CONDUITE DE PROJET LOGICIEL

EXPRESSION DU BESOIN
SPECIFICATION ET
CONCEPTION

Expression du besoin

Si le besoin n'est pas clairement défini ...

... le produit risque de ne satisfaire le client qu'à moitié

Définition de la spécification du logiciel

Ensemble des activités consistant à définir de manière précise, complète et cohérente ce dont l'utilisateur a besoin.
C'est une des phases du cycle de vie.

(AFNOR: Vocabulaire de la Qualité du Logiciel).

D'une manière générale, les spécifications ont pour objet de répondre à la question:

«QUELLES EXIGENCES DOIT SATISFAIRE LE LOGICIEL?»

Elles donnent une description complète, précise et cohérente des interfaces, des données d'entrée et de sortie, des fonctions à réaliser, des contraintes opérationnelles, des contraintes de réalisation, etc.

Spécification de système vs. spécification de logiciel

LES SPECIFICATIONS DE SYSTEME:

..précisent les caractéristiques matérielles et logicielles du système apte à satisfaire les besoins de l'utilisateur.

Un système est une collection d'hommes, de machines et de méthodes organisés pour accomplir un ensemble de fonctions spécifiques.

(ISO)

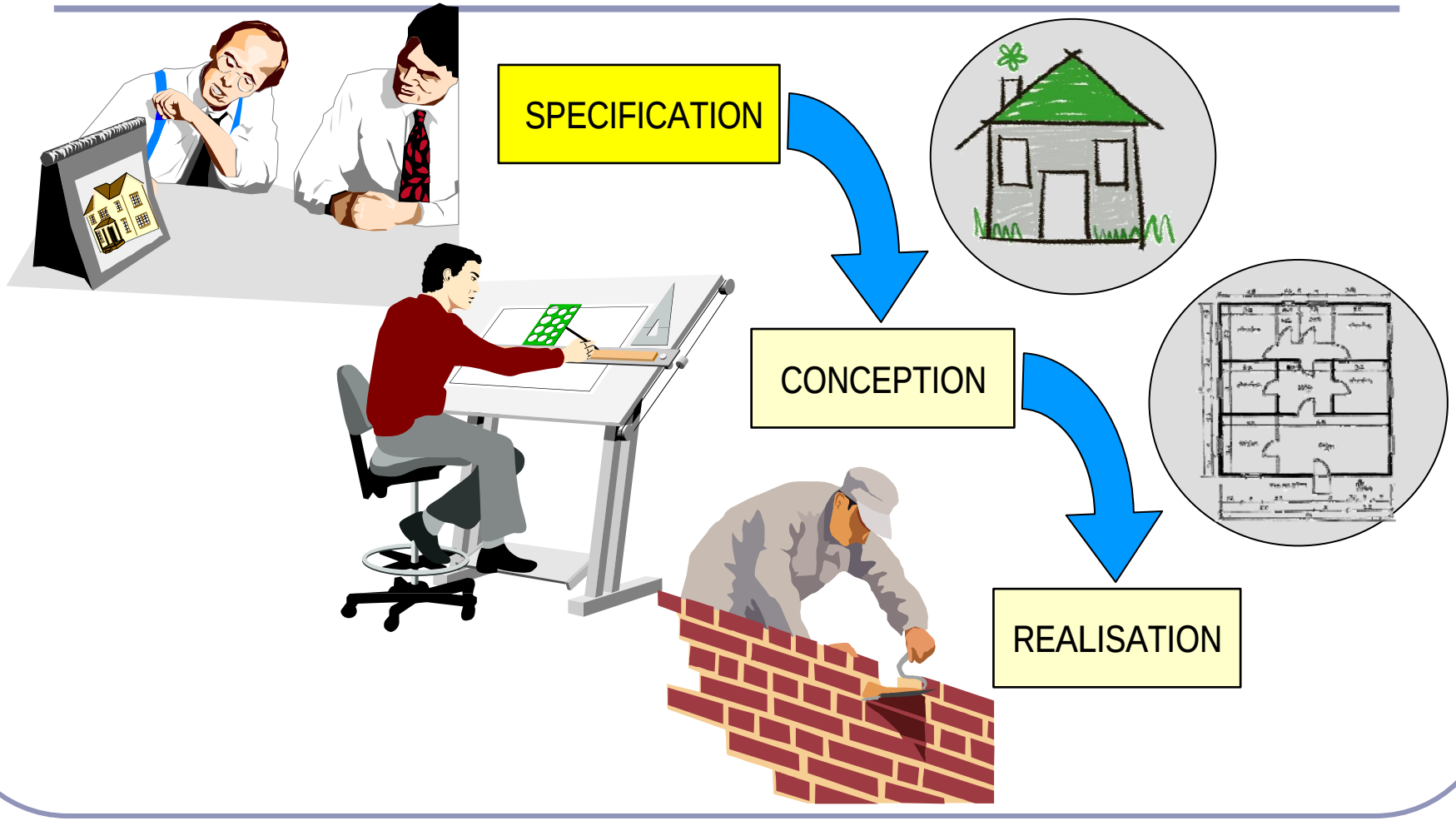
LES SPECIFICATIONS DE BESOINS DE LOGICIEL:

.. désignent l'expression des besoins et des contraintes de l'utilisateur (le client) vis à vis du développeur (le fournisseur).

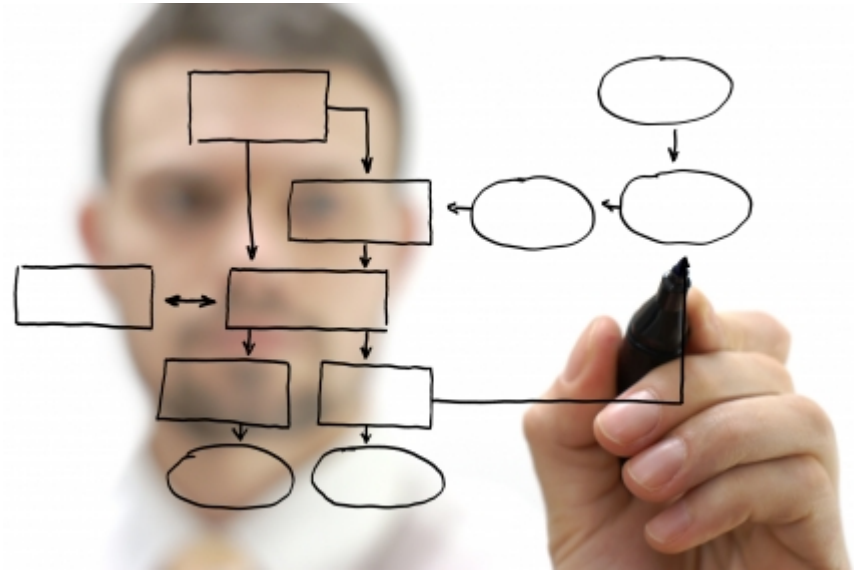
Un logiciel est un ensemble de programmes, procédés et règles, et éventuellement de la documentation associée, relatif au fonctionnement d'un ensemble de traitement de l'information

(arrêté du 22/12/81).

