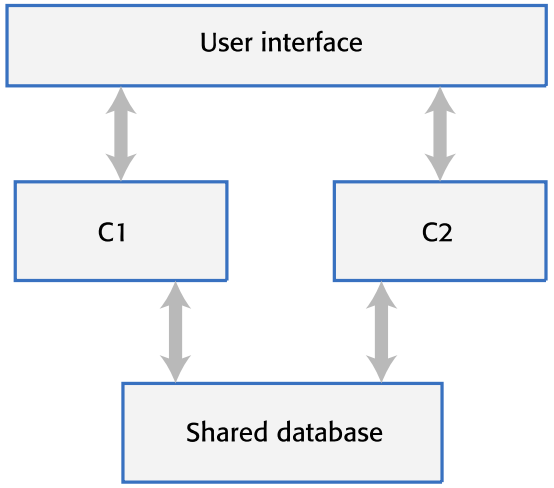
- L'architecture est importante car elle a une influence fondamentale sur les propriétés non fonctionnelles du système.
- La conception architecturale implique :
  - de comprendre les questions qui affectent l'architecture de votre produit
  - et de créer une description architecturale qui montre les composants critiques et leurs relations.
- La réduction de la complexité doit être un objectif important pour les architectes.
  - Plus un système est complexe, plus il est difficile et coûteux de le comprendre et de le modifier.
  - Les programmeurs sont plus susceptibles de commettre des erreurs et d'introduire des bogues et des failles de sécurité lorsqu'ils modifient ou étendent un système complexe….

Attributs non fonctionnels de la qualité des systèmes

- **Réactivité :** Le système renvoie-t-il les résultats aux utilisateurs dans un délai raisonnable ?
- **Fiabilité** : Les fonctionnalités du système se comportent-elles comme prévu par les développeurs et les utilisateurs ?
- **Disponibilité** : Le système peut-il fournir ses services à la demande des utilisateurs ?
- **Sécurité** : Le système se protège-t-il et protège-t-il les données des utilisateurs contre les attaques et les intrusions non autorisées ?
- **Facilité d'utilisation** : Les utilisateurs du système peuvent-ils accéder aux fonctionnalités dont ils ont besoin et les utiliser rapidement et sans erreur ?
- **Maintenabilité** : Le système peut-il être facilement mis à jour et de nouvelles fonctionnalités peuvent-elles être ajoutées sans coûts excessifs ?
- **Résilience** : Le système peut-il continuer à fournir des services aux utilisateurs en cas de défaillance partielle ou d'attaque extérieure ?

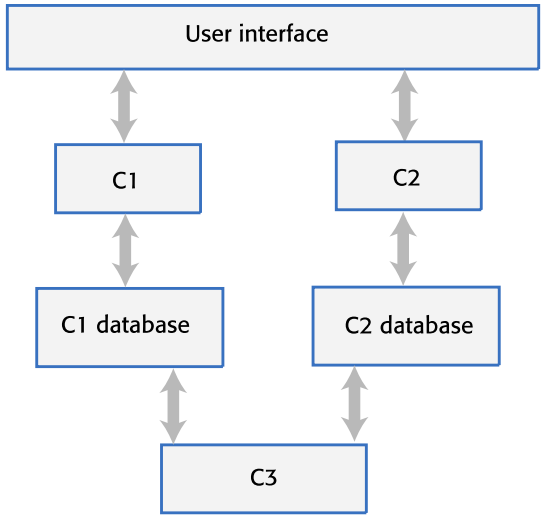
Maintenabilité et performance

- La figure ci-dessous illustre un système composé de deux éléments (C1 et C2) qui partagent une base de données commune.
  - Supposons que C1 fonctionne lentement parce qu'il doit réorganiser les informations contenues dans la base de données avant de les utiliser.
  - Le seul moyen de rendre C1 plus rapide pourrait être de modifier la base de données. Cela signifie que C2 doit également être modifié, ce qui peut éventuellement affecter son temps de réponse.



