

Eléments de Génie Logiciel

Cours 1-3



Pr. Jean-Marc Jézéquel
 IRISA – Université de Rennes
 Campus de Beaulieu
 F-35042 Rennes Cedex
 e-mail : jezequel@irisa.fr
<http://people.irisa.fr/Jean-Marc.Jezequel>
 X @jmjezequel

1 - Introduction

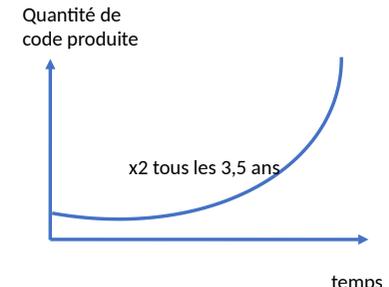
Software is eating the world (Marc Andreessen)



- Le logiciel forme le tissu de notre société
 - Tel l'oxygène qu'on respire, force motrice essentielle, bien qu'invisible, du monde actuel
 - pratiquement aucun aspect de la société qui ne soit pas facilité ou médiatisé par le logiciel
- *Toute (grande) entreprise est désormais une entreprise de logiciels (S. Nadela)*
 - Le logiciel est une technologie clé de transformation et d'habilitation qui impacte de nombreux domaines de l'économie moderne
- Les logiciels stimulent le progrès scientifique dans de nombreux domaines
 - IA, la science des données, la médecine, l'ingénierie, etc.
- Les logiciels sont essentiels pour nos infrastructures critiques
 - énergie, les télécommunications, l'aérospatiale, l'automobile, la finance, la santé
- Le volume total de logiciels dans le monde croît à un rythme exponentiel
 - Double tous les 3,5 ans

Taille de quelques logiciels (lignes de code)

- | | |
|----------------------------|-------------------------------|
| • MS-DOS: 4 k | • Google Chrome: 6.7 M |
| • WhatsApp: 30 k | • Twitter: 10 M |
| • Telegram: 50 k | • Android: 12 M |
| • Zoom: 60 k | • iOS: 12 M |
| • TikTok: 80 k | • Mozilla Firefox: 21 M |
| • Space Shuttle: 400 k | • Windows XP: 45 M |
| • Minecraft: 500 k | • Large Hadron Collider: 50 M |
| • Instagram: 1 M | • Ubuntu: 50 M |
| • US Military Drone: 3.5 M | • Facebook: 62 M |
| • YouTube: 5.4 M | • MacOS X: 84 M |
| • World of Warcraft: 5.5 M | • Tesla: 100 M |
| • Boeing 787: 6.5 M | • Google: 2 milliards |



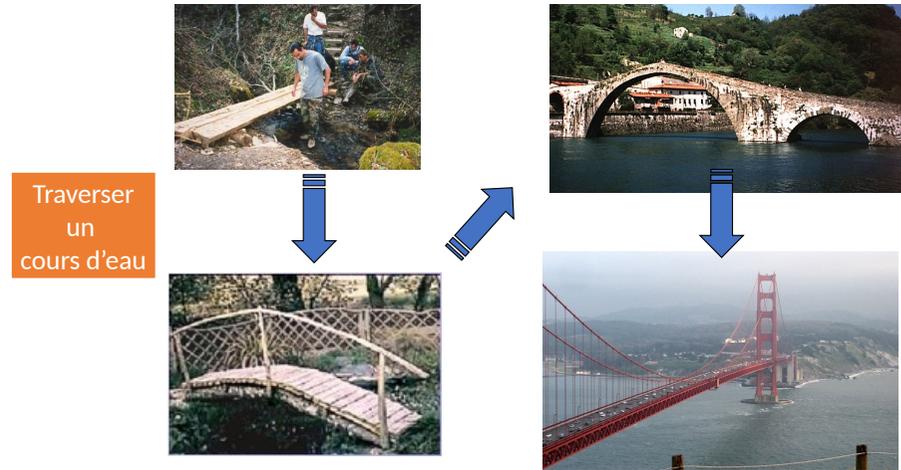
Ce n'est pas demain que les (bons) développeurs seront au chômage!

Même avec Copilot/ChatGPT...

Des logiciels souvent complexes...

- Logiciels de grande taille
 - des millions de lignes de code
 - des équipes nombreuses
 - durée de vie importante
 - des lignes de produits
 - plateformes technologiques complexes
 - évolution continue
 - Problèmes de fiabilité, sécurité, efficacité, utilisabilité, etc.
- Besoin d'ingénierie pour les réaliser
 - Génie logiciel

Ingénierie : Gérer la complexité due à la taille



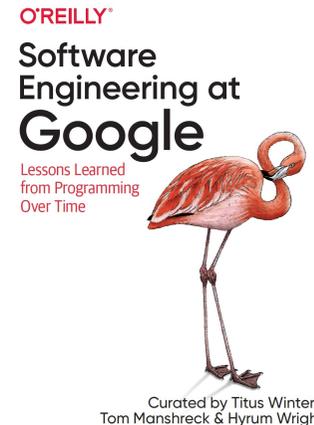
Qu'est-ce que le génie logiciel ? (Wikipédia)

- **Procédures systématiques** qui permettent d'arriver à ce que des **logiciels de grande taille**
 - correspondent aux **attentes du client**,
 - soient **fiables**,
 - aient un **coût d'entretien réduit**
 - et de bonnes **performances**
 - tout en respectant les **délais** et les **coûts de construction**
- Qui utilise ces **procédures systématiques** du génie logiciel ?
 - **Tous les développeurs** de logiciels (en entreprise ou dans le logiciel libre)

Qu'est-ce que le génie logiciel ?

- La définition de Google :
 - *programming integrated over time*

The Google codebase includes approximately one billion files and has a history of approximately 35 million commits spanning Google's entire existence. The repository contains 86TB³ of data, including approximately two billion lines of code in nine million unique source files.



D'où je vous parle...

- Professeur Université de Rennes
 - Membre senior de l'[Institut Universitaire de France \(IUF\)](#)
 - Président de [Informatics Europe](#)
 - Directeur de l'IRISA (2012-2020)
- Chercheur en Sciences du Logiciel
 - Médaille d'Argent du CNRS
 - Principes, méthodes et outils utilisés dans le développement de logiciels
 - Visé à mieux comprendre et améliorer les processus de fabrication : qualité, productivité
 - Nombreuses collaborations industrielles
 - avec Airbus, Thalès, Orange, PME et start-ups.



https://fr.wikipedia.org/wiki/Jean-Marc_Jézéquel

2 - Présentation du module *Eléments de Génie Logiciel (GEN)*

Page du cours :

<http://people.irisa.fr/Jean-Marc.Jezequel/enseignement/L2GEN>

Qu'allez vous faire pendant cette UE

- ?
- Vous ne pourrez plus dire :
 - ce qu'on voit en cours ne nous servira jamais
 - les cours ne parlent pas de choses utilisées dans la "vraie vie"
 - Cette UE va vous confronter à de vrais problèmes de développement
 - travailler en équipe *(seul on va plus vite, en équipe on va plus loin)*
 - communiquer !!
 - organiser l'équipe
 - organiser son travail dans l'équipe
 - partager son code
 - tester son code
 - rendre son code lisible, utilisable et modifiable par d'autres
 - documenter son code
 - Modifier le code des autres
 - Modifier son code
 - ... trouver des solutions algorithmiques, programmer

Importance décroissante



Objectifs généraux du cours

- Confrontation à la réalité du développement collaboratif de logiciels de grande taille, en présentant quelques principes et outils pour parvenir à le maîtriser.
- À l'issue de ce cours, les étudiants sont conscients de problèmes posés par :
 - Le développement de logiciels de taille importante
 - Organisés en modules/classes/interfaces (Architecture logicielle)
 - Le développement en équipe (de ~ 10 personnes)
 - La maintenance évolutive du logiciel
 - Test logiciel (test unitaire, test d'intégration)
 - Gestion de version
 - Méthodes agiles
- Ils ont renforcé leur compréhension de
 - La programmation objet
 - L'utilisation d'IDE de type industriel (VSCode)
 - L'utilisation de LLMs pour la programmation (ChatGPT, Copilot...)

Ce n'est pas un cours de Java, même si Java est le langage qui sera utilisé en TP

En PO vous avez appris à coder l'intérieur des classes. En GEN on va mettre le focus sur les relations entre les classes aka *Architecture Logicielle*

Pourquoi enseigner le génie logiciel dès la L2?

- Parce que c'est nécessaire dans la formation de tout développeur
- Pour que vous puissiez, dès maintenant, vous impliquer dans des projets de développement collaboratifs (par ex. logiciel libre)
- Pour que vous soyez plus rapidement autonomes en stage
- Pour que vous soyez confronté au problème suffisamment tôt
- Permet de comprendre l'importance de beaucoup de cours de L3/M1
- Parce que c'est nécessaire au bon déroulement des projets de L3/M1
- Parce que c'est un prérequis de nombreux cours de L3/M1/M2

Programme des CM

- Concepts de base des langages à objets
 - objets, classes, encapsulation, masquage d'information, interface, modularité
- Concepts fondamentaux des langages à objets
 - héritage, polymorphisme, typage
- Introduction à l'architecture logicielle
 - diagrammes de classes, associations, diagrammes de séquence
- Les 3 dimensions de la gestion de configuration
 - variants, versions, et concurrence, principes de Git
- Introduction au test de logiciel
 - test unitaires, JUnit
- Outils pour la qualité du logiciel
 - documentation, typage statique, programmation par contrats, qualité des tests
- Introduction au développement agile
 - processus de développement, principes de l'agile, Use cases/user stories
 - utilisation de LLMs type ChatGPT ou Copilot

Et les Travaux dirigés ?