Place de la spécification dans le processus de développement



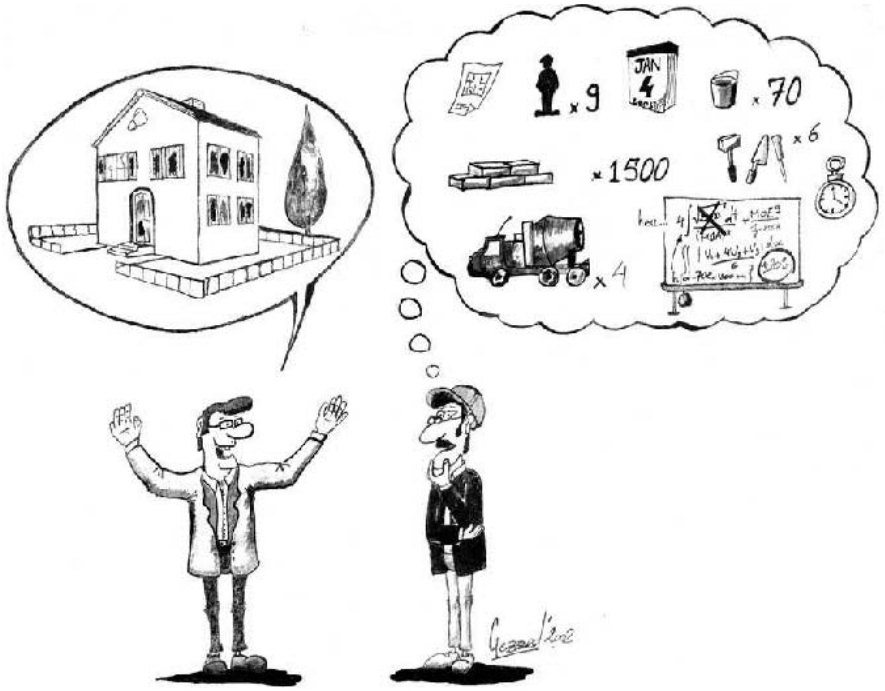
LA SPECIFICATION
Objectifs



- Exprimer le besoin des utilisateurs du logiciel
- Traduire le besoin sous forme d'exigences
  - fonctionnelles
  - d'interfaces
  - opérationnelles
  - de qualité
- Etablir une référence en ce qui concerne la définition du logiciel

Problématique de communication

- ANALYSER & COMPRENDRE le discours de l'utilisateur
- REDUIRE LA COMPLEXITE en quantifiant/modélisant/structurant
- FORMALISER ET COMMUNIQUER clairement
- EXPRIMER précisément les EXIGENCES



La spécification: Questions posées

- **A QUEL USAGE EST DESTINE LE LOGICIEL ?**
  - Quel est son environnement d'exploitation?
  - Quelles sont ses fonctions ?
  - Quelles sont ses limites et restrictions?
- **QUELLES SONT LES INFORMATIONS QUE LE LOGICIEL DOIT TRAITER ?**
  - Quelles sont les données à prendre en considération?
  - Comment faut-il les représenter?
- **QUEL DOIT ETRE LE COMPORTEMENT DU LOGICIEL?**
  - Quels sont les processus qui régissent le fonctionnement?
  - Quels sont les sollicitations et les critères de décision ?

Cadre contractuel de la spécification

Le but de la spécification est de

- **Traduire les besoins des utilisateurs du logiciel en exigences techniques (fonctionnelles, d'environnement, de qualité,...) exploitables par le développeur.**
- **Définir les clauses techniques du contrat passé entre le demandeur et le développeur.**
- **Etablir une référence pour la validation et le prononcé de qualification du logiciel.**

LA SPECIFICATION DOIT DONC OBLIGATOIREMENT ET EN TOUTE CONNAISSANCE DE CAUSE FAIRE L'OBJET D'UN ACCORD ENTRE LE CLIENT ET LE FOURNISSEUR

Contre exemple

Soit le texte suivant, extrait d'un cahier des charges:

(...) Les transactions d'interrogation-réponse véhiculent des articles. Ces articles sont mémorisés sur les mémoires d'une façon ou d'une autre. On décrit la façon dont ils doivent apparaître à l'extérieur. Un article se présentera comme une chaîne de champs ordonnés séparés par des drapeaux.

Il existera un minimum de 20 drapeaux parmi lesquels:

MA	Magasin
CD	Code denrée
TP	Taille ou pointure
DL	Délai de livraison
PU	Prix unitaire

(...) Les drapeaux seront codés sur 2 octets: un octet drapeau et un octet désignateur de drapeau.

Chaque champ sera rédigé en caractères de saisie dont la liste est jointe avec les codes correspondants.

(...) Certains champs des articles seront obligatoires, d'autres facultatifs. Certains champs distincts dans la liste ci-dessus peuvent en fait être confondus en un champ unique: par exemple CD - TP peuvent être remplacés par un code numérique unique (...)

Description fonctionnelle

- **Utilisateurs potentiels :**

  désignation et caractérisation des utilisateurs du logiciel développé

- **Environnement de mise en œuvre :**

  contexte d'utilisation ou d'exploitation du logiciel (description sommaire de la plate-forme cible, lieu physique d'implantation, exigences d'interactions avec d'autres systèmes, etc.)

- **Cas d'utilisation / scénarii clients:**

  1) Liste des différents types d'usage auxquels est destiné le produit = cas dans lesquels le produit peut rendre service à un utilisateur

  Diagramme UML ou expression sous la forme d'une phase courte commençant par « L'utilisateur X devra pouvoir …. ».

  Par exemple : « L'administrateur devra pouvoir installer une nouvelle base de données ».

  2) Hiérarchisation en concertation avec le client (compromis priorité/difficulté)

  3) Description détaillée de chaque cas d'utilisation sous la forme d'un scénario.

Qualités d'une spécification

- **COMPLETE**
- **MODIFIABLE**
- **TRACABLE**
- **COHERENTE**
- **VERIFIABLE**
- **NON AMBIGUE**
- **PRECISE**

Complétude (auto-suffisance)

Une spécification est complète si, et seulement si, elle satisfait les 4 conditions suivantes:

- Elle contient la définition de toutes les exigences relatives aux aspects **fonctionnels** et **opérationnels**, de toutes les **interfaces externes** et toutes les **contraintes de conception**.
- Elle contient la définition de **réponses fournies par le logiciel pour tous les types de sollicitations** et d'entrées possibles (valides ou invalides), dans tous les cas de figures envisageables.
- Elle respecte les **standards ou normes applicables à sa forme ou à son contenu**. Toute partie non applicable doit être référencée et la justification de sa non-application doit être fournie.
- Elle contient un **glossaire** ou un **dictionnaire** qui fournit une définition claire et précise de tous les labels, références, identificateurs, termes, etc. qui sont utilisés dans les textes, diagrammes, tables ou autres figures.

Facilité de modification

Une spécification est modifiable si sa structure et sa forme sont telles que, si une modification intervient, la spécification peut être mise à jour facilement, complètement et en préservant la cohérence.

2 règles élémentaires:

- formaliser et gérer la spécification

  Pour pouvoir répondre à la question « Faut-il modifier? »

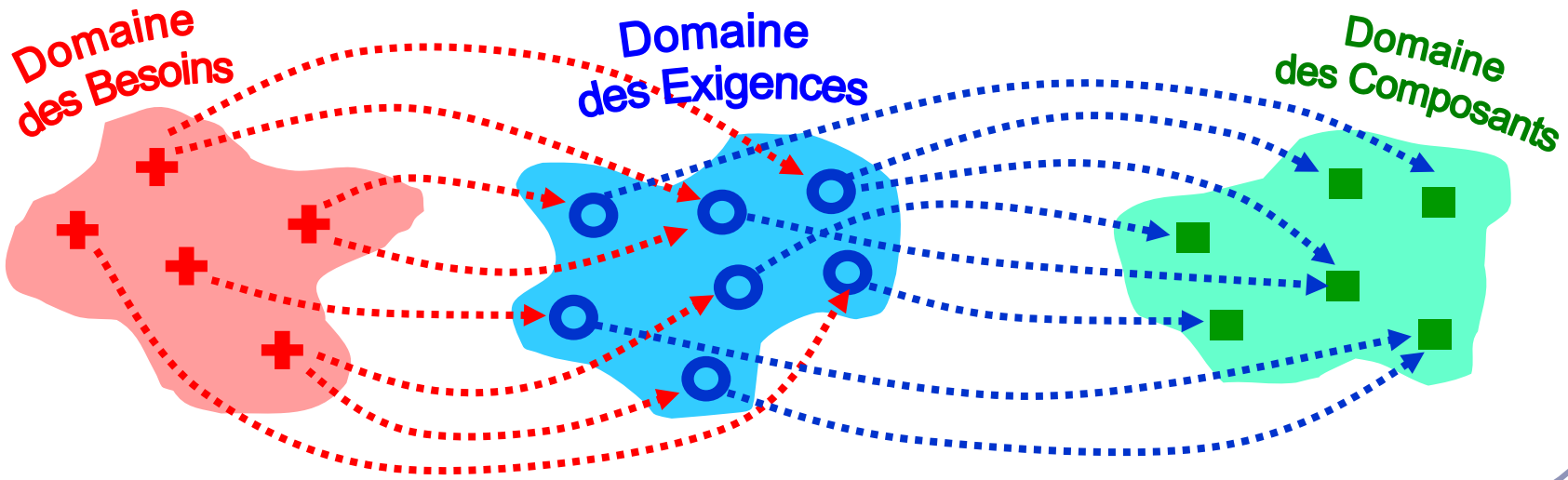
- organiser et rationaliser la structure du support de formalisation

