POOurquoi ??

- La plupart des projets informatiques se passent mal !
  - dépassement important des <u>délais</u>
  - dépassement important des <u>coûts</u>
  - non <u>conformité</u> :
    - bugs,
    - limitations,
    - "flou" car comportement attendu "discutable" (imprécision, quiproquo, ...)
  - difficulté et coût important de de <u>maintenance</u>

POOurquoi : qualité

- Qualité insuffisante !

  - la qualité concerne bien sûr toute la chaîne :
    - expression de besoins
    - cahier des charges fonctionnel
    - préparation des cas tests
    - organisation des recettes,
    - …
  - qualité et l'implication des participants !!!
  - qualité et efficacité des organisations

  - Qualité du développement logiciel : conception, développement, cycle de vie du logiciel

Histoire...

- crise du logiciel en 1968.
  - baisse significative de la qualité des logiciels
    ⇗ puissance de calcul des ordinateurs
    ⇗ complexité des logiciels
  - Retards, dépassement de coût,
    ⇘ fiabilité,
    ⇘ performance
  - Coût important de maintenance
- coût du matériel informatique ⇘ // coût du logiciel ⇗
  - études sur les méthodes de travail adaptées à la complexité des logiciels contemporains => **génie logiciel**
- l'utilisation des méthodes de génie logiciel se répand *doucement* dans l'industrie du logiciel.

Vie d'un logiciel

- POURQUOI: Besoins / exigences
- QUOI: Cahier des charges fonctionnel
- COMMENT : Architecture
- COMMENT : Conception Spécifications techniques
- Dev + tests unitaires + intégration
- Maintenance corrective / évolutive
- Réutilisation / évolutions majeures / refonte

~~Peuvent être flous~~, et changer !
Maturité, évolution marché, contexte entreprise, réorganisation, …

Vie d'un logiciel

- POURQUOI: Besoins / exigences
- QUOI: Cahier des charges fonctionnel
- COMMENT : Architecture
- COMMENT : Conception / Spécifications techniques
- Dev + tests unitaires, + tests d'intégration
- Maintenance corrective
- Réutilisation / évolution / refonte

~~Peuvent être imprécis, irréalistes, incomplet~~, et changer !
Maturité, évolution marché, contexte entreprise, contexte réglementaire, ...