- Pas de Travaux dirigés !
 - 8 CMs et 20 TPs + 4 TPs non encadrés
 - Vous pouvez poser des questions pendant les cours (ou à la fin)
 - Vous devez lire la documentation mise à votre disposition
 - **Vous devez poser des questions en TP**
- Des quiz notés (sans documents) en CM
 - La mauvaise nouvelle : vous devez connaître et comprendre le cours
 - La bonne nouvelle : le cours est réduit ! (8 CMs)

Travaux pratiques

- Conception de l'architecture logicielle du projet
 - Développement collaboratif du projet
 - Génération de la documentation
 - Test systématique et automatique du logiciel
 - Validation par des tiers (enseignants, tests automatisés)
- Outils utilisés en TP :
 - VS Code (avec plugins Java)
 - Java
 - JUnit 4
 - Git
 - LLMs type ChatGPT / Copilot
- Si vous n'avez pas fini un TP pendant le temps imparti, vous **devez** le terminer sur votre temps de travail personnel (sauf indication contraire)
 - Les TP ne sont pas « faciles » et demandent un **vrai travail**

Evaluation : Contrôle Continu intégral

• Modalités du CC

- Durant le semestre, plusieurs quiz de 10 min sans documents
 - en amphi au début ou à la fin de certains CM
 - Absence au CM => 0 au CC (sauf ABJ : note neutralisée)
 - Les PAEH peuvent me demander à être dispensés (par e-mail)
- Durant le semestre, deux notes de travaux pratiques avec pour chaque :
 - Une note d'équipe
 - Une note individualisée après entretien individuel avec le ou la chargé de TP
- À la fin du semestre 4, un CC final d'1h30, tous documents autorisés.

(1/5 de la note)

(2/5 de la note)

(2/5 de la note)



Je vous encourage et je m'attends à ce que vous **utilisiez** l'IA (ChatGPT et/ou Copilot) dans ce cours. *Apprendre à utiliser l'IA est une compétence émergente et il est important de savoir l'utiliser en génie logiciel*

Soyez toutefois conscient des **limites** de ChatGPT/Copilot/... : <https://youtu.be/R2fjRbc9Sa0>

- Si vous fournissez des questions (*prompts*) en fournissant peu d'effort, vous obtiendrez des résultats de faible qualité. Affinez vos *prompts* afin d'obtenir de bons résultats. **Cela demande du travail.**
- Ne vous fiez jamais à ce qu'il dit. **Gardez un esprit critique. Vous serez responsable de toute erreur** ou omission fournie par l'outil. Il vaut mieux utiliser l'IA sur des sujets que vous comprenez.
- L'IA est un outil que vous **devez reconnaître avoir utilisé**. Veuillez utiliser les commentaires de votre code pour expliquer à quelles fins vous avez utilisé l'IA et quels prompts vous avez utilisés pour obtenir les résultats. Ne pas le faire constitue une violation des politiques d'honnêteté académique.
- Vous devez **comprendre** le code que propose l'IA. Vous pouvez d'ailleurs lui demander de vous l'expliquer. Lors de votre évaluation individuelle **tout code que vous n'êtes pas capable d'expliquer entrainera une note de 0.**
- Réfléchissez au moment où cet outil est utile. Ne l'utilisez pas s'il n'est pas approprié au contexte du travail à fournir, **au minimum ayez un usage raisonné en lien avec l'impact environnemental associé.**

Questions?

3 – Concepts de base des langages à objets

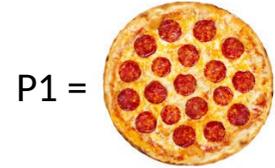
Objets et Classes

“Objet” (Définition)

- Formellement : la fermeture transitive d’une fonction
- Concrètement : encapsulation d’un état avec un ensemble d’opérations travaillant sur cet état
 - abstraction d’une entité du monde réel
 - existence temporelle :
 - création, évolution, destruction
 - identité propre à chaque objet
 - peut être vu comme une machine
 - ayant une mémoire privée et une unité de traitement,
 - et rendant un ensemble de services

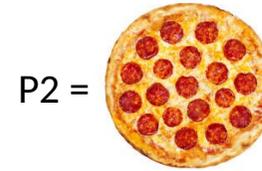


Objet : identité vs. égalité



P2 = P1

=> P2 == P1 // *identité*



=> P2 != P1 // *pas identité*

=> P2.equal(P1) == true // *Egalité*

“Classe” : Définition en temps que type

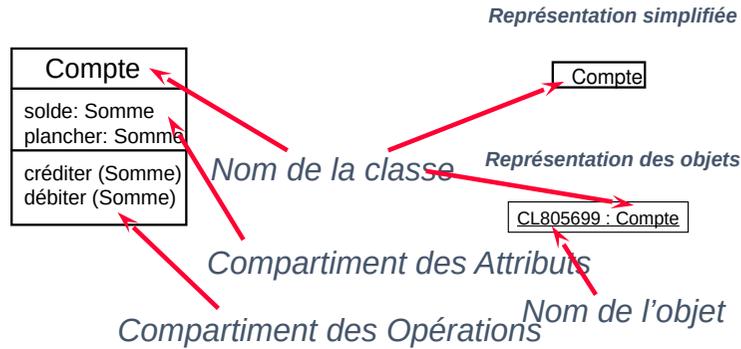
- Description des propriétés et des comportements communs à un ensemble d’objets
 - Implantation d’un type de donné abstrait
 - un objet est une instance d’une classe (eq. variable vs. type)
- Chaque classe a un nom, et un corps qui défini :
 - les attributs possédés par ses instances
 - les opérations définies sur ses instances,
 - et permettant d’accéder, de manipuler, et de modifier leurs attributs
 - une classe peut elle même être un objet (*reflexivité*)
 - Par ex. en Java : `Class<?> maClasse = monObjet.getClass();`
 - Et alors `maClasse.getClass() ???`

Représentation des Objets et des Classes

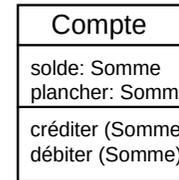
- UML : Langage de modélisation pour le logiciel (*Unified Modeling Language*)
 - Réfléchir
 - Définir la structure « gros grain » : Architecture Logicielle
 - Documenter
 - Guider le développement
 - Développer, Tester, Auditer
- UML permet de modéliser de nombreuses autres choses...
 - Mais ca, c’est pour la L3



• Représentation d'une classe



Correspondance UML Java



```
class Compte {
    public Compte(int plancher) {...}
    private int solde;
    private int plancher;
    public int getSolde(){
        return solde;
    }
    public void credited(int montant) {
    }
    public void debiter(int montant) {
    }
}
```

- Conventions de nommage :
- Le nom d'une **classe** est un **nom**
 - Les **attributs** sont des **noms**
 - Les **méthodes** sont des **verbes**
 - (à l'infinitif)

Quelques langages « orientés objets »

(par ordre de popularité selon [IEEE Spectrum](#))

- **Python** (Académique)
 - -> scripts, ML
- **Java** (Oracle)
 - -> Serveurs etc.
- **C++** (Bell Labs)
 - -> infra/telecom/images
- **C#** (Microsoft)
 - -> .NET
- **Swift** (Apple)
 - -> IOS
- **Dart** (Google)
 - -> IOS + Android (Flutter)
- **Kotlin** (JetBrains+ Google)
 - -> Android

Déjà utilisé  jamais utilisé 

+ Quelques langages OO plus ésotériques:

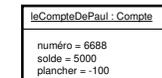
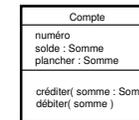
- JavaScript/TypeScript
- Objective-C
- Go
- Scala
- Eiffel
- Julia

A distinguer de langages comme:

- SQL, HTML
- C, Fortran, Cobol
- Haskell, Rust, OCamel, LISP...