  Pour pouvoir répondre à la question « Que faut-il modifier? »

Traçabilité

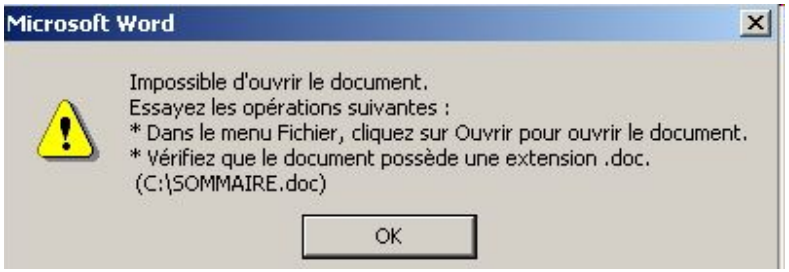
Une spécification est traçable si, seulement si,

- **Les besoins de l'utilisateur qui en sont l'origine peuvent être identifiés sans ambiguïtés**
- **Les composants logiciels créés pour sa mise en œuvre pourront être facilement recensés**



Cohérence

Une spécification est cohérente si, et seulement si, il n'existe aucune contradiction et aucun conflit entre les différentes propriétés qui la déterminent.



Vérifiabilité

Une spécification est vérifiable si, et seulement si, il existe un moyen (automatique ou manuel) de s'assurer que le logiciel répond aux exigences définies.

⇨ **EXEMPLE DE SPECIFICATION NON VERIFIABLE:**

- *"Le produit doit fonctionner correctement..."*
- *"Le produit doit avoir des performances satisfaisantes..."*
- *"Le temps de réponse ne doit pas être supérieur à 2 secondes en exploitation normale... »*

⇨ **EXEMPLE DE SPECIFICATION VERIFIABLE:**

- *"Le temps de réponse doit être inférieur à 2 secondes dans 70% des cas et inférieur à 5 secondes dans 100% des cas"*

Absence d ’ambiguïté

Une spécification est non ambiguë si, et seulement si, chaque énoncé de spécification a une et une seule interprétation possible.

EXEMPLE DE SPECIFICATION AMBIGUE:

"Dans l'entreprise, tous les projets sont gérés par un responsable."

peut signifier que

- *Un responsable gère l'ensemble des projets de l'entreprise ...*
- *Chaque projet est géré par son propre responsable ...*
- *Chaque projet est géré par un responsable qui peut gérer plusieurs projets ...*

Intérêt de la spécification formelle

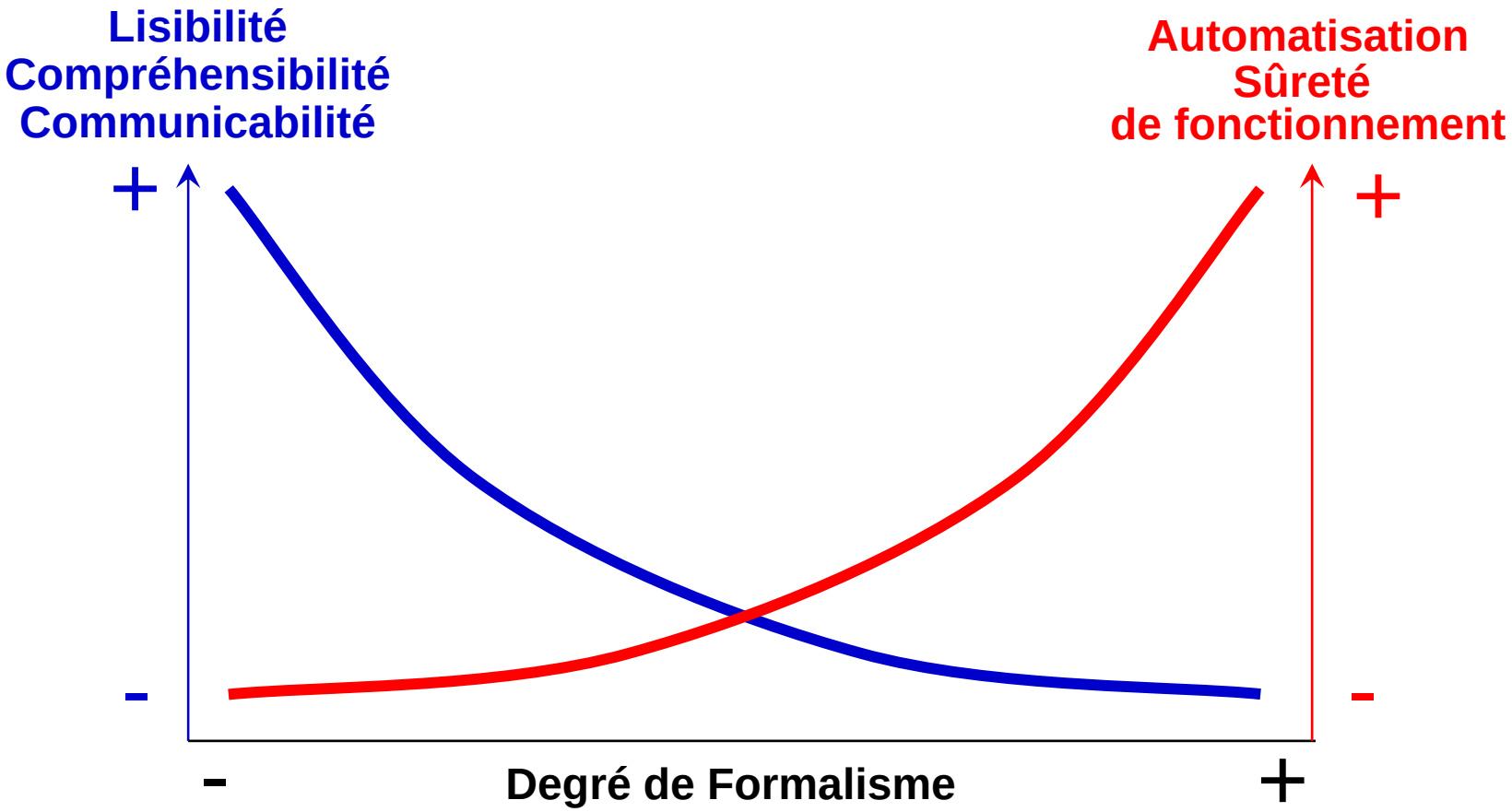
LE LANGAGE FORMEL DE SPECIFICATION

- remplace le langage naturel ou le précise
- est conçu pour exprimer un certain type de caractéristique
- supprime les ambiguïtés
- permet d'automatiser la production du logiciel
- permet de prouver que le logiciel répond à sa spécification
- peut être graphique

MAIS

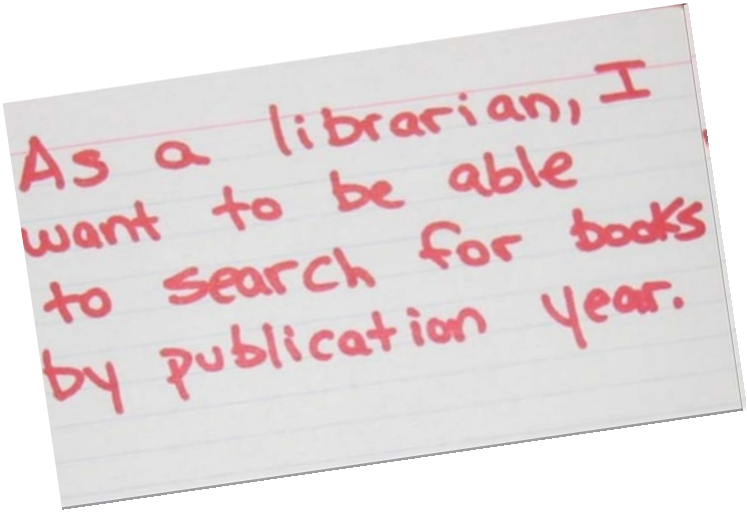
- requiert une formation ou un apprentissage
- est difficilement compréhensible par un lecteur non averti (cf. Spécifications algébriques ou axiomatiques, Réseaux de Pétri, etc.)