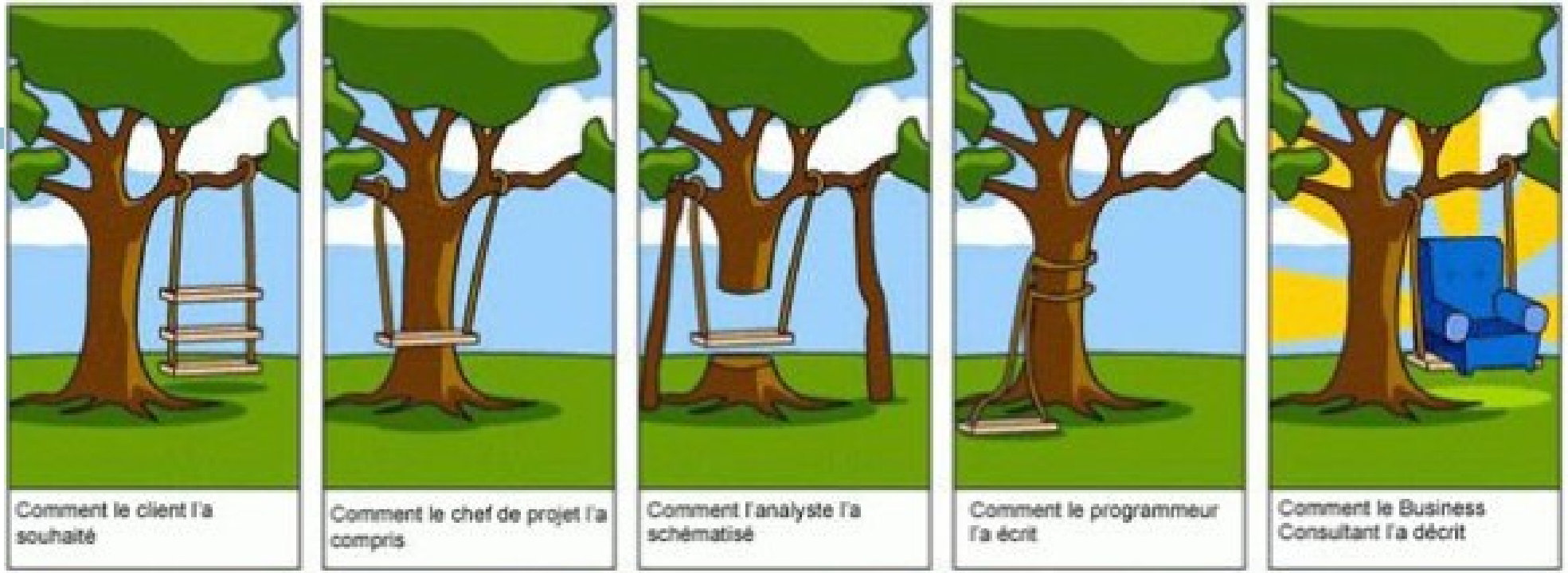
Vie d'un logiciel

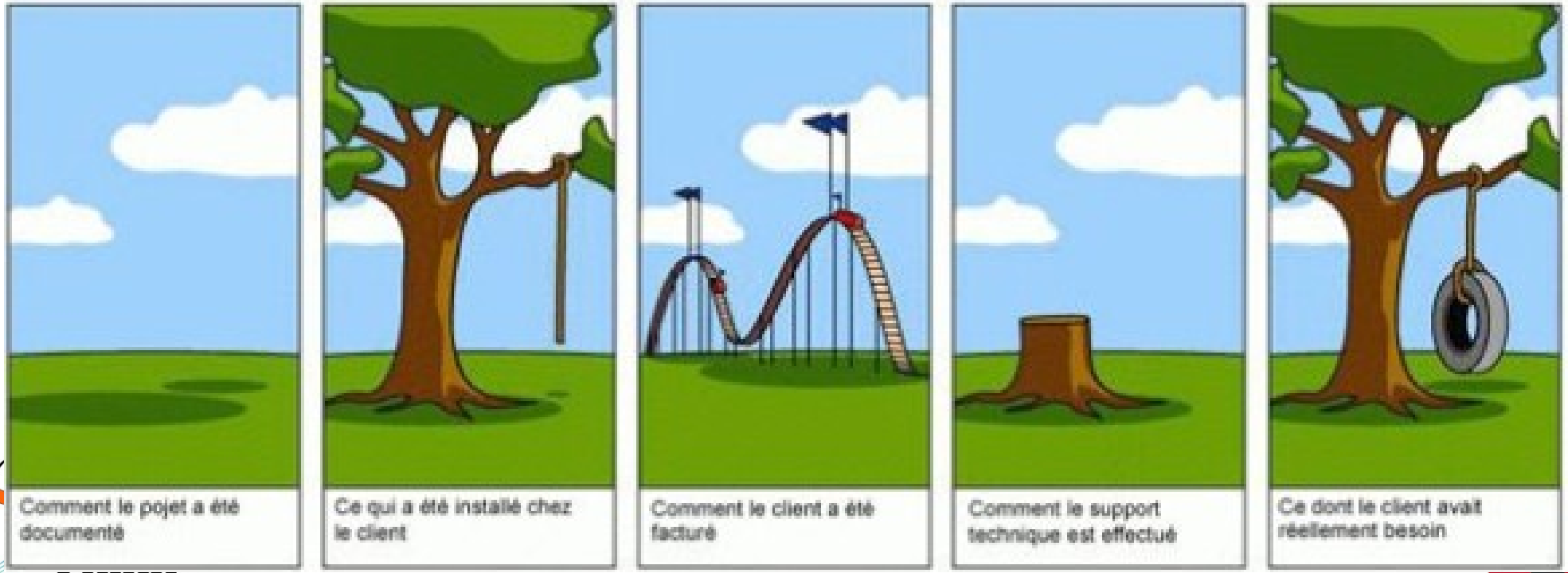
- POURQUOI: Besoins / exigences
- QUOI: Cahier des charges fonctionnel
- COMMENT : Architecture
- COMMENT : Conception, Spécifications techniques
- Dev + tests unitaires, + tests d'intégration
- Maintenance corrective / évolutive
- Réutilisation / refonte