Vous devriez être capable de passer d'un langage orienté objet à l'autre avec un **minimum d'efforts**

Instanciation



- Java / C# / C++
 - `Compte leCompteDePaul = new Compte(-100);`
- Python
 - `leCompteDePaul = Compte(-100)`
- Kotlin
 - `leCompteDePaul = Compte(-100)`
- Dart
 - `var leCompteDePaul = Compte(-100)`
- Swift
 - `let leCompteDePaul = Compte(plancher:-100)`

```
class Compte:
    def __init__(self, plancher):
        self.plancher = plancher
```

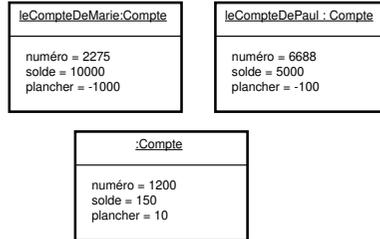
```
class Compte(val plancher: Int)
```

```
class Compte { int plancher;
    Compte(this.plancher);}
```

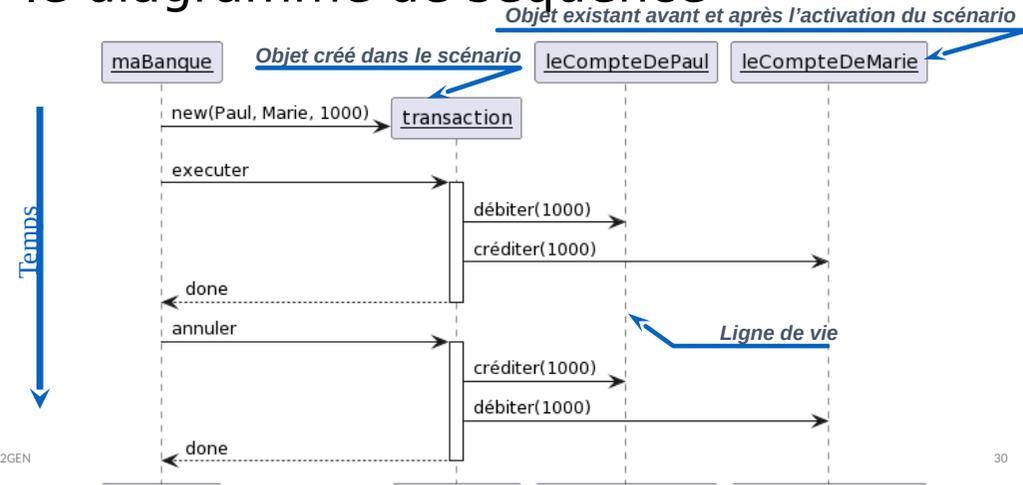
```
class Compte {
    var plancher: Int
    init(plancher : Int) {
        self.plancher = plancher
    }
}
```

Diagramme d'objets

- Des **objets** peuvent être ajoutés ou détruits pendant l'exécution
- La valeur des attributs des objets peut changer



Exemple de diagramme d'objets : le diagramme de séquence



Classe vs. Objets

Une **classe** spécifie la structure et le comportement d'un ensemble d'objets de même nature

- La structure d'une classe est constante

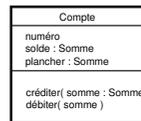


Diagramme de classes

M1

M0

- Des **objets** peuvent être ajoutés ou détruits pendant l'exécution
- La valeur des attributs des objets peut changer

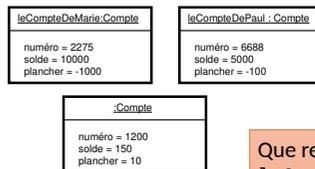


Diagramme d'objets

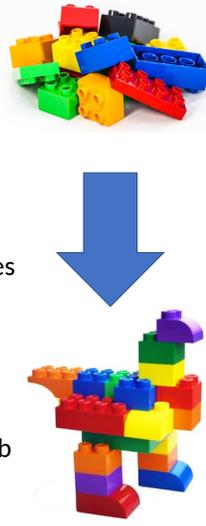
Que retourne : `leCompteDeMarie.getClass()`?

4 - Modularité

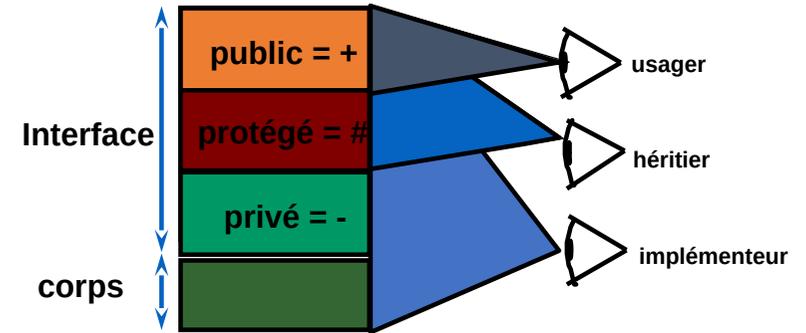
La "Classe" en temps que module

Principe de l'approche objet

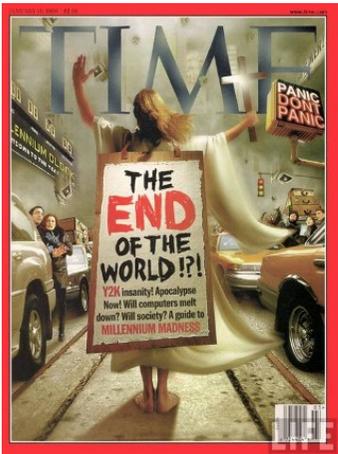
- Structurer les systèmes autour des objets
 - Plutôt qu'autour des fonctions
- Obtenir des systèmes modulaires et maintenables
 - Par assemblage de briques de base
 - Composition d'éléments simples pour obtenir des systèmes sophistiqués
- Favorise :
 - masquage d'information (abstraction)
 - encapsulation (facilite les modifications à portée locale)
 - dissociation interface/implantation=> composant réutilisab



- Différentes visibilités des membres d'une classe



Contre exemple de modularité : le bug de l'an 2000



Encapsulation/masquage d'information

```

date
year: String[2]

equal(date)
isLess(date)
delta(date)
    
```

- Un choix de format de stockage pour l'année
 - 3 opérations principales (et leurs dérivées) + print
- Correction bug an 2000?
 - modification du format de stockage
- **Sans encapsulation/masquage d'information:**
 - Cout de la modification **proportionnel** à la taille du programme
 - Car il faut passer sur toutes les lignes pour reporter l'éventuelle modification
- **Avec encapsulation/masquage d'information:**
 - Cout de la modification **indépendant** de la taille du programme
 - Le reste du code est protégé de mes choix

Notion d'API

« Application Programming Interface »

- **Interface de Programmation Applicative**
 - ensemble de classes, de méthodes, de fonctions et de constantes
 - qui sert de *façade* par laquelle un logiciel offre des services à d'autres logiciels
- Le programmeur n'a pas besoin de connaître les détails de la logique interne du logiciel qui se cache derrière la façade de son API
 - Seule l'API est réellement nécessaire pour utiliser le système
- Convention de nommage
 - Noms pour les classes (Compte)
 - Verbes infinitif pour action (créditer)
 - Noms pour attributs (solde)
- En anglais idem
 - + getter & setter

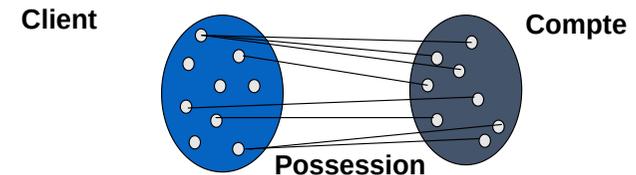
5 - Liens entre Objets

Et associations entre classes

Les objets ne sont pas définis isolément

- 2 catégories
 - Liens entre objets
 - Marie possède les comptes en banque n°12345 et n° 67890
 - Cette voiture est composée du châssis n° XX564 et des 4 roues AvG, AvD, ArG, ArD
 - => Associations entre classes
 - Une Personne possède un ou plusieurs Comptes en banque
 - Une Voiture est composée de 1 Châssis et de 4 Roues
 - Classification
 - Médor est un Chien, il est donc aussi un Mammifère
 - Médor appartient à l'ensemble des Chiens, qui est contenu dans l'ensemble des Mammifères
 - => Héritage entre classes
 - Un Chien est une sorte de Mammifère, qui est une sorte d'Animal

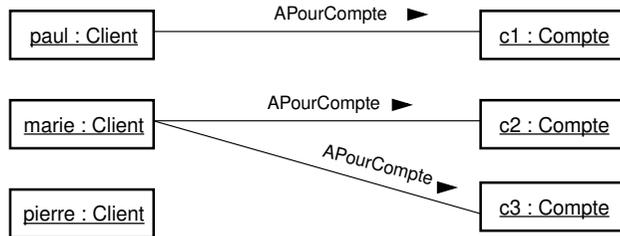
Vue ensembliste d'une relation : Graphe de la relation



Une relation met en correspondance des éléments d'ensembles

Liens (entre objets)

Un **lien** indique une connexion entre deux objets



Note de style :

- les noms des liens sont des formes verbales et commencent par une majuscule
- ▶ indique le sens de la lecture (ex: « paul APourCompte c1 »)

Associations (entre classes)

Une **association** décrit un ensemble de liens de même "sémantique"