Degré de formalisme



User stories (histoire d'utilisateur)

- **User story:** possibilité offert à l'utilisateur du système, formulée en une ou deux phrases en langage naturel.



- Exemples:
  - En tant que bibliothécaire, je peux enregistrer un nouveau livre dans le catalogue.
  - En tant qu'abonné, je peux consulter le catalogue des livres de la bibliothèque.
  - En tant qu'abonné, je peux emprunter un livre s'il est disponible.
  - En tant qu'abonné, je peux réserver le livre que je souhaite emprunter.

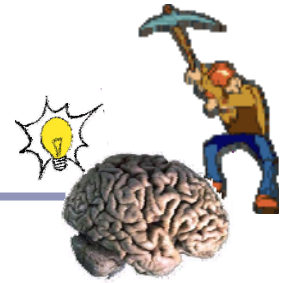
Synthèse des user stories

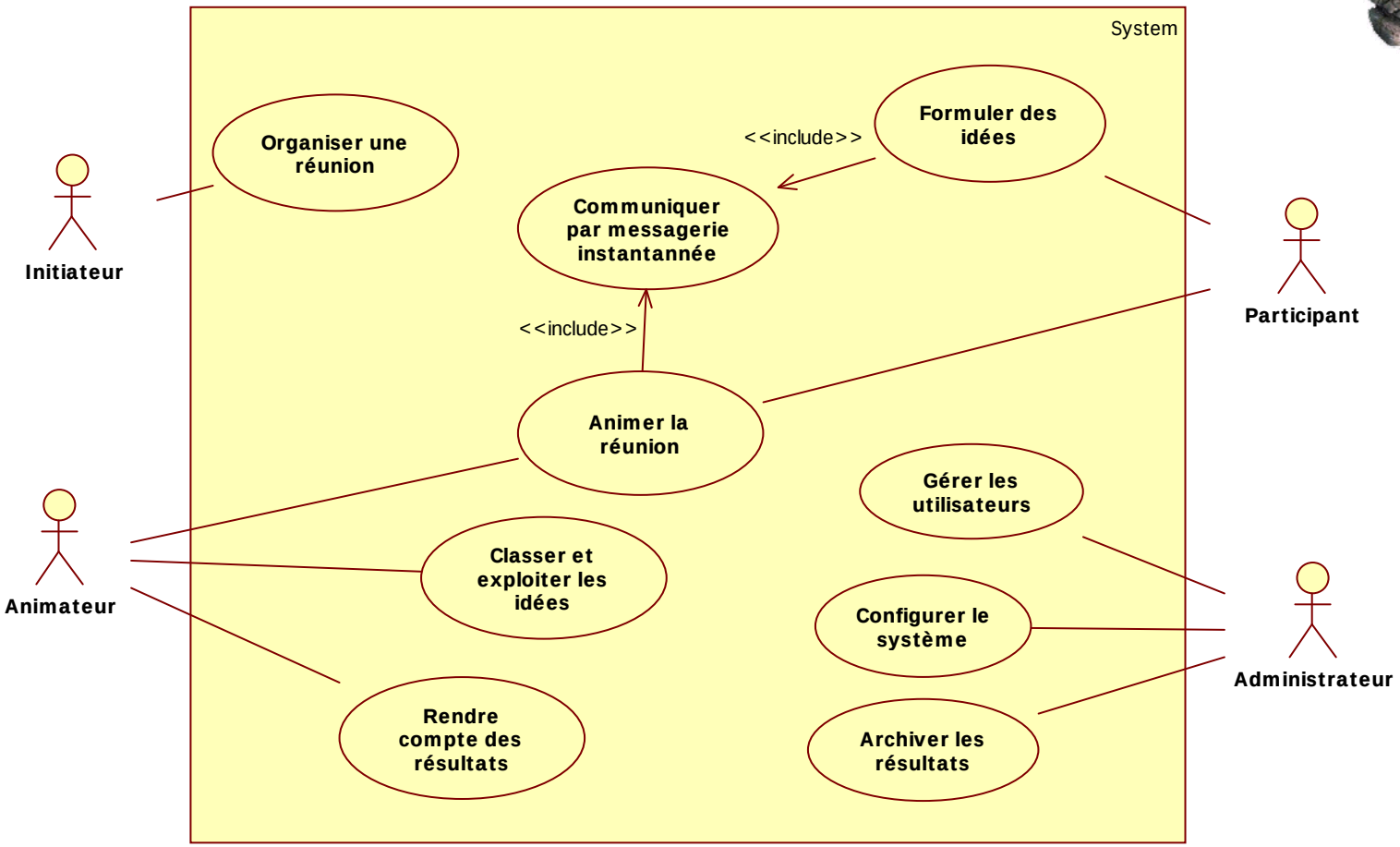
En tant que…	**je peux …**	**pour ..**
bibliothécaire	consulter la liste des fournisseurs agréés	accéder à leurs sites Web
bibliothécaire	commander des livres chez le fournisseur choisi	acquérir des nouveautés
bibliothécaire	entrer le compte d'imputation pour une commande	vérifier le budget
bibliothécaire	enregistrer une livraison	solder la commande associée et ajouter les livres dans le catalogue
abonné	Rechercher un livre dans le catalogue à partir d'un mot de son titre	connaître sa référence
abonné	Rechercher un livre dans le catalogue en fonction de son auteur	connaître sa référence

Cas d’utilisation (Use case)

- Définit une manière d'utiliser le système et permet d'en décrire les exigences fonctionnelles.
- Correspond à une séquence d'évènements qui décrit un système faisant quelque chose d'utile.
- Contient un ou plusieurs scénarios qui définissent les interactions entre le système et ses utilisateurs (appelés acteurs) pour atteindre un but ou une fonction spécifique
- Un acteur d'un cas d'utilisation peut être un humain ou un autre système externe à celui que l'on tente de définir.
- Un cas d'utilisation correspond à un ensemble autonome de fonctionnalités possédant une forte cohésion.
- Dans l'Unified Process, les spécifications d'un système sont décrites dans la vue des cas d'utilisation et les itérations sont basées sur la réalisation de cas d’utilisation.