CE QU'ON NE VEUT PAS CHANGER
LES FONDATIONS

Vie d'un logiciel

- POURQUOI: Besoins / exigences
- QUOI: Cahier des charges fonctionnel
- COMMENT : Architecture
- COMMENT : Conception, Spécifications techniques
- Dev + tests unitaires, + tests d'intégration
- Maintenance corrective / évolutive
- Réutilisation / évolutions majeures / refonte


Comment le client l'a souhaité
Comment le chef de projet l'a compris
Comment l'analyste l'a schématisé
Comment le programmeur l'a écrit
Comment le Business Consultant l'a décrit


Comment le pojet a été documenté
Ce qui a été installé chez le client
Comment le client a été facturé
Comment le support technique est effectué
Ce dont le client avait réellement besoin

Architecture

- Une description de niveau supérieur du logiciel:
  - les composants principaux et la manière dont ils interagissent
  - le découpage de plus haut niveau d'un système en parties/sous-systèmes/composants
- Qui prend en compte le matériel, les autres logiciels et l'utilisateur, les canaux de communication
- Sans toutes ces information une erreur fondamentale peut arriver

- Du bon sens : *« Prendre les décisions qui seront dures à changer »*
  - *CE QUI NE DOIT PAS CHANGER !*

Analyse / Conception fonctionnelle

- Première étape de la conception
  - Définit les fonctions/responsabilités des composants d'un système
  - Décrit leurs interactions fonctionnelles

C
R
C

- Fonctions principales, sans préjuger des solutions
  - « Lier » les deux parties et non « visser »...
  - «réduire la hauteur de l'herbe » vs « tondre »

U
C
D

- Prise en compte des contraintes
  - Sécurité
  - Respect de standard, de normes
  - Fixée par le client...

Savoir oublier les détails techniques !
Savoir oublier le "Comment" !

Conception

- Définition générale :
  activité créatrice qui consiste à élaborer un projet ou une partie des éléments le constituant, en partant
  - des besoins exprimés
  - des moyens existants
  - des possibilités technologiques

dans le but de créer un produit ou un service

Conception descendante top-down design

- Le problème global est décomposé en sous-problèmes, eux-même décomposés en opérations plus élémentaires jusqu'à obtenir la granularité souhaitée

Ce qu'on vous a appris depuis des années !

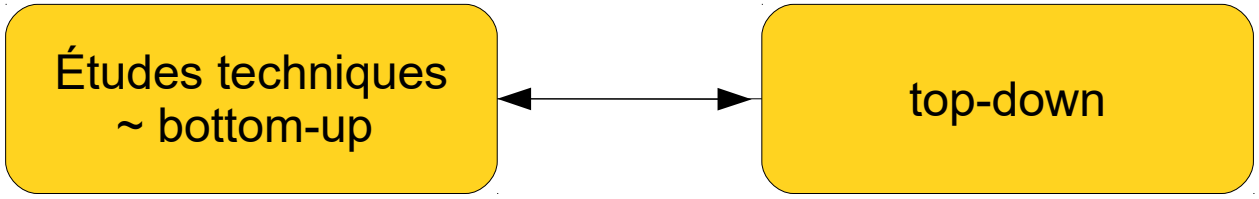
Symbole de l'esprit de synthèse et de l'analyse

Conception ascendante bottom-up design

- On commence par définir les fonctions les plus élémentaires, pour ensuite les utiliser et définir des fonctions de plus en plus spécifiques et complexes

