

# **Ingénierie du Logiciel**

## **Master 1 Informatique – 4I502**

### **Cours 4 : Analyse (Fin)**

### **Séquences d'Analyse, Tests de Validation**

**Yann Thierry-Mieg**  
**[Yann.Thierry-Mieg@lip6.fr](mailto:Yann.Thierry-Mieg@lip6.fr)**

# Fiches Détaillées : Factorisation

- Exemple : Editer Liste étudiants
  - Ajouter, Modifier, Supprimer des étudiants
  - Modélisation avec des ALT :
    - SN :
      - 1. Le système affiche la liste actuelle
      - 2. L'acteur choisit créer un étu,
      - 3. Le système affiche un formulaire étudiant :....
      - 4. L'acteur renseigne les champs, valide
      - 5. Le système enregistre la modification
    - ALT : 1. En SN2, L'acteur choisit un étudiant dans la liste.  
2. retour en SN3 avec les données de l'étudiant chargées
    - ALT 2 : 1. En SN4, l'acteur choisit de détruire l'étudiant 2.  
valide 3. retour en SN5
    - +EX...

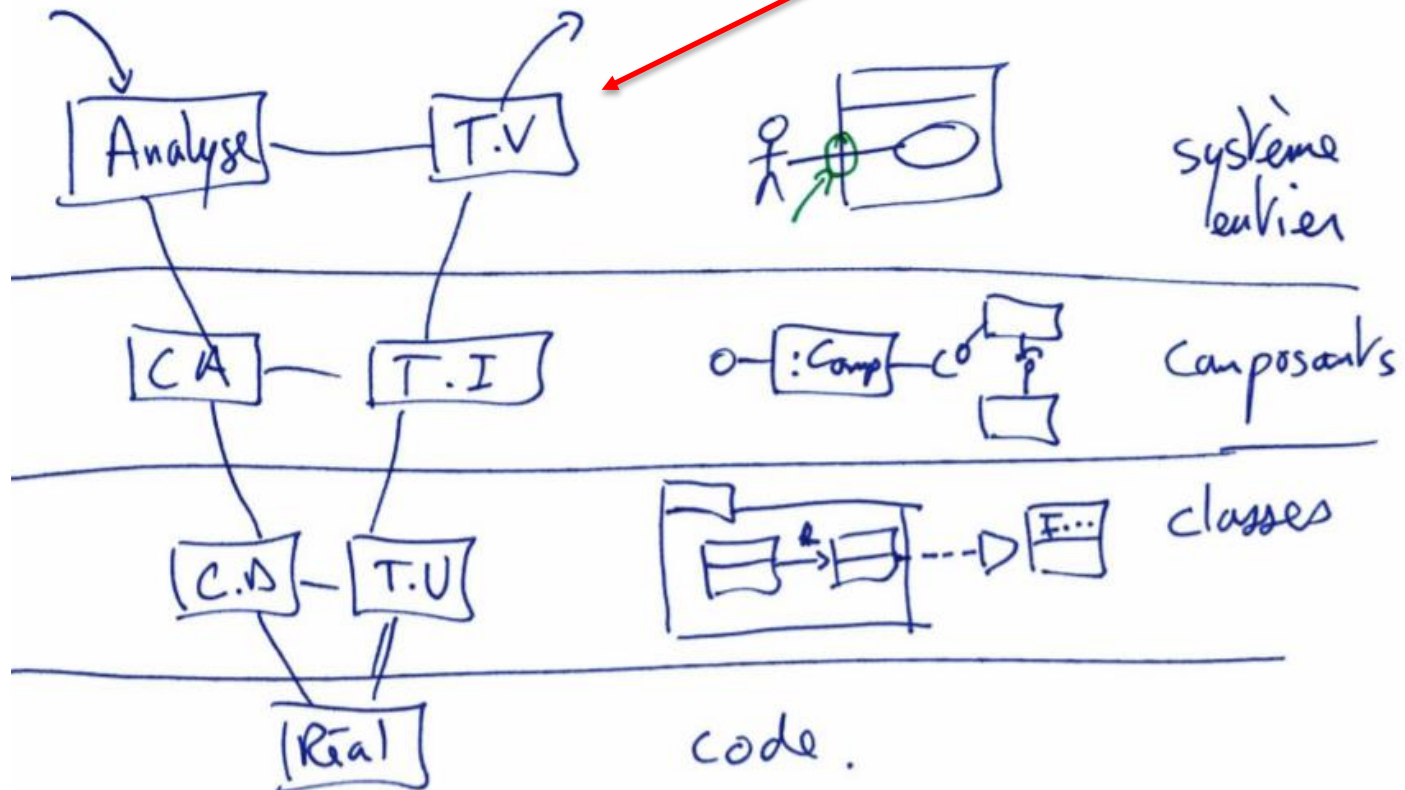
# Rappel

Analyse : trois artefacts introduits :