Diagramme UML de cas d'utilisation





Formalisation des cas d'utilisation

Pour chaque cas d'utilisation

- Titre
- Priorité / importance
- Acteurs concernés
- Conditions de déclenchement
- Déroulement nominal
- Conditions d'arrêt
- Traitements des exceptions
- Exigences à satisfaire

Comparaison

User story	Use case
●Définit un but ●Correspond à un seul scénario ●Exprimé en langage naturel sur des cartes ●Ne nécessite pas une analyse détaillée ●Émerge rapidement dans le processus de définition ●Est discutée avec l'utilisateur ●N'est pas nécessairement conservée	●Définit une séquence d'actions ●Se décline en scénarii nominaux et alternatifs ●S'appuie sur UML ●Doit être décrit précisément ●Nécessite une étude analyse ●Est formalisé ●Est tracée tout au long du processus ●Aide à la définition ultérieure des tests ●Eventuellement lié avec d'autres cas

Code minimal de bonne conduite

REGLE 1: Il existe un document de spécification de besoin du logiciel

REGLE 2: Ce document contient au minimum une description externe du logiciel:
- fonctions
- interfaces
- performances, qualité
- contraintes de réalisation

REGLE 3: Ce document est un document de travail pour
- le demandeur du logiciel (client, utilisateur, maître d'œuvre)
- le réalisateur du logiciel (analyste, concepteur, équipes de validation)

REGLE 4: Il sert de référence
- en cas de litige entre le demandeur et le réalisateur
- pour les concepteurs et les codeurs
- pour les testeurs

REGLE 5: Il doit être validé et approuvé (en l'état) au début du projet puis éventuellement amendé en cours de projet.

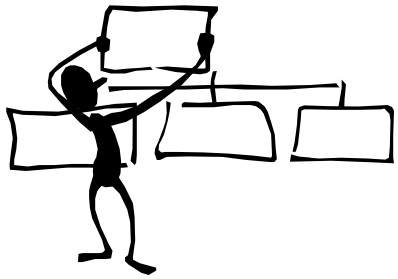
Les 7 péchés capitaux du spécifieur

- le **BRUIT**: élément de spécification n'apportant d'information sur aucune des caractéristiques du problème.
- le **SILENCE**: caractéristique du problème à laquelle ne correspond aucun élément de spécification
- la **SUR-SPECIFICATION**: élément de spécification correspondant à une caractéristique de la solution et non pas du problème
- l'**AMBIGUITE**: élément de spécification pouvant être interprété d'au moins 2 façons différentes
- la **CONTRADICTION**: caractéristique du problème définie de façons incompatibles par différents éléments de spécification
- la **REFERENCE EN AVANT**: élément de spécification faisant référence à un autre élément ou à une caractéristique du problème non encore défini
- le **VOEU PIEUX**: élément de spécification décrivant une caractéristique du problème de telle façon qu'il sera impossible de valider la solution retenue relativement à cette caractéristique

Les 4 commandements du spécifieur

- Tu travailleras par abstraction

  Abstraction: Opération intellectuelle qui permet d 'isoler une caractéristique, une propriété d 'un objet et de la considérer indépendamment des autres.



- Tu t'appuieras sur des modèles

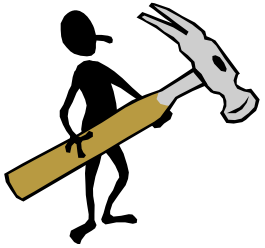
  Modèle = Représentation approchée d 'un système réel afin de pouvoir l étudier.

- Tu utiliseras des méthodes

  Méthode = Ensemble ordonné de manière logique de principes, de règles, d 'étapes permettant de parvenir à un résultat.

- Tu utiliseras des outils

  Outil = Instrument destiné à aider un opérateur dans la réalisation d 'une activité.

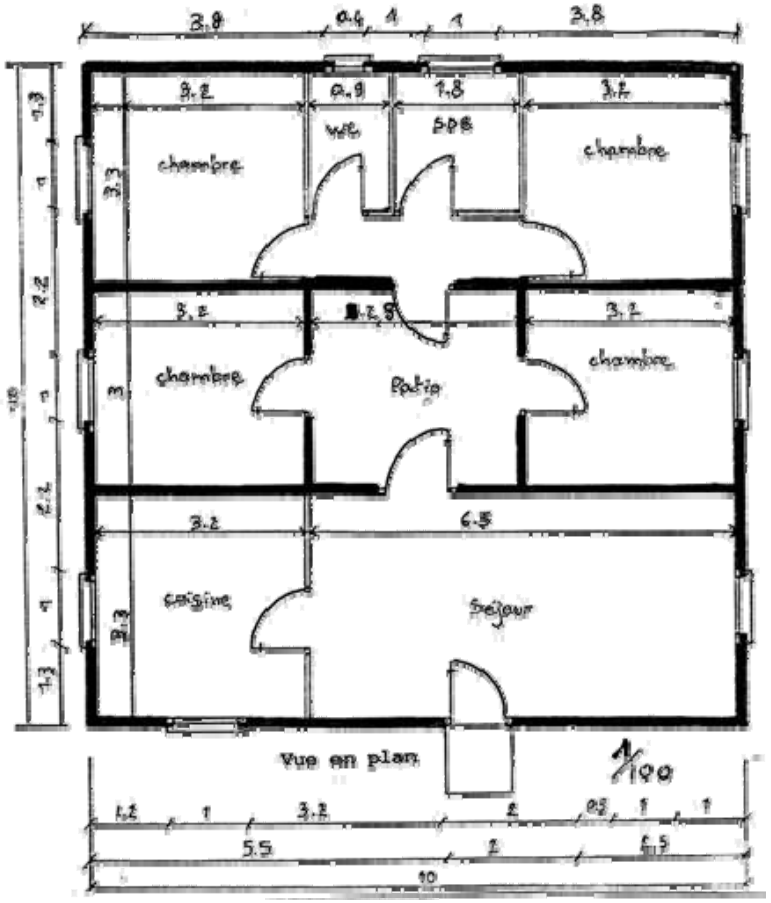


Plan type de document de spécification

1. Objet
2. Documents de référence
3. Terminologie et sigles utilisés
4. Exigences
- 4.1. Présentation de la mission du produit logiciel
- 4.2. Exigences fonctionnelles
- 4.3. Exigences opérationnelles
  - 4.3.1. Environnement
  - 4.3.2. Mise en oeuvre
- 4.4. Interfaces
  - 4.4.1. Interfaces avec des matériels
  - 4.4.2. Interfaces avec d'autres produits logiciels
  - 4.4.3. Interfaces avec des fichiers ou bases de données
  - 4.4.4. Interfaces homme-machine
- 4.5. Exigences concernant la conception et la réalisation
- 4.6. Traçabilité des exigences

5. Exigences concernant la qualification du produit logiciel
6. Préparation de la livraison
7. Logistique

Objectifs de la conception préliminaire



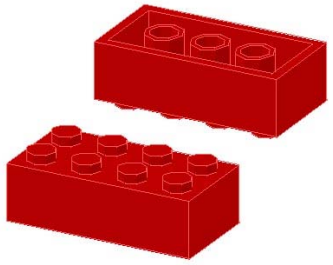
- Faire le choix d'une solution informatique en réponse à la spécification
- Définir l'architecture et spécifier les constituants du logiciel
- Organiser et préparer le développement des constituants du logiciel

Architecture

L ’architecture formalise un ensemble de décisions et de dispositions significatives concernant:

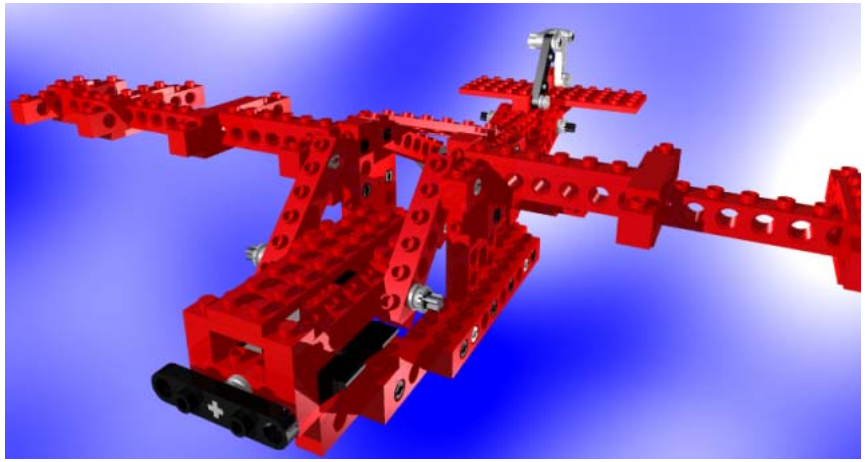
- **Le choix des éléments structuraux composant le logiciel,**
- **le style architectural guidant l’organisation des éléments,**
- **les mécanismes de collaboration entre les différents éléments,**
- **la définition des interfaces et des comportements des éléments,**
- **les préoccupations relatives au développement, à l’intégration et au déploiement de chaque élément.**