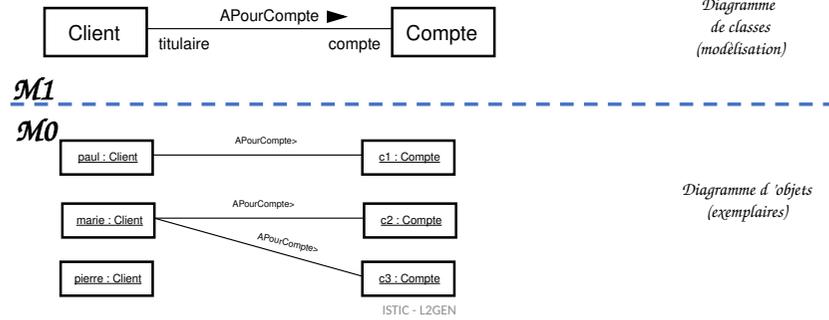
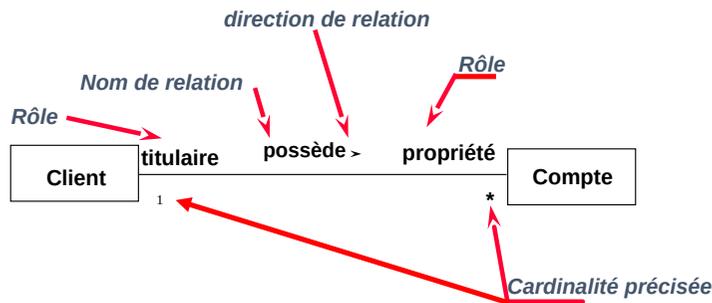


Diagramme de classes (modélisation)

Diagramme d'objets (exemplaires)

Représentation des associations : direction, rôle, cardinalité

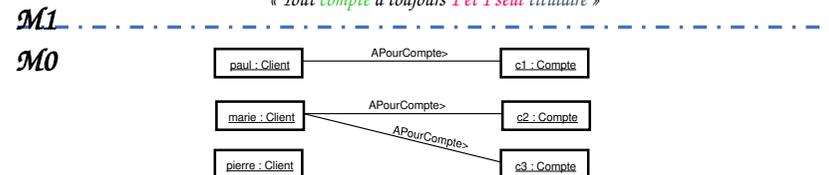


Cardinalités d'une association

- Précise combien d'objets peuvent être liés à un seul objet source
- Cardinalité minimale et cardinalité maximale ($C_{min}..C_{max}$)
- Doivent être des constantes

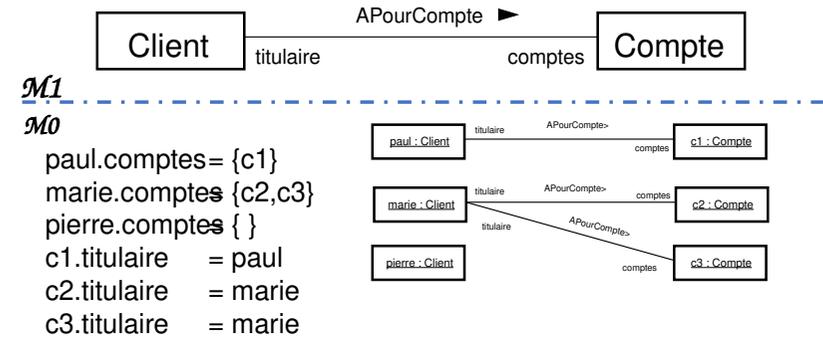


« Tout client a toujours 0 ou plusieurs comptes »
 « Tout compte a toujours 1 et 1 seul titulaire »



0,1	Compte	Optionnelle (0 ou 1)	En Java par exemple : Optional<Compte> c;
1	Compte	Exactement un	Compte c; //or @NotNull
3	Compte	Exactement 3	Compte c1, c2, c3;
*	Compte	Plusieurs, non ordonnés	Set<Compte> c;
{ordered} *	Compte	Plusieurs, ordonnés	List<Compte> c;
1..*	Compte	Au moins un	Set<Compte> c; // @NotEmpty
1-10	Compte	Intervalle	Compte[] c = new Compte[10];

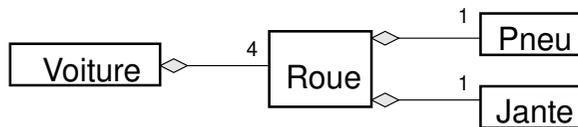
Utiliser les rôles pour « naviguer »



Nommer en priorité les rôles

Composition

Notion intuitive de "composants" et de "composites"

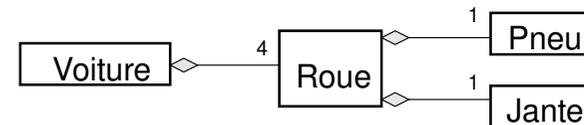


composition =
cas particulier d'association
+ contraintes décrivant la notion de "composant"...

Composition

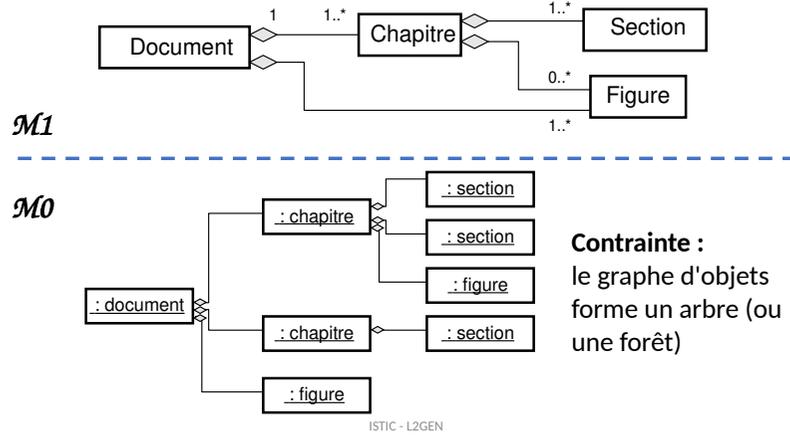
Contraintes liées à la composition :

1. Un objet *composant* ne peut être que dans 1 seul objet *composite*
2. Un objet *composant* n'existe pas sans son objet *composite*
3. Son objet composite est détruit, ses composants aussi



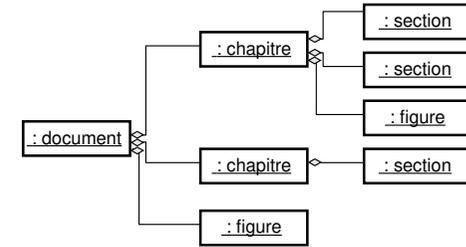
Dépend de la situation modélisée !
(Ex: vente de voitures vs. casse)

Composition



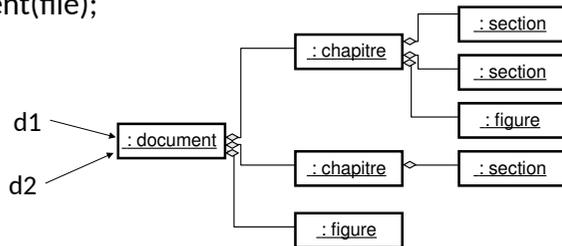
Copie d'objets composites

- Document d1 = new Document(file);



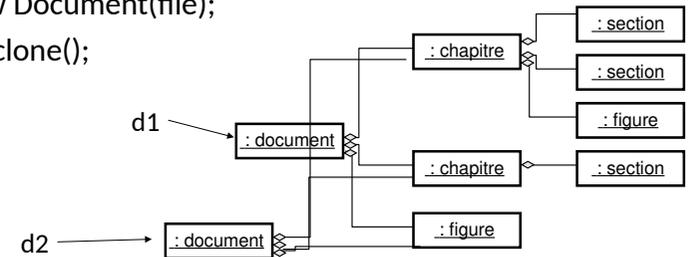
Affectation

- Document d1 = new Document(file);
- Document d2 = d1;



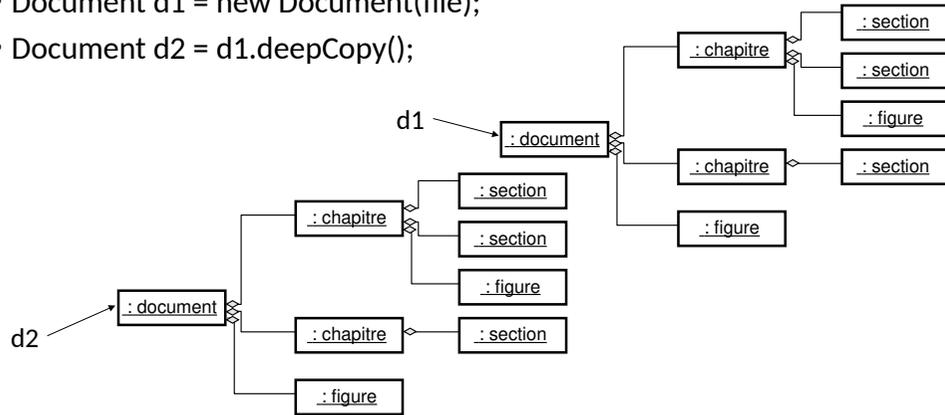
Copie simple

- Document d1 = new Document(file);
- Document d2 = d1.clone();



Copie profonde

- Document d1 = new Document(file);
- Document d2 = d1.deepCopy();