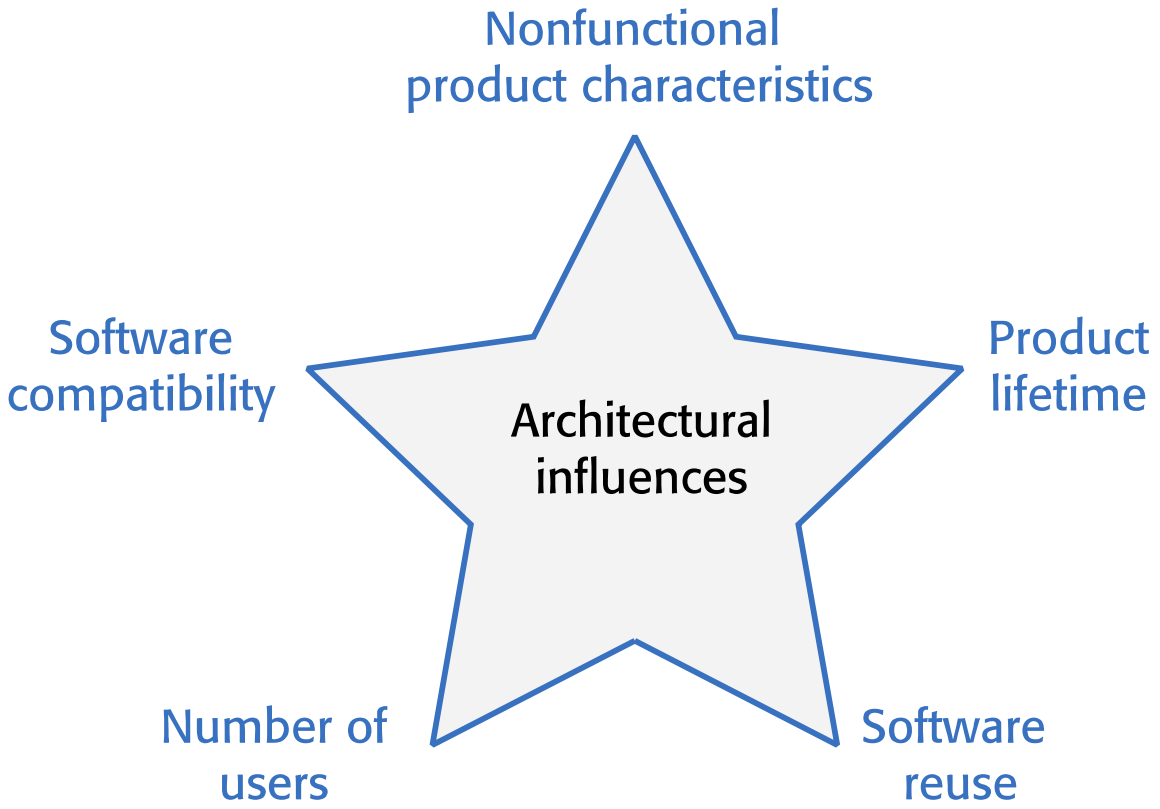
Maintenabilité et performance

- La figure ci-dessous présente une architecture différente dans laquelle chaque composant possède sa propre copie des parties de la base de données dont il a besoin.
  - Si un composant doit modifier l'organisation de la base de données, cela n'affecte pas l'autre composant.
- Cependant, une architecture multi-bases de données peut fonctionner plus lentement et coûter plus cher à mettre en œuvre et à modifier.
  - Une architecture multi-bases de données nécessite un mécanisme (composant C3) pour garantir que les données partagées par C1 et C2 restent cohérentes lorsqu'elles sont modifiées.



Database reconciliation

Questions relatives à l'architecture



Questions relatives à l'architecture

- Caractéristiques non fonctionnelles du produit
  - Les caractéristiques non fonctionnelles du produit, telles que la sécurité et les performances, affectent tous les utilisateurs.
  - Malheureusement, certaines caractéristiques sont opposées, de sorte que vous ne pouvez optimiser que les plus importantes.
- Durée de vie du produit
  - Si vous prévoyez une longue durée de vie du produit, vous devrez procéder à des révisions régulières du produit.
  - Vous avez donc besoin d'une architecture évolutive, afin de pouvoir l'adapter aux nouvelles fonctionnalités et technologies.

Questions relatives à l'architecture

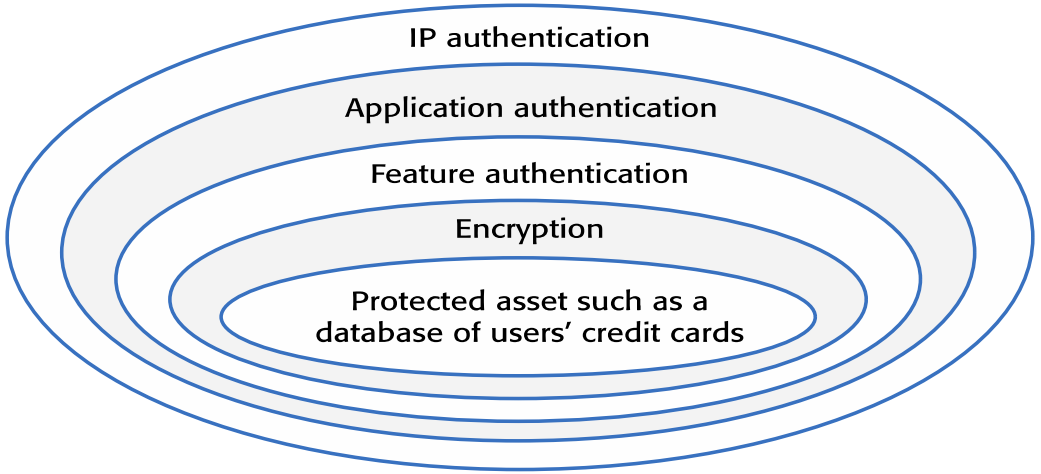
- Réutilisation de logiciels
  - Vous pouvez économiser beaucoup de temps et d'efforts en réutilisant de grands composants provenant d'autres produits ou de logiciels libres.
  - Cela peut limiter vos choix architecturaux car vous devez adapter votre conception au logiciel réutilisé.
- Nombre d'utilisateurs
  - Si vous développez un logiciel grand public, le nombre d'utilisateurs peut changer très rapidement.
  - Cela peut entraîner une grave dégradation des performances, à moins que vous ne conceviez votre architecture de manière à ce que votre système puisse être rapidement augmenté ou réduit.
- Compatibilité logicielle
  - Pour certains produits, il est important de maintenir la compatibilité avec d'autres logiciels afin que les utilisateurs puissent adopter votre produit et utiliser des données provenant d'autres systèmes.
  - Cela peut limiter les choix architecturaux, tels que le SGBD que vous pouvez utiliser.