Modularité

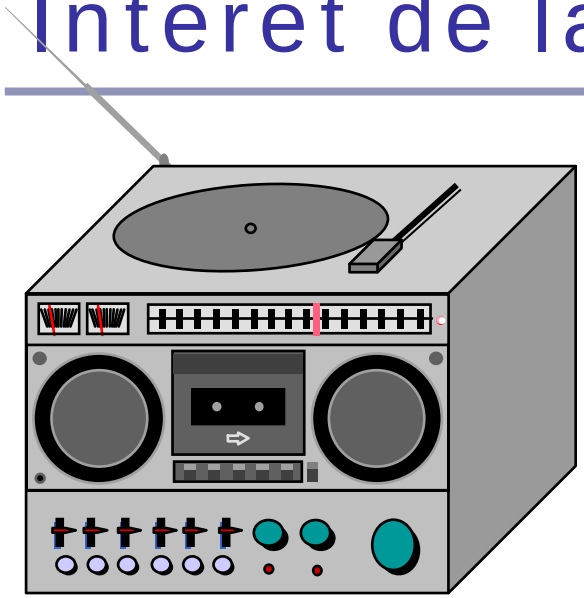


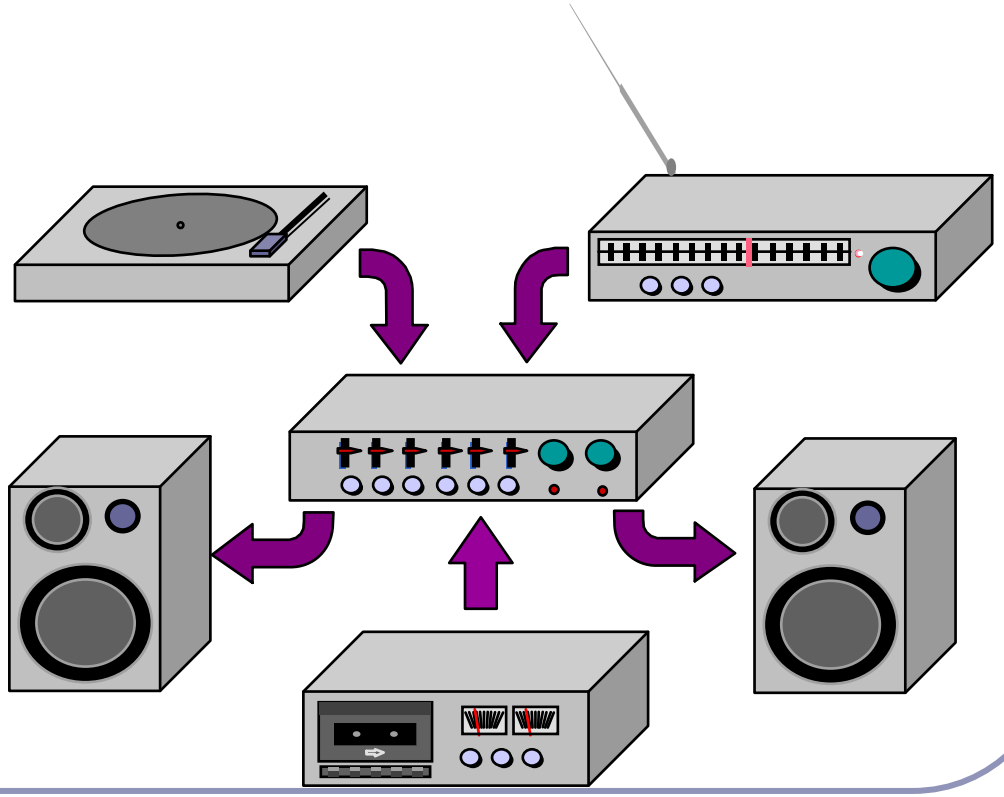
•Construction d'un système complexe par assemblage de briques/modules.
•Conception de briques modulables (faiblement couplés, cohésifs)

•Définition d'un système modulaire dans lequel un module peut facilement être remplacé par une autre.



Intérêt de la modularité

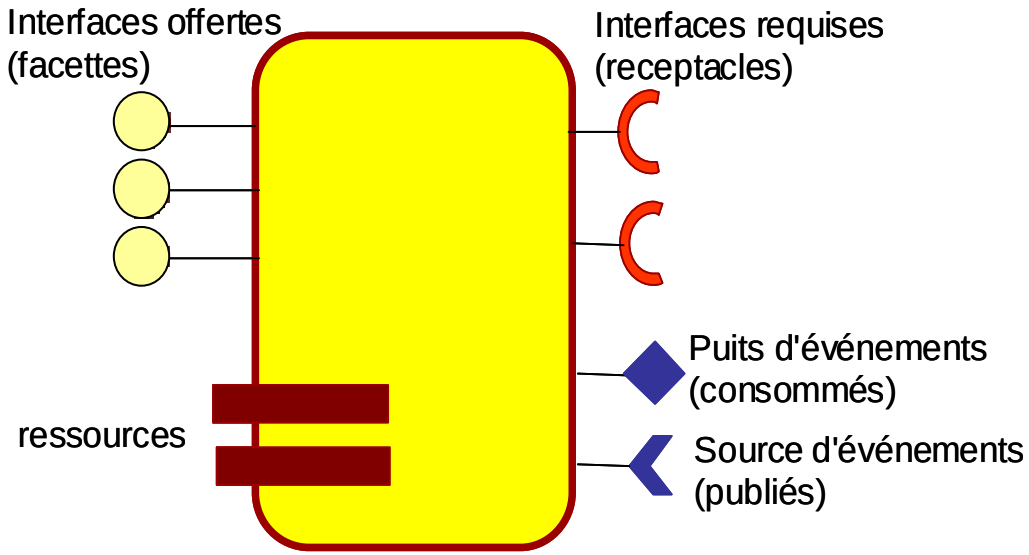




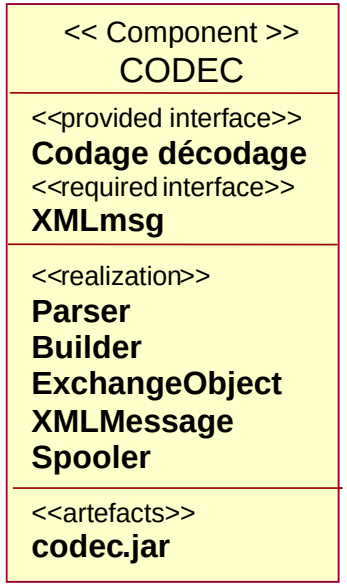
Qu'est-ce qu'un composant logiciel?

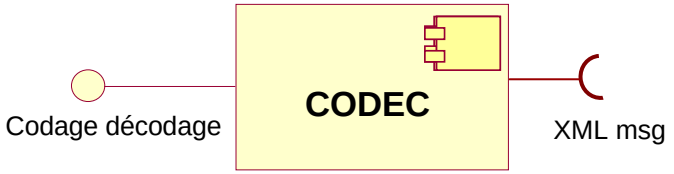
- **Un composant logiciel est une entité physique, autonome, indissociable et remplaçable qui réalise un ensemble d'interfaces clairement définies au travers desquelles il peut interagir avec d'autres composants.**
- **Un composant logiciel doit pouvoir être déployé seul (même s'il est dépendant d'autres composants) pour mise en œuvre dans le cadre d'une composition réalisée par une tierce entité (sans accéder au code d'implémentation).**

Modèle abstrait d'un composant



Notation UML 2.0





Quelques modèles de composants connus

- **Java Bean**

  Classe Java construite selon un "pattern" permettant l'assemblage et la réutilisation dans une application.