Structure : Classes métier

Dynamique : Use Case, Fiches Détaillées

Scope : Acteur vs Système



# Branche Contrôle

---

Le travail réalisé correspond à la branche  
*élaboration*

Le cycle en V prévoit un contrôle de chaque étape de production

Comment contrôler

1. Que le besoin est correctement capturé/compris ?

Adéquation des documents d'analyse par rapport au cahier des charges

2. Que le système final satisfait sa spécification ?

Conformité du produit final aux documents issus de l'analyse

---

# Test : Définition

---

# Validation par les tests

Principale approche industrielle pour la validation : Tester

Cette  
exécution sur  
ces données  
est correcte



AABABA



TFTF  
TT

oracle

entrées



sorties



System Under Test

++ Confiance dans le système

-- Pas de **garantie** que le système est correct

# Notions du domaine métier : *Test*

---

- Système à tester
  - Boîte noire dont on souhaite valider les comportements
  - On est à la frontière : Entrées + Sorties **observables**
- Séquence de test
  - Scenario de stimulation du système sur ses entrées
  - Mot dans l'alphabet du système
- Exécution d'un test
  - L'exécution d'une séquence de test sur un système produit des *sorties observables*
- Oracle
  - Chargé de décider si ces sorties correspondent au **résultat attendu**  
Dans les cas simples, assertEquals(attendu,obtenu)  
Si le système n'est pas déterministe un peu plus compliqué  
Entrelacements ? Générateurs aléatoires ?
- Verdict
  - PASS, FAIL, (INCONCLUSIVE, TIMEOUT)

# Tests : complétude

---

- Tester exhaustivement une fonction `int add (int a, int b)` sur une machine à 1GHz.
  - Combien de tests faut-il réaliser ?
  - Combien de temps cela représente ?
- Tester exhaustivement une fonction :
  - String concat (String prefix, String suffix)
  - Domaines a priori infinis !
- Cas général dans les tests : on ne peut pas être complet



# Tests : Suite de test

- Un test est toujours une séquence d'interactions sur le sujet
  - Tester une liste : `size=0`, `add(x)`, `size=1`
  - Le sujet du test est en général muni d'un état interne
    - A notre stade = classes métier du système
  - ✓ Sa façon de réagir au stimuli dépend de cet état interne
    - Pour des tests **reproductibles** nécessité de contrôler cet état interne
    - EX : Créer compte => on peut maintenant se logger avec ces identifiants
- Le système *accepte* un certain langage
  - Ensemble de mots dans l'alphabet de ses entrées menant à une sortie attendue
    - Pour un système OO, séquences de méthodes
    - En analyse, séquence de clics
  - Un test = valider qu'un certain mot est accepté
  - Test d'erreur
    - Valider que certains mots ne sont **pas** acceptés
    - EX: se logger avec le mauvais mot de passe

Objectif : capturer au mieux le langage du système à travers des *traces caractéristiques*

# Les tests : bilan

---

- On teste à la frontière du système
  - Le système est une boîte noire
    - D'autres approches existent greybox, whitebox, concolic testing...
- Un test valide une séquence particulière sur des données particulières dans un état particulier du système
  - L'utilisateur saisit un login en UTF-16 ??
  - Augmente la confiance dans le système
    - Mais pas de garanties de correction
    - Des approches existent pour construire des systèmes prouvés corrects : vérification formelle

---

# Tests de Validation

---

# Test de *Validation*

---

- Objectif :
  - Validation par le client
    - Communication du client vers l'équipe, aspect contractuel
    - Elaboré conjointement avec le client, valide les scenarios des fiches
  - ✓ Validation de la conformité
    - Applicable dans la procedure de *recette*
- Nature :
  - Ce sont des *tests*
    - Suite d'actions de l'acteur sur le système => une réponse attendue
  - ✓ Langage naturel structuré
    - *Reproductible* par le testeur (un humain) sur le système en tant qu'utilisateur "normal"
  - ✓ Accent sur la reproductibilité
    - Valeurs précises assurant la reproductibilité du test