Compromis : Maintenabilité VS Performance

- La maintenabilité est un attribut qui mesure la difficulté et le coût des modifications à apporter à un système une fois qu'il a été mis à la disposition des clients.
  - Vous améliorez la maintenabilité en construisant un système à partir de petites parties autonomes, dont chacune peut être remplacée ou améliorée si des changements sont nécessaires.
- En termes d'architecture, cela signifie que le système doit être décomposé en composants à granularité fine, dont chacun fait une chose et une seule.
  - Cependant, il faut du temps pour que les composants communiquent entre eux.
  - Par conséquent, si de nombreux composants sont impliqués dans la mise en œuvre d'une fonction du produit, le logiciel risque d'être plus lent (temps de réponse).

Compromis : Sécurité VS Facilité d'utilisation

- Vous pouvez assurer la sécurité en concevant la protection du système comme une série de couches
  - Un attaquant doit pénétrer toutes ces couches avant que le système ne soit compromis.
- Les couches peuvent comprendre des couches d'authentification du système, une couche d'authentification des fonctions critiques, une couche de cryptage, etc.
- D'un point de vue architectural, vous pouvez mettre en œuvre chacune de ces couches en tant que composants distincts, de sorte que si l'un de ces composants est compromis par une attaque, les autres couches restent intactes.



Problèmes d'utilisabilité

- Une approche stratifiée de la sécurité affecte la facilité d'utilisation du logiciel.
  - Les utilisateurs doivent se souvenir des informations, comme les mots de passe, qui sont nécessaires pour pénétrer une couche de sécurité.
  - Leur interaction avec le système est inévitablement ralentie par les dispositifs de sécurité.
  - De nombreux utilisateurs trouvent cela irritant et cherchent souvent des solutions de contournement afin de ne pas avoir à s'authentifier à nouveau pour accéder aux fonctions ou aux données du système.
- Pour éviter cela, vous avez besoin d'une architecture
  - qui ne comporte pas trop de couches de sécurité,
  - qui n'impose pas de sécurité inutile,
  - qui fournit des composants d'aide qui réduisent la charge des utilisateurs.

Compromis : Disponibilité VS délai de mise sur le marché

- La disponibilité est particulièrement importante pour les produits d'entreprise, tels que les produits destinés au secteur financier, qui doivent fonctionner 24/24 et 7/7.
- La disponibilité d'un système est une mesure du temps de fonctionnement de ce système.
  - La disponibilité est normalement exprimée en pourcentage du temps pendant lequel un système est disponible pour fournir des services aux utilisateurs.
- D'un point de vue architectural, la disponibilité est assurée par la présence de composants redondants dans un système.
  - Pour utiliser la redondance, il faut inclure des capteurs qui détectent les pannes et des composants de commutation qui basculent vers un composant redondant lorsqu'une panne est détectée.
- La mise en œuvre de composants supplémentaires prend du temps et augmente le coût de développement du système.
  - Elle ajoute de la complexité au système et augmente donc les risques d'introduction de bogues et de vulnérabilités.

Questions relatives à la conception architecturale

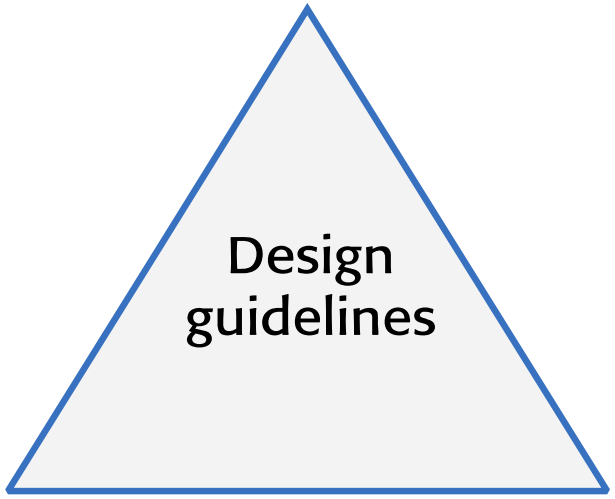
- Comment le système doit-il être organisé comme un ensemble de composants architecturaux, où chacun de ces composants fournit un sous-ensemble de la fonctionnalité globale du système ?
  - L'organisation doit fournir la sécurité, la fiabilité et la performance du système dont vous avez besoin.
- Comment ces composants architecturaux doivent-ils être répartis et communiquer entre eux ?
- Quelles sont les technologies à utiliser pour construire le système et quels sont les composants à réutiliser ?

Complexité architecturale

- La complexité de l'architecture d'un système résulte du nombre et de la nature des relations entre les composants de ce système.
- Lors de la décomposition d'un système en composants, il convient d'essayer d'éviter toute complexité logicielle inutile.
  - **Localiser les relations**
    S'il existe des relations entre les composants A et B, il est plus facile de les identifier si A et B sont définis dans le même module.
  - **Réduire les dépendances partagées**
    Lorsque les composants A et B dépendent d'un autre composant ou de données, la complexité augmente car les changements apportés au composant partagé signifient que vous devez comprendre comment ces changements affectent à la fois A et B.
- Il est toujours préférable d'utiliser des données locales dans la mesure du possible et d'éviter de partager des données.