D'abord savoir se servir de ses outils avant de construire sa maison

- La pratique est souvent un mix des 2 approches



Conception de système

- La conception de système inclut
  - architecture matérielle
  - architecture logicielle
    - définition des composants, et des problèmes qu'ils doivent résoudre
    - Recherche de solution
  - définition des interfaces
  - définition des *données*

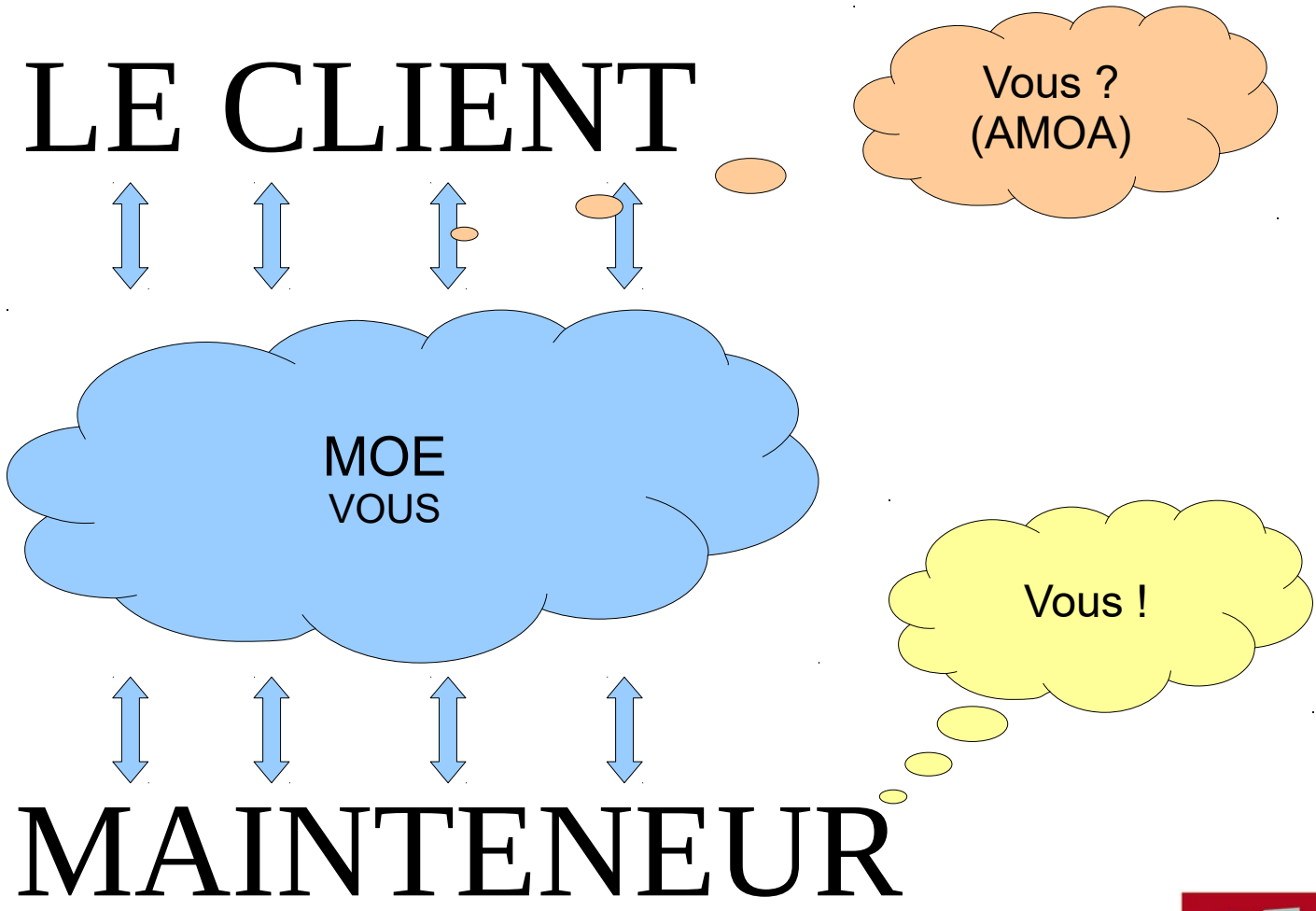
- Recherche de solutions

Et alors ??