# Rubriques

---

- Identifiant, Titre
  - + cross références au use case visé
  - TV02 : Créer Section (UC01 nominal).
- Contexte :
  - Dans quel état se trouve le système au début du test ?
  - Soit comment l'amener dans cet état
    - À travers d'autres interactions ou tests qui doivent précéder celui-ci
    - Ce test doit suivre TV01 : Créer Livre UC02 (nominal)
  - ✓ Soit en précisant l'état des données métier,
    - existence de certaines instances de classes métier
    - Le **livre** actuellement ouvert est vide

# Rubriques (2)

---

- Entrée :
  - Citer **toutes les données** que l'utilisateur a besoin de fournir au cours du test
    - Le testeur ne doit **rien** inventer
  - Toutes les données à saisir dans des formulaires
    - Le login « root », Le mot de passe « soleil ».
  - Possiblement aussi des fichiers fournis avec le cahier de tests
    - Fichier Grille1.sdk à charger fourni avec l'application

# Rubriques (3)

---

- Scenario
  - Séquences d'action de l'utilisateur sur le système
    - 1. Sélectionner « se logger »
    - 2. Saisir le login « root » et le mot de passe « soleil »
    - 3. Valider
  - ✓ Pas de variation/alternative etc... c'est un scenario
    - En boite noire sur le système, on clique...
  - ✓ Pas d'étapes du système avant le résultat attendu !
    - Ce n'est pas un scenario de fiche détaillée
  - ✓ Précision et reproductibilité des étapes
    - Les données saisies sont citées dans l'entrée

# Rubriques (4)

---

- Résultat Attendu (R.A.) :
  - Comment le système est-il modifié par cette série d'action ?
  - **ESSENTIEL** pas de test sans résultat attendu
    - Si le résultat correspond => test PASS, sinon test FAIL
  - Modifications (post-conditions) sur les données métier
    - L'utilisateur est maintenant logé
    - Un nouvel Etudiant a été ajouté au groupe
  - Affichages d'informations
    - Préciser quelle information exactement : screenshots, valeurs à lire/contrôler...



# Rubriques (5)

---

- Moyen de Vérification (M.V.)
    - De quelle manière peut-on contrôler que le R.A est atteint
      - Affichages :
        - Visuellement
        - en comparant aux valeurs de contrôle
      - Modifications de données métier :
        - Observables via d'autres tests
- TV03 afficher liste étudiant affiche l'étudiant créé

# Test de Validation

---

- Rubriques :
  - Titre, ID,
  - Contexte,
  - Entrée,
  - Scenario,
  - Résultat Attendu,
  - Moyen de Vérification
- Lisible par le client
- Reproductible
- Tests nombreux avec différentes valeurs etc...
- *Suites* de tests nécessaires pour amener l'application dans un état depuis son état initial
  - On livre souvent avec un jeu d'essai utilisé pour ces tests

# TV : Position dans le cycle

---

- Elaboration des tests
  - En analyse, conjointement avec FD, UC, CM
  - Construction du Cahier de Tests de Validation
  - Validation par le client du jeu de test **avant** d'entamer la conception
    - **Contractuel** : Si je livres un produit qui passe ces tests, mission accomplie !
- ✓ Exécution des tests au cours de la *recette*
  - On prend une machine nue, on installe le logiciel, on exécute les TV assis devant la machine
  - Un Test => PASS ou FAIL
  - Tous les tests doivent être validés

# TV05 : Acheter une carte (UC02 SN)

---

- Contexte : on est connecté sur "testeur", le compte dispose de fonds suffisants en joyaux (au moins 20 joyaux).

- Entrée : la carte commune "Dragon vert"

- Scenario :

1. sélection de l'option acheter carte
2. saisie de dragon vert dans la boîte de recherche
3. sélection du dragon vert
4. valider

R.A. :

- la carte « dragon vert » est ajoutée à la collection,
- la quantité de joyaux du compte est diminuée de 20.

MV:

- dans la construction de deck, s'assurer que la carte est disponible.
- La quantité de joyaux affichée en permanence dans le coin en bas à droite est passée à 5 joyaux.