Copie d'objets complexes / Identité vs Egalité

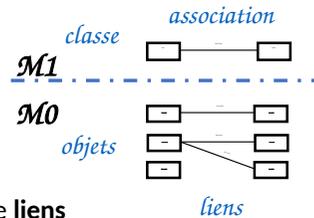
- | | | |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"> 1. Affectation <ul style="list-style-type: none"> • d1 = d2; 2. Copie simple <ul style="list-style-type: none"> • d1 = d2.clone(); 3. Copie profonde <ul style="list-style-type: none"> • d1 = d2.deepClone(); | | <ol style="list-style-type: none"> 1. Test d'identité <ul style="list-style-type: none"> • d1 == d2 2. Test d'égalité <ul style="list-style-type: none"> • d1.equals(d2); 3. Test d'égalité profonde <ul style="list-style-type: none"> • d1.deepEquals(d2); |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Support plus ou moins direct en fonction du langage

- Java : seulement 1 et ~2
- Python : 1, 2 (copy) et 3 (deepcopy)

Association vs. Liens

- Un **lien** lie deux **objets**
- Une **association** lie deux **classes**
- Un **lien** est une instance d'**association**
- Une **association** décrit un ensemble de **liens**
- Des **liens** peuvent être ajoutés ou détruits pendant l'exécution, (ce n'est pas le cas des associations)



6 - Classification

Héritage entre classes

Les objets ne sont pas définis isolément

- 2 catégories

- Liens entre objets

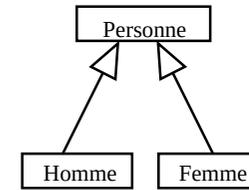
- Marie possède les comptes en banque n°12345 et n° 67890
- Cette voiture est composée du châssis n° XX564 et des 4 roues AvG, AvD, ArG, ArD
- => Associations entre classes
 - Une Personne possède un ou plusieurs Comptes en banque
 - Une Voiture est composée de 1 Châssis et de 4 Roues

- Classification

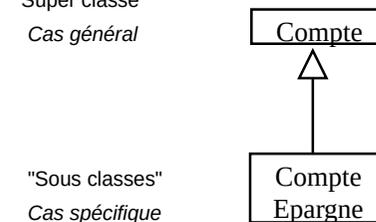
- Médor est un Chien, il est donc aussi un Mammifère
 - Médor appartient à l'ensemble des Chiens, qui est contenu dans l'ensemble des Mammifères
- => Héritage entre classes
 - Un Chien est une sorte de Mammifère, qui est une sorte d'Animal

Héritage et Polymorphisme : Généralisation / Spécialisation

Une classe peut être la généralisation d'une ou plusieurs autres classes. Ces classes sont alors des spécialisations de cette classe.



"Super classe"
Cas général



"Sous classes"
Cas spécifique

On dit que *CompteEpargne* hérite de *Compte*

Deux points de vue complémentaires

- Mécanisme de classification : sous-typage

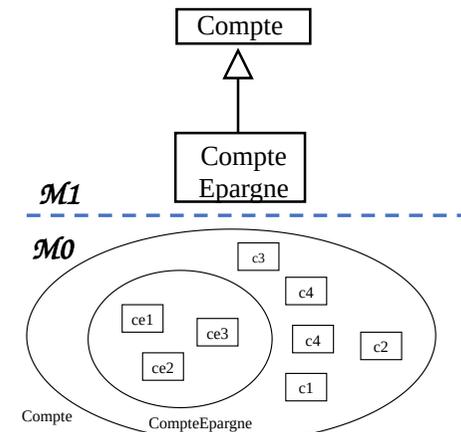
- relation X est-une-sort-de Y (est substituable à)
- organisation des systèmes complexes (cf. Linnaeus)
- réutilisation d'interface

- Mécanisme d'extension de module

- ajout de fonctionnalités dans la sous-classe
- "customisation" et combinaison de composants logiciels
- réutilisation de code

Héritage : Vision ensembliste

Tout objet instance d'une sous-classe appartient également à l'extension de la superclasse



Liskov's Substitution Principle (LSP)
Partout où un *Compte* est attendu, on peut toujours utiliser un *CompteEpargne* à la place

Héritage dans les langages à objets

- Java / Dart

```
class CompteEpargne extends Compte {
    void ajouterIntérêts() {...}
}
```

- Swift

```
class CompteEpargne : Compte {
    func ajouterIntérêts() {...}
}
```

- Python

```
class CompteEpargne (Compte):
    def ajouterIntérêts(self):
```

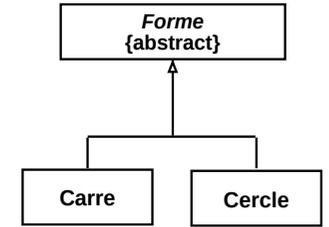
- Kotlin

```
class CompteEpargne : Compte() {
    fun ajouterIntérêts() {...}
}
```

- C++/ C#

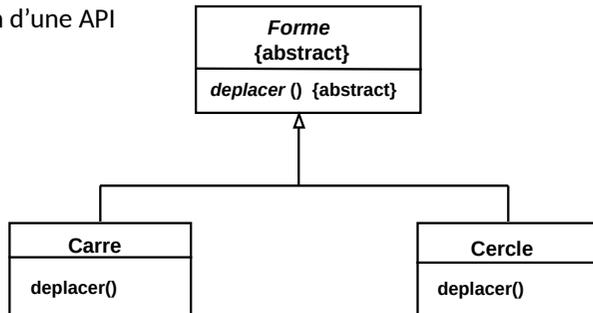
```
class CompteEpargne : Compte {
    void ajouterIntérêts() {...}
}
```

- Capturent des comportements communs
- Ne peuvent donc pas être instanciées
- Servent à structurer un système
- Peuvent avoir des opérations dont l'implantation est absente
 - *pure virtual* en C++
 - *abstract* en Java, C#, Kotlin etc.
- Classes sans instances immédiates



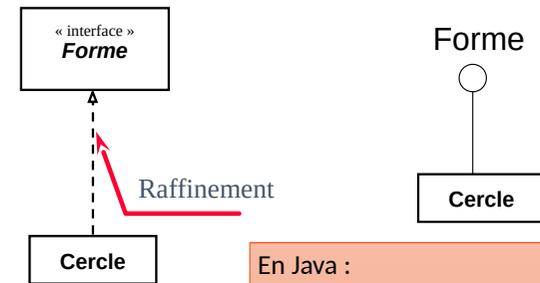
Une instance de «**Forme**» est obligatoirement une instance de la classe **Carre** ou de la classe **Cercle**

- Opération sans corps d'une classe abstraite
 - Définition d'une API



Interfaces et « lollipop »

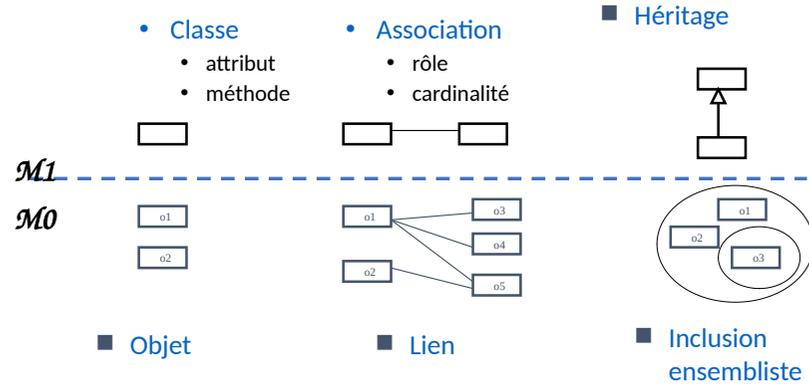
- Interface = classe « totalement » abstraite
 - Dont toutes les méthodes sont abstraites



En Java :

```
class Cercle implements Forme {
    ...
}
```

Synthèse des concepts de base



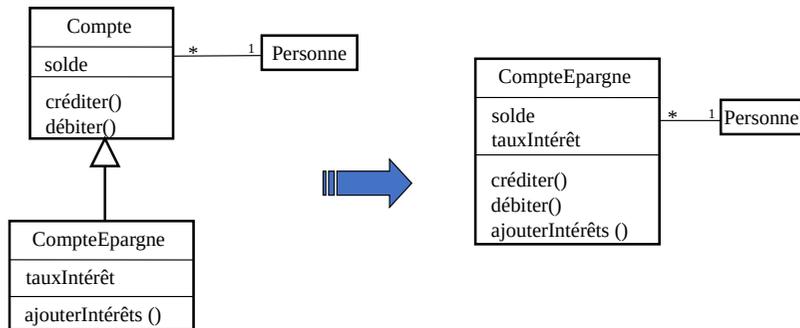
Deux points de vue complémentaires

- Mécanisme de classification : sous-typage
 - relation X est-une-sort-de Y (est substituable à)
 - organisation des systèmes complexes (cf. Linnaeus)
 - réutilisation d'interface

- Mécanisme d'extension de module
 - ajout de fonctionnalités dans la sous-classe
 - "customisation" et combinaison de composants logiciels
 - réutilisation de code

Relation d'héritage

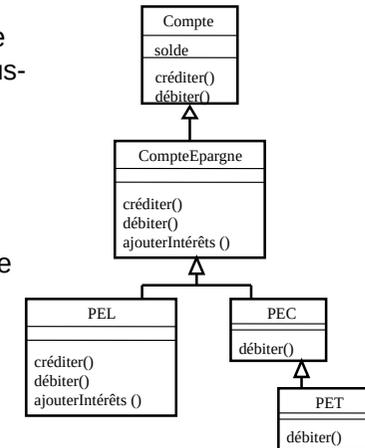
Les sous-classes « héritent » des propriétés des super-classes (attributs, méthodes, associations, etc.)