Lignes directrices pour la conception architecturale

Separation of concerns
Organize your architecture into components that focus on a single concern



Stable interfaces
Design component interfaces that are coherent and that change slowly

Implement once
Avoid duplicating functionality at different places in your architecture

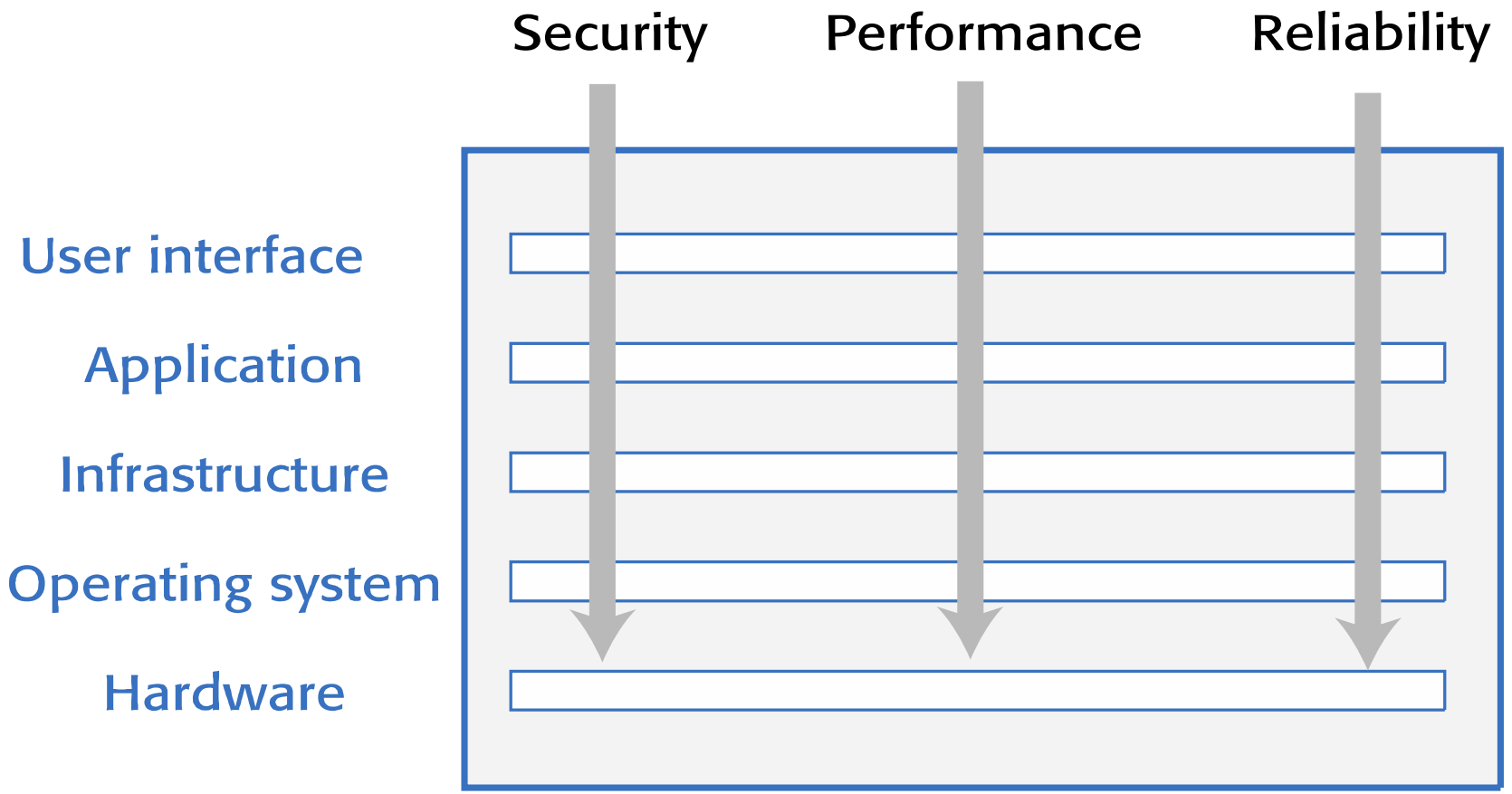
Directives de conception et architectures en couches

- Chaque couche est un domaine de préoccupation et est considérée séparément des autres couches.
  - La couche supérieure s'occupe de l'interaction avec l'utilisateur, la couche suivante de la gestion de l'interface utilisateur, la troisième de la recherche d'informations, etc.
- Au sein de chaque couche, les composants sont indépendants et leurs fonctionnalités ne se chevauchent pas.
  - Les couches inférieures comprennent des composants qui fournissent des fonctionnalités générales, de sorte qu'il n'est pas nécessaire de les reproduire dans les composants d'un niveau supérieur.
- Le modèle architectural est un modèle de haut niveau qui ne contient pas d'informations sur la mise en œuvre.
  - Idéalement, les composants du niveau X ne devraient interagir qu'avec les API des composants du niveau X-1.