- **Active X / Composant .Net**

  Composant (généralement) visuel pour environnements Microsoft.

- **EJB**

  Composant transactionnel pour plate-forme J2EE implémentant une logique métier et exécutable dans un "serveur d'application".

- **CORBA**

  Spécification OMG X-plates-formes – X-langages. Diverses implémentations commerciales et libres (Visibroker, Orbix, Orbacus, OpenORB, JavaIDL, TAO, …).

- **DCOM +**

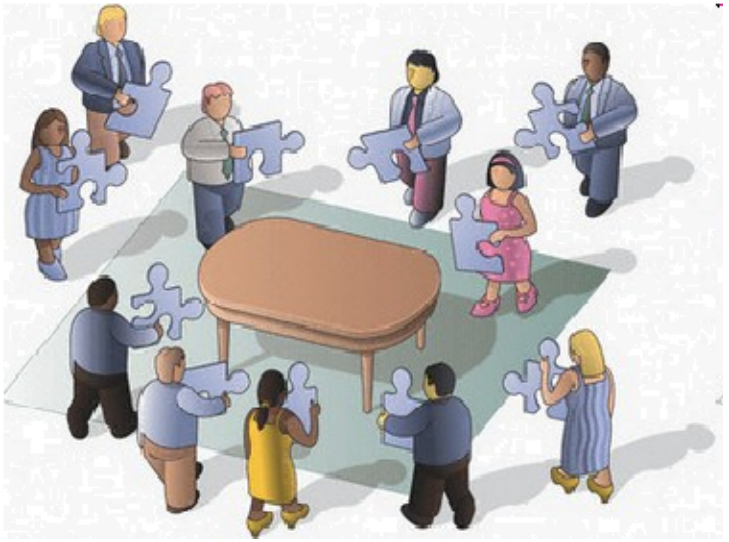
  Modèle de composant Microsoft équivalent à CORBA pour plate-forme Win32.

- **WEB Services**

  Modèle d'architecture orienté services (SOA) reposant sur des standards W3C (XML, SOAP, WSDL, UDDI, …)

Développement collaboratif

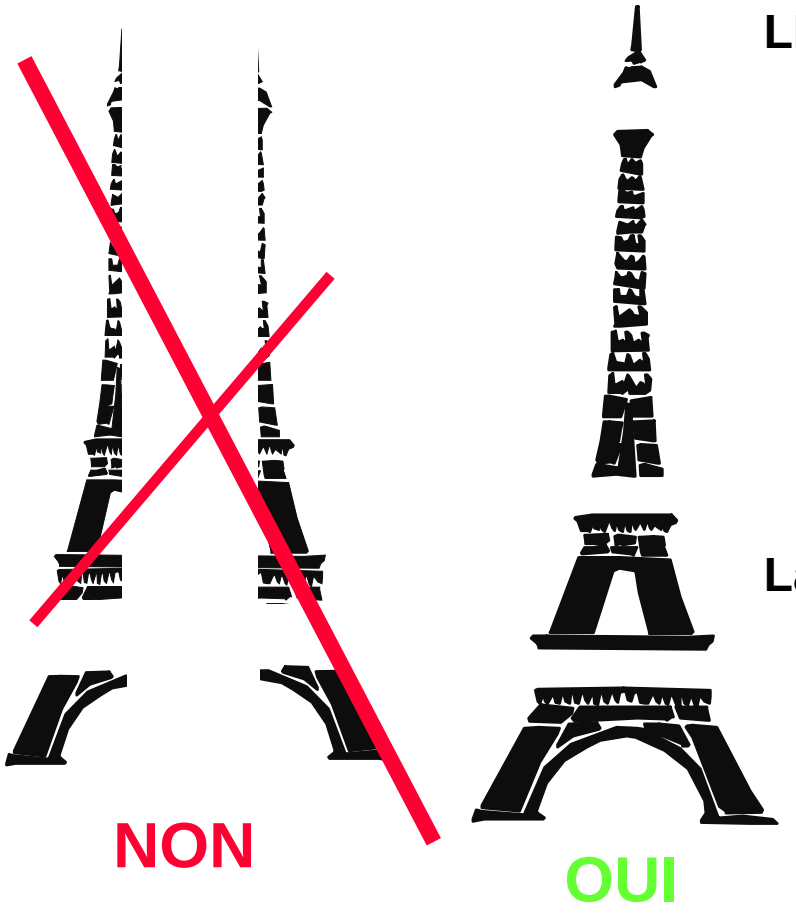
- Chaque individu/partenaire/prestataire est fournisseur d'un composant/service.



- Chaque composant/service est livré/mis à disposition de la communauté ou de l'intégrateur.

- L'intégrateur assemble/orchestre les composants/services disponibles pour constituer les applications.

Influence de la décomposition



LE MODULE EST

- **une unité de conception**
- **une unité de codage et de test**
- **une unité d'intégration**
- **une unité de gestion**
- **une unité de réutilisation et de modification**

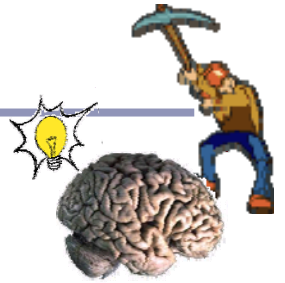
La structure issue de la décomposition en modules détermine

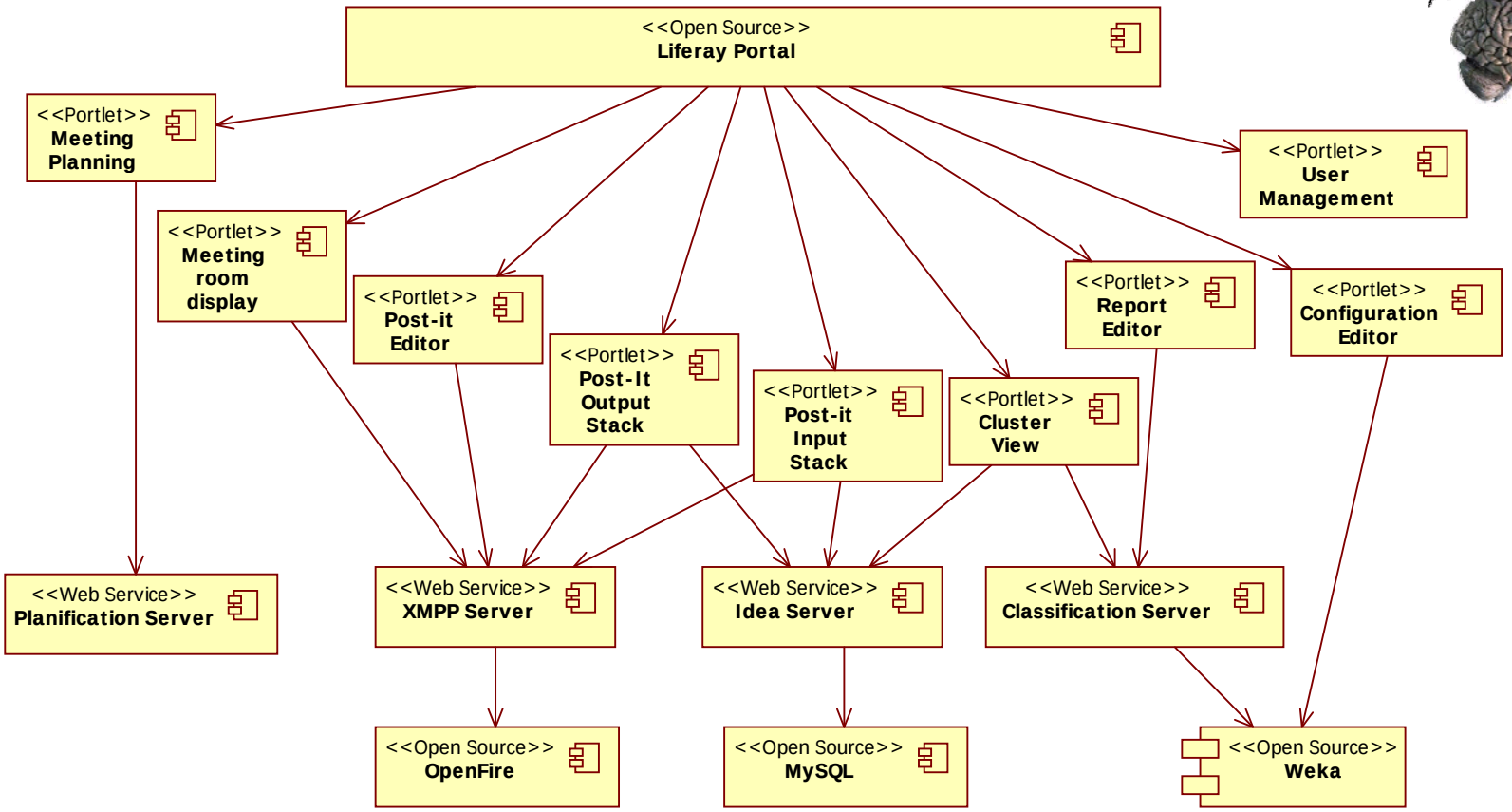
- **la logique du développement**
- **la stratégie d'intégration**
- **l'organisation des équipes**
- **etc.**