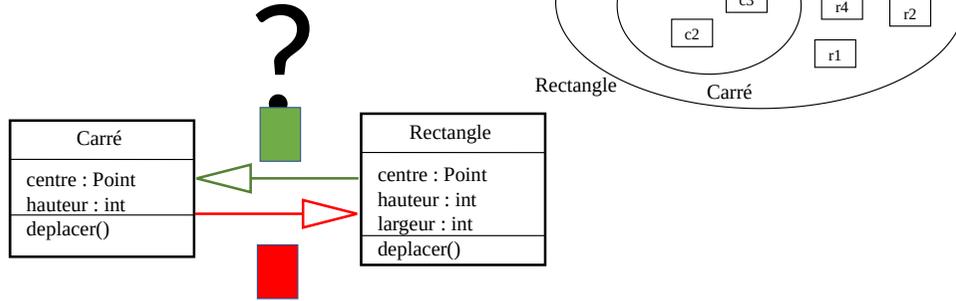
Relation d'héritage et redéfinitions

Une opération peut être "redéfinie" dans les sous-classes

Permet d'associer des méthodes spécifiques à chaque pour réaliser une même opération



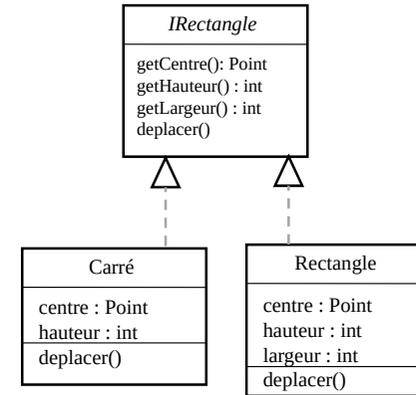
Carré vs. Rectangle



Parfois besoin de dissocier sous-typage et héritage d'implantation!

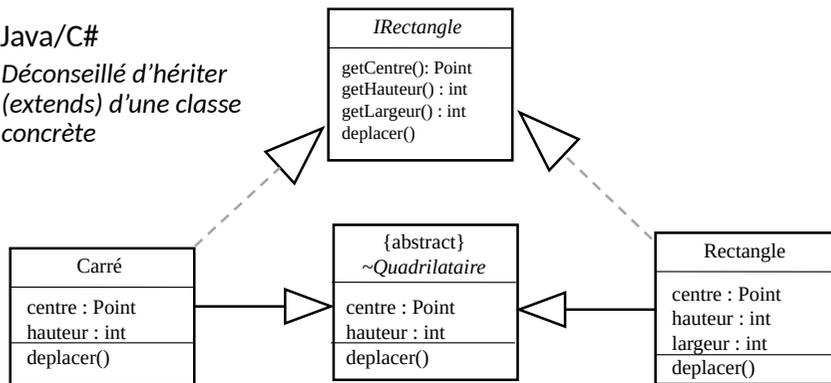
Distinguer Sous-Type de Sous-Classe

- En Java/C#



Distinguer Sous-Type de Sous-Classe

- En Java/C#
 - *Déconseillé d'hériter (extends) d'une classe concrète*



7 - Polymorphisme

Et liaison dynamique

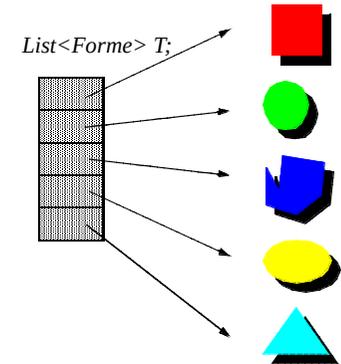
Polymorphisme et liaison dynamique

- Polymorphisme : possibilité de changer de forme
 - Forme f; c = new Cercle(); r = new Rectangle();
 - f = c;
 - f = r;
- Liaison dynamique : l'effet de l'appel d'une opération d'un objet dépend de sa forme effective à l'exécution
 - f.imprimer(); -- différent selon que f est Cercle ou Rectangle
- Espace de nommage réduit et uniforme
 - Mise en facteur des parties communes
 - Possibilité de "conteneurs" hétérogènes

Polymorphisme : exemple

- T contient des Formes
 - en fait des instances de sous-types de Forme
- Programmes de type :


```
T[0] = new Carré();
T[1] = new Cercle();
...
for (Forme f : T) f.imprimer();
```
- Si ajouts ultérieurs:
 - e.g. triangle
- Pas de modifications globales
 - case-less programming



Parcours d'arbre orienté objet

- Un Groupe contient
 - Des cercles, des rectangles...
 - ...et d'autres groupes
- Ce qui forme une structure d'arbre

```
void deplacer(double dx, double dy){
    for (Forme forme : formes) {
        forme.deplacer(dx, dy);
    }
}
```

```
void deplacer(double dx, double dy){
    formes.forEach(f->f.deplacer(dx,dy));
}
```

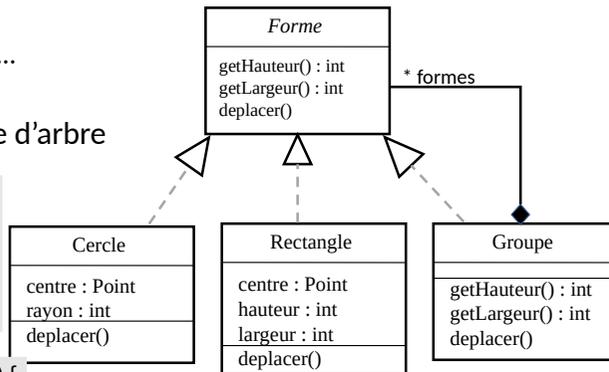
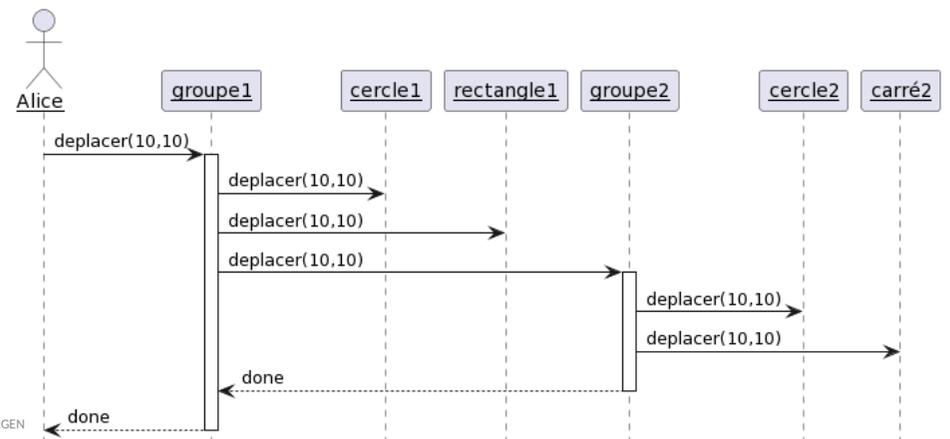


Illustration de la dynamique avec un diagramme de séquence

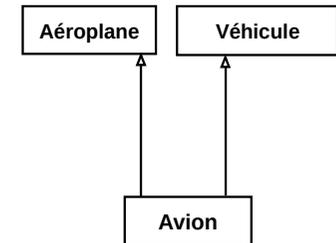


8 - Héritage multiple

Héritage entre classes

Héritage multiple

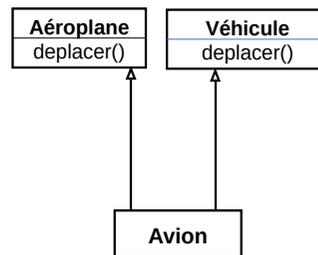
- Héritage multiple, exemple :
 - Un avion est à la fois
 - Un Aéroplane
 - Un Véhicule de transport
 - L'ensemble des avions est donc l'intersection entre
 - l'ensemble des aéroplanes et
 - l'ensemble des véhicules de transport
- Réalisation de ce concept très différente selon les langages de programmation
 - Seuls Python, C++ et Eiffel le supportent directement
 - Les autres (dont Java) le supportent indirectement



```
#Python
class
Avion(Aéroplane, Véhicule) :
```

Problème avec l'héritage multiple

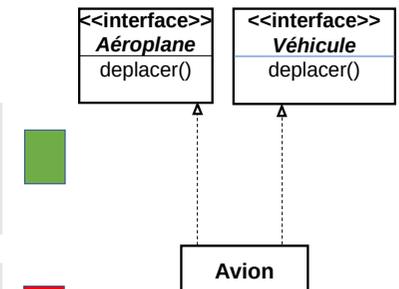
- Avion hérite de quelle version de déplacer()?
 - Aéroplane::déplacer()
 - Véhicule::déplacer()
- Des algorithmes complexes pour fonctionner dans le cas général (Python)
- Des constructions ad hoc pour aider le développeur à résoudre ces ambiguïtés (C++, Eiffel)
- Java : seule l'héritage multiple d'interfaces est autorisé