- La définition de conception ne nous donne pas d'idée sur comment en faire une...
- Il faut traduire les exigences en solutions

- Analyse = comprendre les exigences
- Concevoir = construire la solution

Contexte lors de la conception trois AXES majeurs



Client / MOE / Mainteneur

- Client
  - Veut être satisfait
  - Est celui qui paye !
  - N'est pas concerné par « comment » mais uniquement par « quoi »
- Mainteneur
  - Veut en faire le minimum !
  - SAIT que :
    - « changer = temps + bugs + régression + ... »
      => douleur
  - → VEUT minimiser les changements
- MOE – conçoit /réalise
  - Doit respecter ses engagements
    - Conformité
    - Coût
    - Délai

MOE

- Création du logiciel, la MOE a du bon sens :
  - Cherche évidemment à éviter d'avoir trop de lignes de code → augmentation temps codage/tests
  - Mais veut que ça marche "ici et maintenant" pour respecter les engagements
    - N'est pas *directement* concerné par la suite …
- Importance que la MOE prenne aussi en compte les besoins du Mainteneur !
  - Question d'organisation et/ou contractuelle

Le changement commence dès la création du logiciel
→ la MOE est donc obligée de le prendre en compte

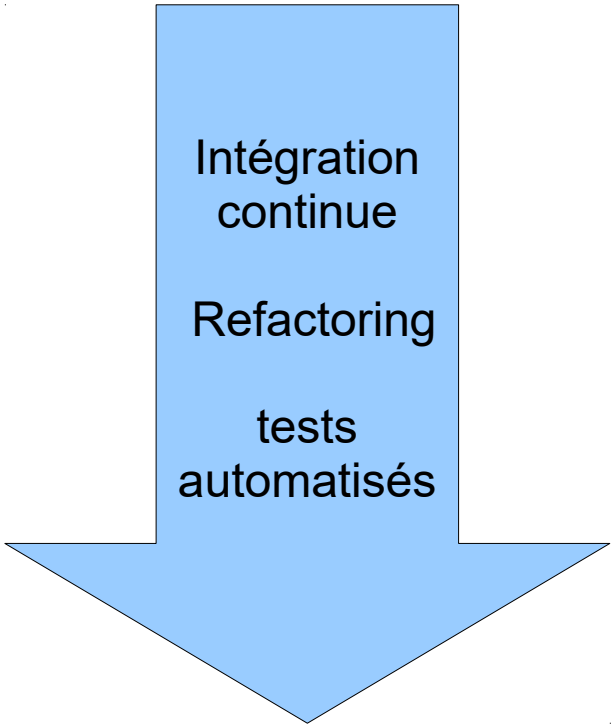
La vie d'un logiciel
Changements → risques & coûts

- Besoins de changements → 3 risques principaux
  - Rigidité
    - Les changements impactent de nombreux modules → compliqué donc coût élevé
    - difficile de faire évoluer le produit
  - Fragilité
    - Quand on touche quelque part, on casse ailleurs !
    - Les changements sont de plus en plus risqués
    - Coût très important, nombreuses régressions
    - (syndrôme spaghetti)
  - Immobilité
    - Impossibilité de réutiliser une partie du logiciel, impossible "d'extraire" un sous-ensemble cohérent.
    - il faut toujours redémarrer *from scratch*

Les risques sont liés aux impacts des changements.
Plus de dépendances → plus d'impacts:
→ risques ↗
→ coûts ↗

Gestion des dépendances

Une préoccupation systématique !



Build & co (make, maven, nexus)

Conception - POO

Design patterns

L'essentiel

- Les difficultés
  - Supporter le changement
  - Penser aussi à la Maintenance
  - Gestion des dépendances

- Limiter les risques
  - Brainstorming (CRC, …)
  - Use Case Diagram – compréhension des besoins
  - Conception préalable – schémas UML
  - *POO comme règles de ~~vie~~ … de design*
  - *DP comme aide / repères / idées*