Formalisation de l ’architecture

- **VISION STATIQUE**
  - ***Elaborer la solution technique et organiser son développement***
  - Quels sont les constituants du logiciel (les **modules**) à fabriquer pour satisfaire les exigences de spécification ?
  - Quels sont leurs rôles respectifs?
  - Quelles sont leurs interrelations?
- **VISION DYNAMIQUE**
  - ***Décrire les processus d’exécution et les inter-actions avec l’environnement de mise en oeuvre***
  - Quels sont les comportements possibles du logiciel?
  - Quand et comment sont ils déclenchés?
  - Comment les composants collaborent ils entre eux?
  - Quelles sont les ressources utilisées?

Structure statique





Fonctionnement dynamique

Pour chaque scénario de cas d'utilisation

- Liste des composants mis en jeu
- Séquence d'appels / interactions
- Services rendus par chacun des composants
- Flux de données échangées

Le document d'architecture du logiciel

1. **Identification**
2. **Terminologie et sigles utilisés**
3. **Structure statique**
4. **Allocations fonctionnelles et spécifications**
5. **Description des composants**
    - 5.1. Composant 1
        - 5.1.1. Nom
        - 5.1.2. Rôle
        - 5.1.3. Références externes
        - 5.1.4. Opérations externes
            - 5.1.4.1. Opération 1
            - 5.1.4.2. Opération 2
            - .....
        - 5.1.5. Données externes
    - 5.2. Composant 2
    - .....
6. **Aspects dynamiques**
7. **Données partagées**
8. **Annexes**

XP: Autres pratiques de définition

- Tests de recette
  - *Expression du besoin par le client au travers des tests de validation (en relation avec les user stories)*
- Remaniement (refactoring)
  - *Réaménagement des programmes, reprise et amélioration permanente du code*
- Développement piloté par les tests unitaires
  - *Automatisation des tests qui sont conçus en même temps que le code. Un programme terminé est un programme qui a passé tous les tests avec succès.*

« Usure » du code

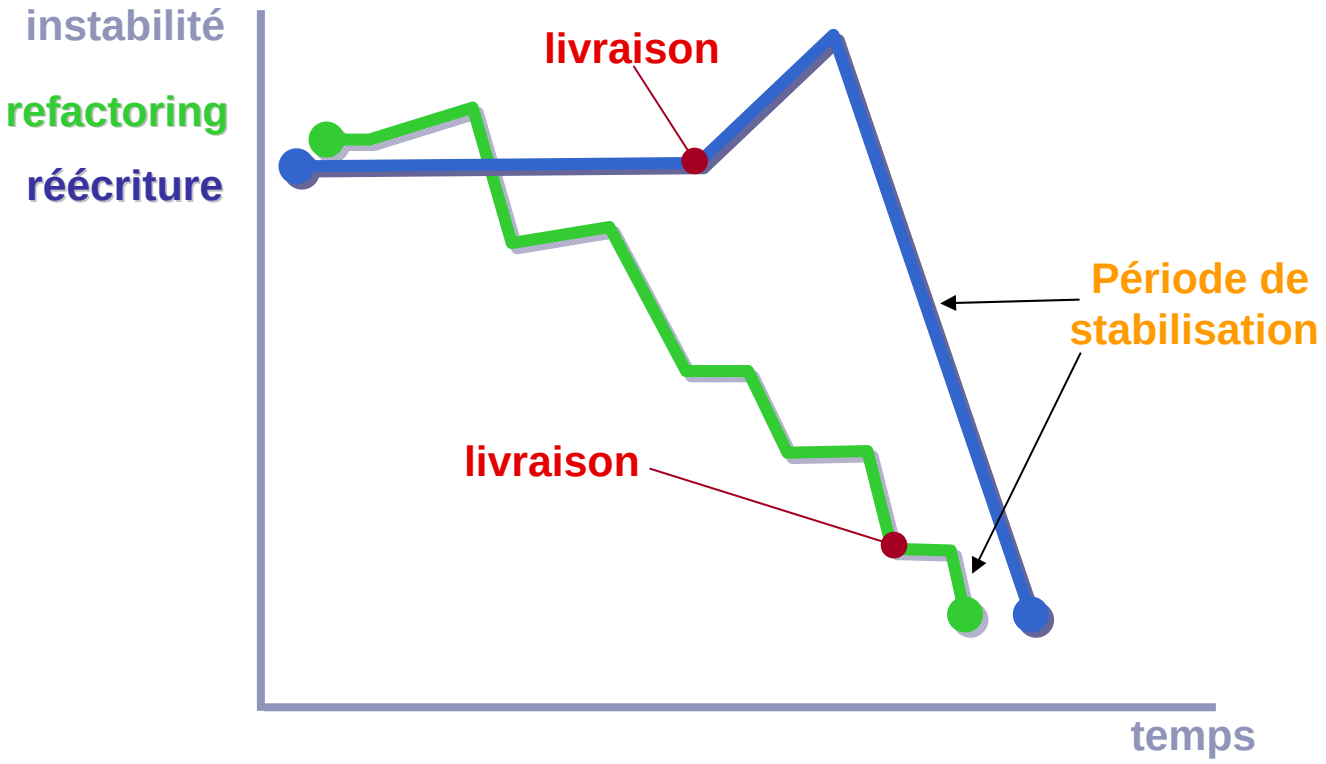
Le code source des grands logiciels se détériore au fil du temps:

- **Ajout de fonctionnalités** ⇒ augmentation de la complexité
- **Modifications temporaires (patches)** ⇐ Parer au plus pressé
- **Turn-over des développeurs** ⇒ Les nouveaux n'ont pas l'historique des décisions et des choix de conception.
- **Croissance de l'équipe de développement:** Plus de monde ⇒ moins de communication ⇒ choix non concertés ⇒ conception incohérente

Exemples de refactoring

- Supprimer le code dupliqué
- Supprimer le code « temporaire »
- Supprimer le code d'instrumentation
- Simplifier un algorithme
- Décomposer une méthode dont le code est trop long
- Décomposer une classe et répartir les responsabilités
- Réduire le nombre de paramètres d'une méthode
- Confiner les sections de code sensibles ou « changeantes »
- Réduire le couplage entre deux classes
- Simplifier les structures de contrôle
- Factoriser la définition des interfaces
- Rationaliser les relations d'héritage
- Etc.

Refactoring ou réécriture



Réingénierie et rétro-conception