Héritage multiple d'interfaces en Java

```
// Java?
class Avion implements Aéroplane,
Véhicule {
//...
}
```

```
// Java?
class Avion extends Aéroplane, Véhicule {
//...
}
```



Avec la notion de Trait (Java 8)

```
public interface Aéroplane {
    public default void deplacer() {
        System.out.println("deplacer un aéroplane");
    }
}
```

```
public interface Véhicule {
    public default void deplacer() {
        System.out.println("deplacer un véhicule");
    }
}
```

```
public class Avion implements Aéroplane, Véhicule {
    public void deplacer() {
        Véhicule.super.deplacer();
    }
}
```

ISTIC - L2GEN

81

9 - Typage statique

Détecter des erreurs dès la compilation

Typage dans les langages objets

- Type de chaque objet est déterminé par la classe dont il est instance
 - Défini quelles opérations peuvent être appelées sur l'objet, et avec quels paramètres
 - $x.f()$ légal \Leftrightarrow l'objet désigné par x dispose d'une méthode $f()$
- liaison dynamique :
 - la *bonne* interprétation de f est choisie, selon le type de x

Typage dynamique

YOLO!

- Type de chaque objet est déterminé par la classe dont il est instance
 - Défini quelles opérations peuvent être appelées sur l'objet, et avec quels paramètres
- **Typage dynamique** : les variables peuvent désigner n'importe quel objet
 - Exemple : JavaScript ou Python
 - $c = Cercle()$; $c = Compte()$
 - Les erreurs de type sont **détectées pendant l'exécution** du programme
 - $c.crediter(100)$ // OK
 - $c.deplacer(10,20)$ // Erreur

Typage statique



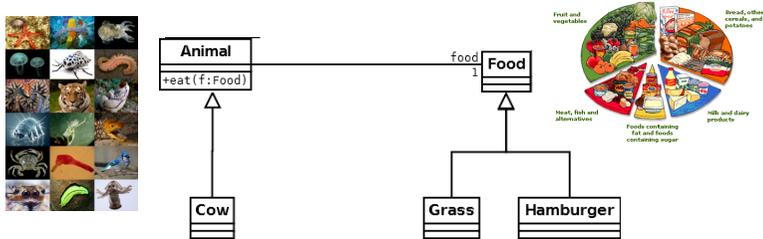
- Type de chaque objet est déterminé par la classe dont il est instance
 - Défini quelles opérations peuvent être appelées sur l'objet, et avec quels paramètres
- **Typage statique** : les variables doivent être déclarées avec un type spécifique
 - `Compte c = new Compte();`
 - Ce type ne peut généralement pas être modifié par la suite.
 - Les opérations sur les variables sont vérifiées par le compilateur pour s'assurer qu'elles respectent les règles du type déclaré
 - Les erreurs de type sont **détectées avant l'exécution** du programme
 - ce qui permet d'identifier et de corriger les problèmes liés aux types dès le stade de développement

Héritage et typage

- Typage statique :
 - `x.f()` légal \Leftrightarrow vérification avant l'exécution que tout objet potentiellement désigné par `x` dispose d'une méthode `f()`
- liaison dynamique :
 - la bonne interprétation de `f` est choisie
- Combinaison de ces notions
 - Python, JavaScript : typage dynamique, liaison dynamique
 - Java, C#, Swift, Dart, Kotlin : typage statique, liaison dynamique
 - C++ : typage statique, liaison dynamique
 - pour les fonctions "virtuelles"
 - UML : au choix (!)

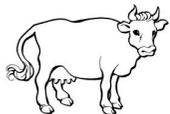


Co-variance et contra-variance

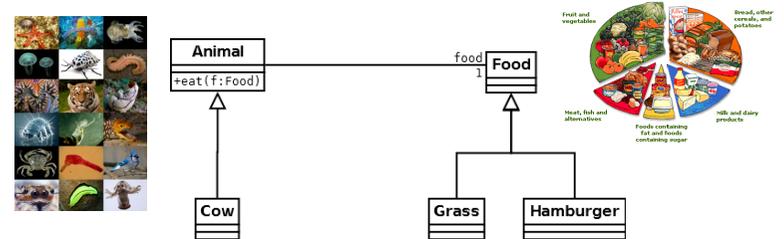


```
Animal a = new Animal()
Food f = new Food()
a.eat(f)
```

```
Animal a = new Cow()
Food f = new Food()
a.eat(f)
```

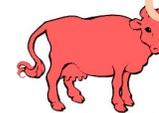
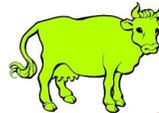


Co-variance et contra-variance



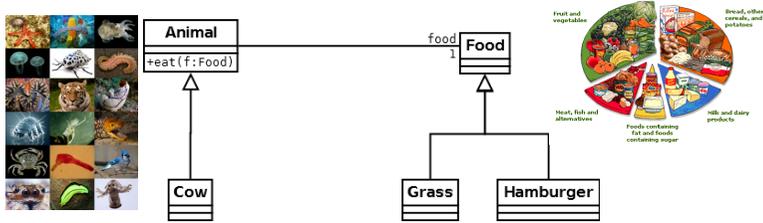
```
Cow a = new Cow()
Grass f = new Grass()
a.eat(f)
```

```
Cow a = new Cow()
Food f = new Hamburger()
a.eat(f)
```



Comment interdire ça ?

Co-variance et contra-variance

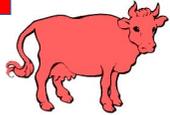


Redéfinition **covariante**

Ne règle pas le problème !

Légal en Java ? ■

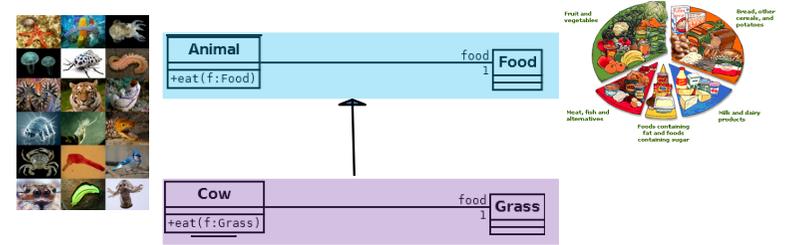
Illégal ? ■



```
Animal a = new Cow()
Food f = new Hamburger()
a.eat(f)
```



Co-variance et contra-variance



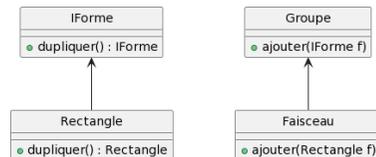
- En Java (& autres langages à typage statique) : généricité paramétrique
 - **Class Animal<T extends Food> {void eat(f:T){}}**
 - **Class Cow extends Animal<Grass>{} // devient implicitement eat(f:Grass)**
- Erreur `cow.eat(new Hamburger())` détectée à la compilation

Co-variance et contra-variance

- **Covariance**
 - **Définition** : Autorise la substitution d'un type dérivé (sous-type) à la place d'un type de base (super-type)
- **Contravariance**
 - Autorise la substitution d'un type de base à la place d'un type dérivé
- Règle en Java lors de la redéfinition de méthodes
 - Covariance pour les types de retours
 - Contravariance pour les paramètres
 - Exemples :

Légal ■

Illégal ■



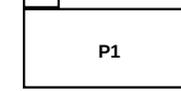
10 - Notion de package

Modularisation à l'échelle supérieure

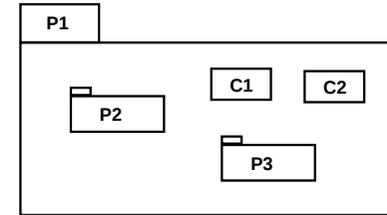
Notion de package

- Élément structurant les classes
 - Modularisation à l'échelle supérieure
 - Un package partitionne l'application :
 - Il référence ou se compose des classes de l'application
 - Il référence ou se compose d'autres packages
 - Un package réglemente la visibilité des classes et des packages qu'il référence ou le compose
 - Les packages sont liés entre eux par des liens d'utilisation, de composition et de généralisation
 - Un package est la représentation informatique du contexte de définition d'une classe

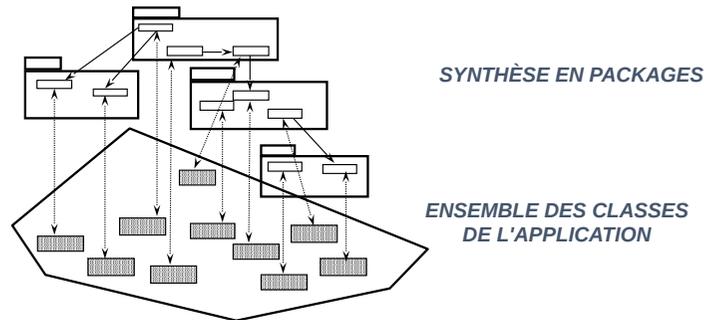
- Vue graphique externe



- Vue graphique externe et interne



- Définition de vues partielles d'une application



N.B.: une classe appartient à un et un seul package

Visibilité dans un package

- Réglementation de la visibilité des classes
 - **Classes de visibilité publique :**
 - classes utilisables par des classes d'autres packages
 - **Classes de visibilité privée :**
 - classes utilisables seulement au sein d'un package
- Représentation graphique