Préoccupations transversales

- Les préoccupations transversales sont des préoccupations systémiques, c'est-à-dire qu'elles affectent l'ensemble du système.
- Dans une architecture en couches, les préoccupations transversales affectent toutes les couches du système ainsi que la manière dont les gens utilisent le système.
- Les préoccupations transversales sont complètement différentes des préoccupations fonctionnelles représentées par les couches.
- Chaque couche doit les prendre en compte et il y a inévitablement des interactions entre les couches en raison de ces préoccupations.
- L'existence de préoccupations transversales est la raison pour laquelle il est souvent difficile de modifier un système après sa conception pour en améliorer la sécurité.

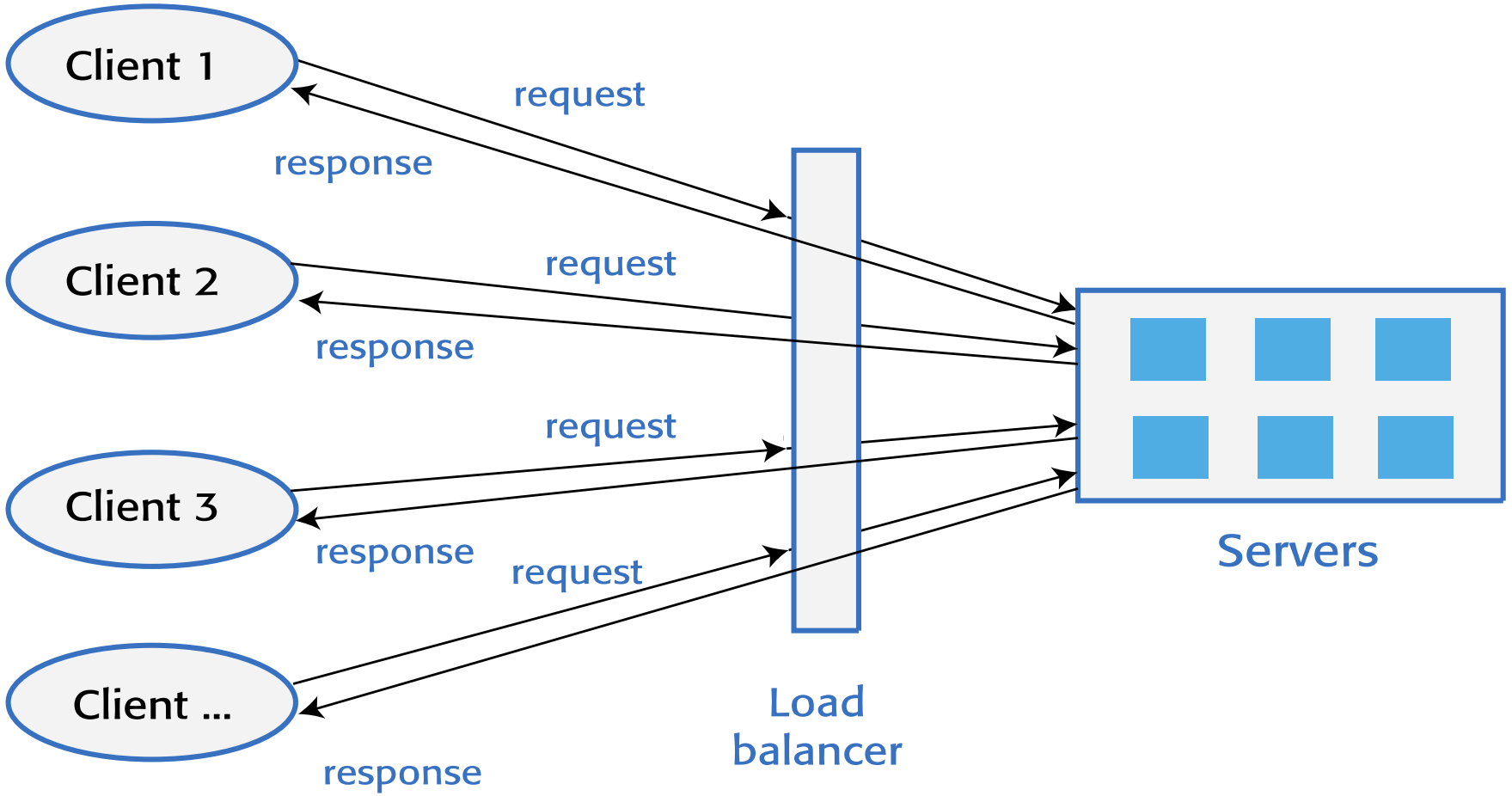
Préoccupations transversales



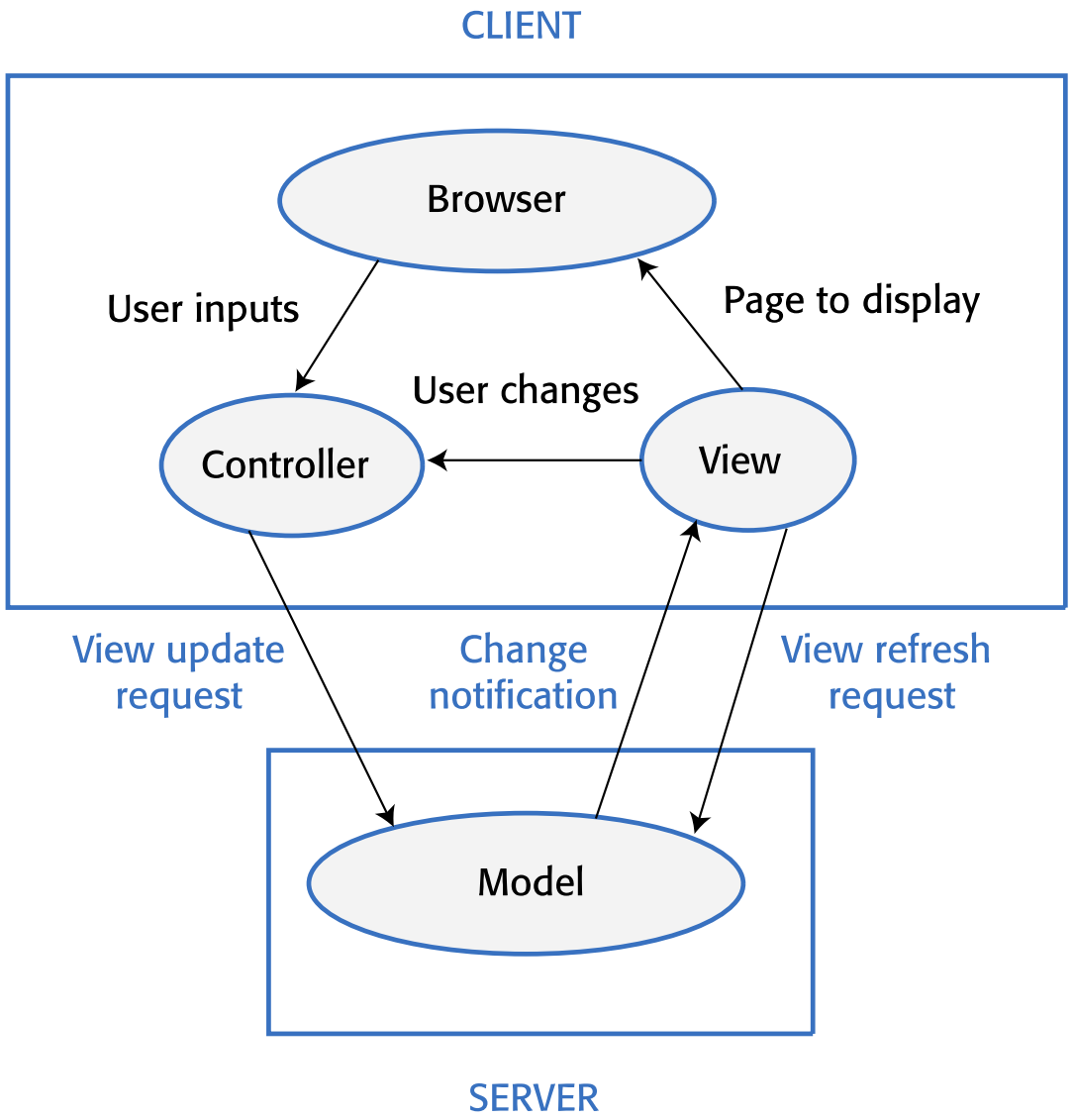
Architecture distribuée

- Une architecture distribuée d'un logiciel définit les serveurs du système et l'attribution des composants à ces serveurs.
- Les architectures client-serveur sont un type d'architecture distribuée qui convient aux applications dans lesquelles les clients accèdent à une base de données partagée et à des opérations de logique métier sur ces données.
- Dans cette architecture, l'interface utilisateur est mise en œuvre sur l'ordinateur ou l'appareil mobile de l'utilisateur.
  - Les fonctionnalités sont réparties entre le client et un ou plusieurs ordinateurs serveurs.