# Cahier de tests

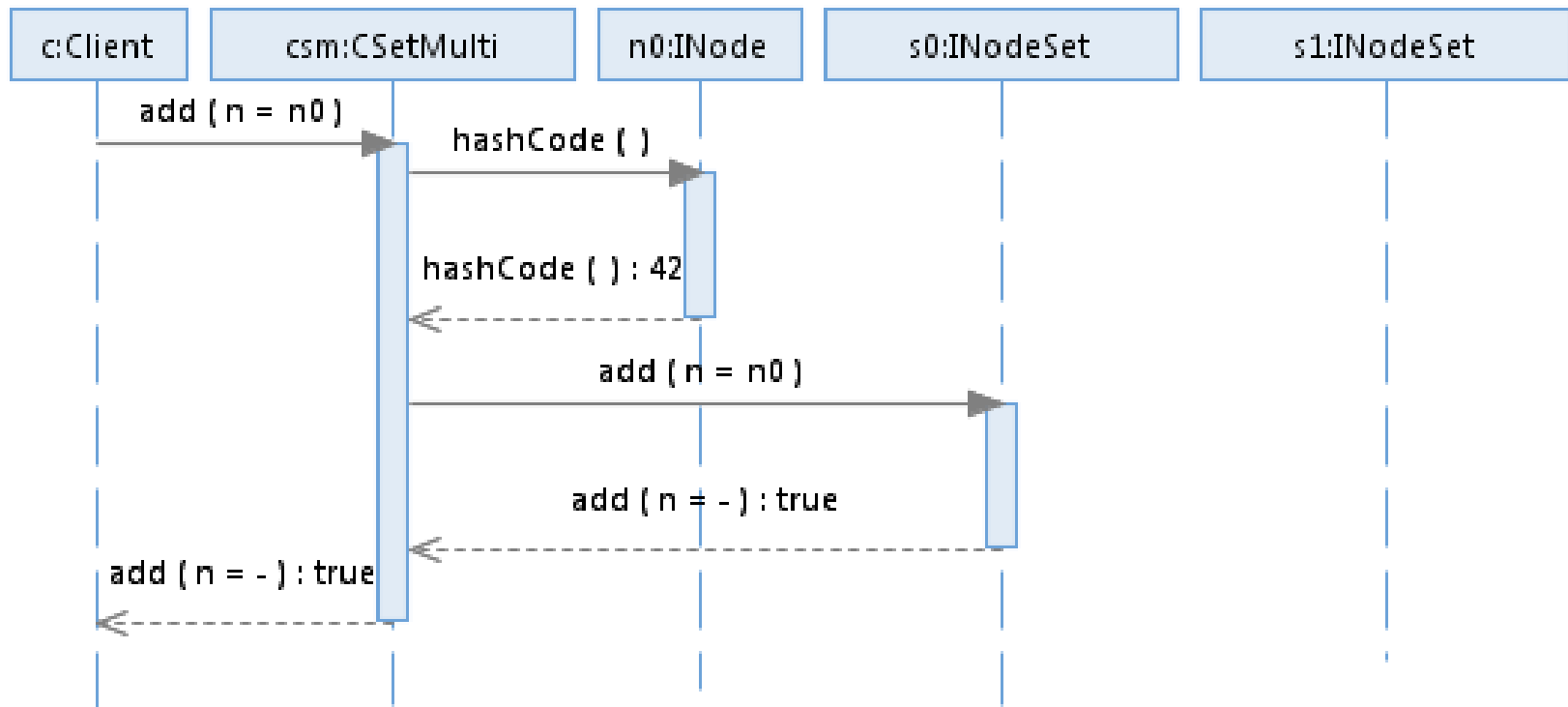
---

- Idéalement chaque *scenario* SN, ALT, EX des fiches détaillées est couvert par plusieurs tests avec différents jeux de données
  - Le plus complet possible
  - Exhibe les divers comportements possibles du système
  - On voit comment le système réagit à diverses fautes possibles (EX, ALT)
  - A accompagner par des maquettes d'IHM pour mieux comprendre les scenarios
- On ne sera jamais exhaustif
  - Se satisfaire à un certain niveau de ce qu'on a testé
  - Métriques de qualité : couverture vis-à-vis des use case, de leurs scenarios

# Séquences d'Analyse

# Diagrammes de séquence

- Lifeline :
  - instance d'un objet
- Message :
  - Instance d'invocation (ou retour) d'une méthode d'un objet
- Barre de *focus* : thread/comportement actif à cet endroit