Architecture client-serveur



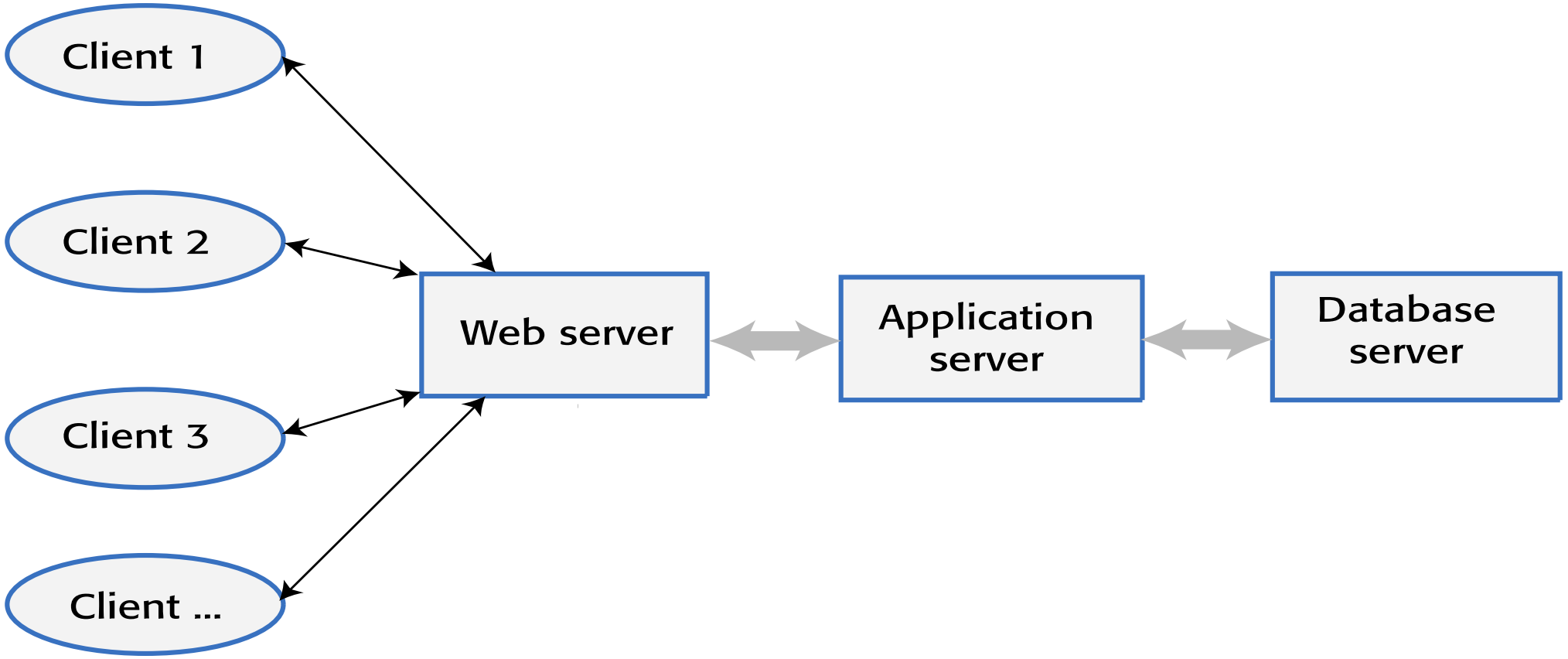
Le modèle modèle-vue-contrôleur



Communication client-serveur

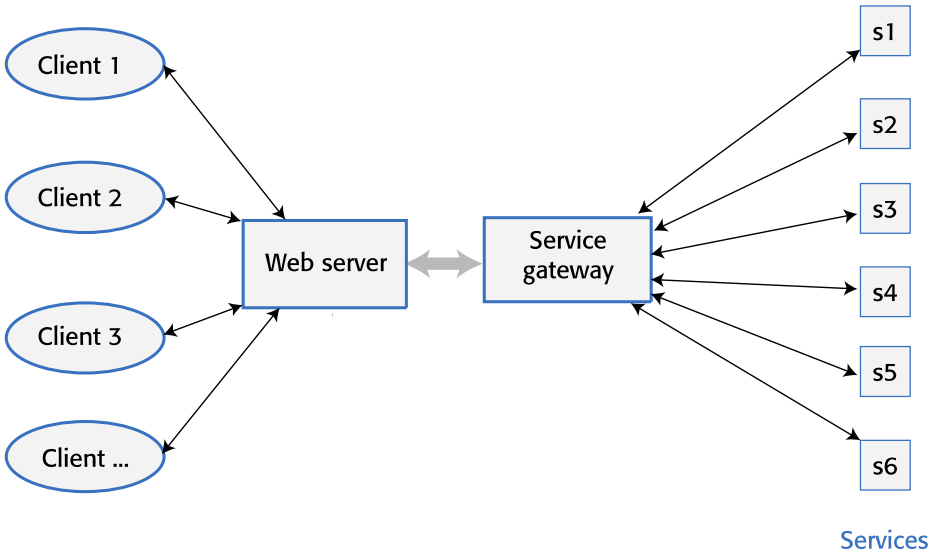
- La communication client-serveur utilise normalement le protocole HTTP.
  - Le client envoie un message au serveur qui comprend une instruction telle que GET ou POST ainsi que l'identifiant d'une ressource (généralement un URL) sur laquelle cette instruction doit fonctionner. Le message peut également contenir des informations supplémentaires, telles que des informations collectées à partir d'un formulaire.
- Le protocole HTTP est un protocole texte uniquement, de sorte que les données structurées doivent être représentées sous forme de texte. Il existe deux façons de représenter ces données qui sont largement utilisées, à savoir XML et JSON.
  - XML est un langage de balisage dont les balises servent à identifier chaque élément de données.
  - JSON est une représentation plus simple basée sur la représentation des objets dans le langage Javascript.

Architecture client-serveur à plusieurs niveaux



Architecture orientée services

- Dans une architecture orientée services, les services sont des composants sans état, ce qui signifie qu'ils peuvent être répliqués et migrer d'un ordinateur à l'autre.
- De nombreux serveurs peuvent être impliqués dans la fourniture de services
- Une architecture orientée services est généralement plus facile à adapter à l'augmentation de la demande et résiste aux pannes.



Questions relatives au choix de l'architecture

- Type de données et mise à jour des données
  - Si vous utilisez principalement des données structurées susceptibles d'être mises à jour par différentes fonctionnalités du système, il est généralement préférable de disposer d'une seule base de données partagée qui assure le verrouillage et la gestion des transactions. Si les données sont réparties entre les services, vous devez trouver un moyen de les maintenir cohérentes, ce qui alourdit la charge de travail de votre système.
- Modifier la fréquence
  - Si vous prévoyez que les composants du système seront régulièrement modifiés ou remplacés, le fait de les isoler en tant que services distincts simplifie ces changements.
- La plate-forme d'exécution du système
  - Si vous prévoyez de faire fonctionner votre système dans le cloud et que les utilisateurs y accèdent par internet, il est généralement préférable de le mettre en œuvre sous la forme d'une architecture orientée services, car la mise à l'échelle du système est plus simple.
  - Si votre produit est un système d'entreprise qui fonctionne sur des serveurs locaux, une architecture à plusieurs niveaux peut être plus appropriée.