# Cohérence Séquence vs Classe

---

- Classes Grand-père, Horloge
  - Qui possède l'opération « LireHeure():Date » ?
  - Le récepteur *offre* un service à son client (l'invocateur)
- Les méthodes invoquées dans un message sont celles de la classe du récepteur
  - Un message a en plus des *valeurs* explicites pour les arguments
    - Types simples : age=18, nom=« toto »
    - Noms des autres lifelines du diagramme pour passer des références
- Une opération offerte = une charge de développement
  - Mais précise, *signature* des données échangées



# Cohérence Séquence vs code

---

- On ne voit que les invocations *entre* objets
  - Les mise à jour des attributs sont éliminées
  - Toute invocation à un objet peut être rendue visible
    - Granularité du diagramme
- Le code = un ensemble de comportements
  - Une séquence = une seule trace possible
    - Branche if ou branche else, pas les deux
    - Un tour de boucle = une invocation, deux tours = deux, ... la boucle est suggérée mais jamais complètement décrite.
  - ✓ Etant donné une situation particulière des données R.A. déterministe le plus souvent
    - Facile à tester ou exhiber sur une séquence

# Diagrammes de Séquence d'Analyse

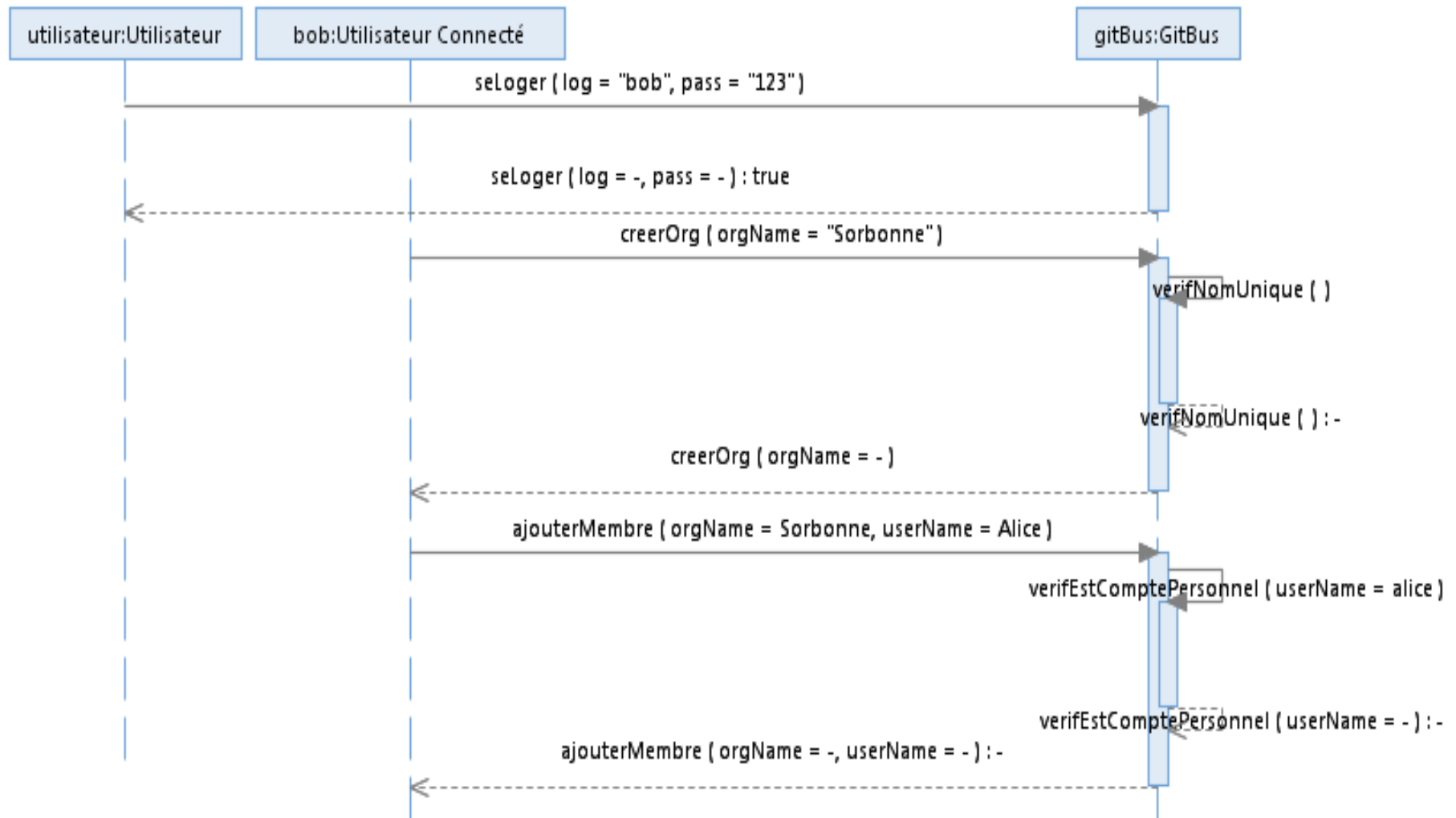
- Objectif :
  - Préparer la *transition* vers la conception
  - Identifier les responsabilités du système et les énumérer *techniquement*
  - Chaque opération identifiée = une charge de développement
- Nature :
  - Diagrammes de séquence très simples
    - Un ou plusieurs acteurs vs. Le système
    - Le système est le seul à offrir des opérations
      - On ne va pas *développer* les acteurs !
  - ✓ Des traces d'exécution du système
    - Élaboré conjointement avec les T.V.
    - Réutilisation des données des T.V.

Déduire des opérations portées par une classe fictive *Système*

# Cohérence Fiches détaillées/Scenarios

- On recherche les responsabilités du système
  - Affichages, calculs, vérification, enregistrements
- Gros accent sur ce qui circule à la frontière
  - Le résultat des opérations de saisie utilisateur (bouton OK)
    - Un ensemble d'informations structurées = signature
    - Signatures figée par les workflows des utilisateurs
    - Une responsabilité « public », visible des acteurs
      - Peu remise en cause en conception
    - Fusion de l'acteur et de son IHM
      - On démarre quand le *marshalling* des arguments est fait
  - ✓ Les autres opérations (calcul, affichage, enregistrement) sont identifiées
    - Mais signature peu figée, possible à remettre en cause etc...
    - On les note « private », invoquées par le système sur lui-même
    - Ne pas en abuser (granularité)

# Exemple



# Exemple :classe système induite

## GitBus

- + seLogger ( log : String, pass : String ) : Boolean
- + creerOrg ( orgName : String )
- + ajouterMembre ( orgName : String, userName : String ) : Boolean
- + ajouterAdmin ( orgName : String, userName : String ) : Boolean
- verifNomUnique ( ) : Boolean
- verifEstComptePersonnel ( userName : String ) : Boolean

# Bilan Analyse

# Bilan : Analyse

---

- Scope : Acteur vs Système
- Quoi ? Sans préciser Comment ?
  - Responsabilités du système dérivées du besoin
- Pas d'éléments techniques, modélisation *métier*
- Utilisation d'une partie de la notation UML d'une manière précise
  - Méthode de développement !
  - Artefacts intermédiaires du processus
- Importance pratique de l'analyse
  - Contractuel, traçabilité des *requirements*
  - *Anticipe* le résultat des phases de conception et dev

# Analyse : le besoin ?

---

- Développement continuellement tiré par le besoin
  - Découpe fonctionnelle de l'application = vis-à-vis des use case
    - Différent de la vue composant
- Application à la conduite de projet
  - Construire de façon *itérative* un prototype
  - Amener une par une dans le prototype des fonctionnalités *démontrables* (feature)
  - Découpage basé sur les scenarios des use case
- Approches agiles
  - Construction du backlog produit : features souhaités + priorité client
  - Découpage en réalisation :
    - Tâches induites par chaque feature
    - Pricing difficulté de dév en Fibonacci
    - Choix des tâches = Backlog itération
  - Itération gérée en Kanban
    - S'achève sur une démonstration d'un prototype fonctionnel



# Spécification Technique Du Besoin

---

- Maquette possible d'une STB :
  - Titre, Date, Versions, Auteurs, Changelog, droit d'accès, copyright,...
  - Table des matières
  - Glossaire des termes
  - Description générale (1 à 2 pages) : mission du système
  - Acteurs : sens métier des acteurs, rôle vis-à-vis du système
  - Use case : diagramme d'ensemble + commentaires
  - Fiches détaillées : une par use case
    - + maquettes IHM potentiellement
  - Volumétrie, contraintes non fonctionnelles...
  - Classes métier + Dictionnaire des données
    - Pour chaque entité/attribut, donner son sens métier
- ✓ Matrice de traçabilité

# Prêts pour le partiel !

On a maintenant les 5 artefacts demandés en examen réparti 1. S'entraîner sur les annales.