Choix technologiques

- Base de données
  - Devriez-vous utiliser une BD SQL ou une BD NOSQL non structurée ?
- Plateforme
  - Devriez-vous livrer votre produit sur une application mobile et/ou une plateforme web ?
- Serveur
  - Devriez-vous utiliser des serveurs internes dédiés ou concevoir votre système pour qu'il fonctionne sur un cloud public ? S'il s'agit d'un cloud public, devriez-vous utiliser Amazon, Google, Microsoft ou une autre option ?
- Open source
  - Existe-t-il des composants open source appropriés que vous pourriez intégrer dans vos produits ?
- Outils de développement
  - Vos outils de développement intègrent-ils des hypothèses architecturales sur le logiciel en cours de développement qui limitent vos choix architecturaux ?

Les bases de données

- Deux types de bases de données sont aujourd'hui couramment utilisés :
  - les BD relationnelles, dans lesquelles les données sont organisées en tableaux structurés
  - les BD NoSQL, dans lesquelles les données sont organisées de manière plus souple et définie par l'utilisateur.
  - Les BD relationnelles, telles que MySQL, sont particulièrement adaptées aux situations où vous avez besoin d'une gestion des transactions et où les structures de données sont prévisibles et relativement simples.
  - Les BD NoSQL, telles que MongoDB, sont plus flexibles et potentiellement plus efficaces que les BD relationnelles pour l'analyse des données.
    - Les BD NoSQL permettent d'organiser les données de manière hiérarchique plutôt que sous forme de tableaux plats, ce qui permet un traitement simultané plus efficace des "big data".

Plate-forme de livraison

- La livraison peut se faire sous la forme d'un produit web ou mobile, ou les deux.
- Problèmes de téléphonie mobile :
  - Connectivité : Vous devez être en mesure de fournir un service limité sans connectivité réseau.
  - Puissance du processeur : Les appareils mobiles ont des processeurs moins puissants, vous devez donc minimiser les opérations à forte intensité de calcul.
  - Gestion de l'énergie : L'autonomie de la batterie des appareils mobiles étant limitée, vous devez essayer de minimiser la consommation d'énergie de votre application.
  - Clavier à l'écran : Les claviers à l'écran sont lents et sujets aux erreurs. Vous devez minimiser la saisie à l'aide du clavier de l'écran afin de réduire la frustration de l'utilisateur.
- Pour faire face à ces différences, vous avez généralement besoin de versions distinctes du front-end de votre produit pour les navigateurs et pour les téléphones portables.
  - Il se peut que vous ayez besoin d'une architecture de décomposition complètement différente dans ces différentes versions pour garantir le maintien des performances et d'autres caractéristiques.

Serveur

- Une décision clé que vous devez prendre est de savoir si vous concevez votre système pour qu'il fonctionne sur les serveurs du client ou sur le cloud.
- Pour les produits de consommation qui ne sont pas de simples applications mobiles, il est presque toujours judicieux de développer pour le cloud.
- Pour les produits professionnels, la décision est plus difficile à prendre.
  - Certaines entreprises sont préoccupées par la sécurité du cloud et préfèrent exécuter leurs systèmes sur des serveurs internes.
  - Elles peuvent avoir un modèle prévisible d'utilisation du système, de sorte qu'il est moins nécessaire de concevoir votre système pour faire face à d'importantes variations de la demande.
- Un choix important que vous devez faire si vous exécutez votre logiciel dans le cloud est celui du fournisseur de cloud à utiliser.

Source ouverte

- Les logiciels libres sont des logiciels disponibles gratuitement, que vous pouvez changer et modifier à votre guise.
  - L'avantage est que vous pouvez réutiliser les logiciels plutôt que d'en créer de nouveaux, ce qui réduit les coûts de développement et les délais de mise sur le marché.
  - Les inconvénients de l'utilisation d'un logiciel libre sont que vous êtes limité par ce logiciel et que vous n'avez aucun contrôle sur son évolution.
- La décision d'utiliser des logiciels libres dépend également de la disponibilité, de la maturité et du soutien continu des composants libres.
- Les questions de licence de logiciel libre peuvent imposer des contraintes sur la façon dont vous utilisez le logiciel.
- Le choix d'un logiciel libre doit dépendre du type de produit que vous développez, de votre marché cible et de l'expertise de votre équipe de développement.

Outils de développement

- Les technologies de développement, telles qu'une boîte à outils de développement mobile ou un cadre (framework) d'application web, influencent l'architecture de votre logiciel.
  - Ces technologies ont des hypothèses intégrées sur les architectures du système et vous devez vous conformer à ces hypothèses pour les utiliser.
- La technologie de développement que vous utilisez peut également avoir une influence indirecte sur l'architecture du système.
  - Les développeurs privilégient généralement les choix architecturaux qui font appel à des technologies familières qu'ils comprennent.
  - Par exemple, si votre équipe a une grande expérience des BD relationnelles, elle peut les préférer à une BD NoSQL.