CLASSE D'INTERFACE



CLASSE DE CORPS



CLASSE EXTERNE

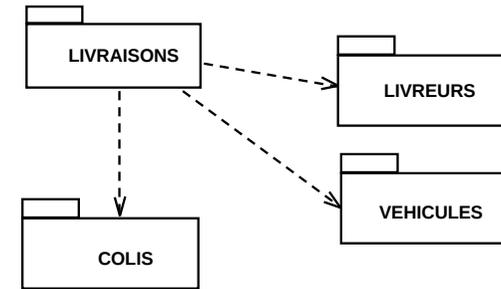
Utilisation entre packages

- Définition
 - Il y a utilisation entre packages si des classes du package utilisateur accèdent à des classes du package utilisé
 - Pour qu'une classe d'un package p1 puisse utiliser une classe d'un package p2, il doit y avoir au préalable une déclaration **explicite** de l'utilisation du package p2 par le package p1
- Représentation graphique



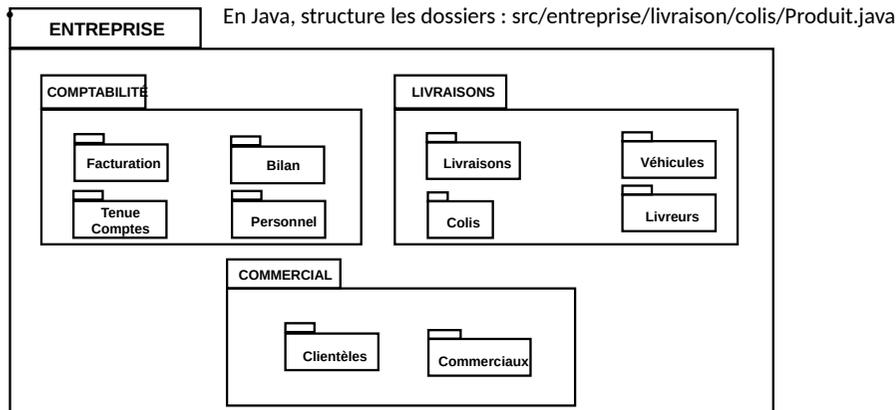
Vue externe du package P1

- Exemple (vue externe du package livraisons)

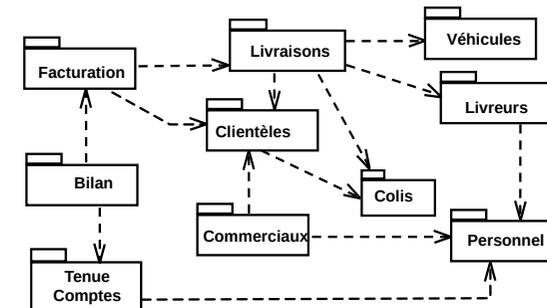


Architecture logicielle : vue hiérarchique

- Composition des packages en sous-packages



Architecture logicielle : dépendances

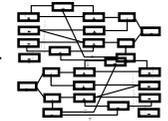
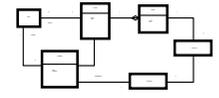


Utilité des packages

- Réponses au besoin
 - Contexte de définition d'une classe
 - Unité de structuration
 - Unité d'encapsulation
 - Unité d'intégration
 - Unité de réutilisation
 - Unité de configuration
 - Unité de production

Synthèse diagrammes de classes et d'objets

- Un diagramme de classes
 - Défini l'ensemble de tous les états possibles
 - les contraintes doivent toujours être vérifiées
- Un diagramme d'objets
 - décrit un état possible à un instant t, un cas particulier
 - doit être conforme au modèle de classes
- Les diagrammes d'objets peuvent être utilisés pour
 - expliquer un diagramme de classe (donner un exemple)
 - valider un diagramme de classe (le "tester")



Éléments de Génie Logiciel

Cours 4

Gestion de Version

Jean-Marc Jézéquel

(d'après Thomas Genêt)

IRISA/ISTIC - Université de Rennes

e-mail : jezequel@irisa.fr

<http://people.irisa.fr/Jean-Marc.Jezequel>

Pourquoi la gestion de version ?

- 1 Développer **collaborativement un code commun**, en évitant



- 2 **Sauvegarder** le code, pour qu'il survive à



- 3 **Revenir à une version antérieure** du code, si la dernière version fait



Déroulement du TP 1-2 par rapport au CMM ?

Capability Maturity Model = évaluation du processus de développement

Niveau 1, Initial/Chaotique/Héroïque

- Développement dans l'urgence, non reproductible
- Le succès du projet dépend d'un petit nombre d'individus
- Comportement du logiciel non prédictible (partiellement testé)

Objectif de l'UE GEN : passer au **Niveau 2** :

Niveau 2, Reproductible/Discipliné

- Sauvegarde, partage, restauration projet, gestion de version (CM4)
- Comportement du logiciel prédictible, défauts détectés (test systématique : CM5)
- Assurance qualité du logiciel, documentation (CM6)
- Tâches définies pour les développeurs, qui fait quoi ? (CM7)

Il reste 3 niveaux au dessus ☺ : défini, maîtrisé, optimisant

Gestion de versions... quel est le problème ?

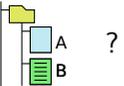
Il était une fois Jo et Mo qui travaillaient sur le même projet...

- Ils copient sur leur machine le projet qui ressemble à ça : 
- Mo modifie le fichier A : 
- Jo ajoute un fichier B : 
- Comment font ils pour obtenir chacun une copie de  ?
- Combien de copies entre Jo et Mo sont nécessaires (au min.) ?
... et s'il y avait eu N développeurs et N modifications ?

Gestion de versions... ça se complique, Mo est une truffe !

Mo s'est trompé et souhaite annuler sa modification de A

- A partir de 

- Comment font ils pour obtenir chacun une copie de  ?
- Combien de copies entre Jo et Mo sont nécessaires (au min.) ?
- ... et pour N développeurs ?

Remarque 1 (Ca n'est possible que si au moins un développeur...)

- garde un historique des projets qu'il copie
- sait quels sont les fichiers/répertoires à ajouter/supprimer d'une version à l'autre

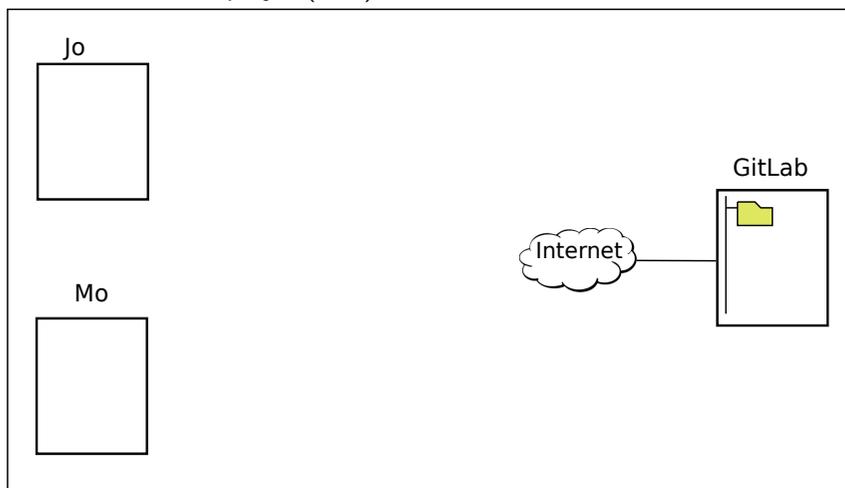
C'est précisément ce que fait un gestionnaire de versions, y compris sur les fichiers eux-mêmes : (suppression, ajout, modification de ligne)

Gestion de versions avec Git, le principe

- Chaque développeur à un **dépôt Git local** qui enregistre toutes les modifications opérées sur **son** projet (toutes les versions)
- Les développeurs qui le souhaitent peuvent synchroniser leur **dépôt local** avec un **dépôt partagé** (sur le GitLab ISTIC pour nous)
- Les développeurs interagissent avec Git avec les commandes :
 - ▶ créer le dépôt local : `init`
 - ▶ signaler des modifications/ajouts sur des fichiers/répertoires : `add`
 - ▶ supprimer un fichier : `rm` , supprimer un répertoire : `rm -r`
 - ▶ propager les modif./ajouts/suppressions sur le dépôt Git local : `commit`
 - ▶ propager les modif./ajouts/suppr. du Git local vers le GitLab : `push`
 - ▶ obtenir une **copie locale** d'un dépôt GitLab : `clone`
 - ▶ obtenir les dernières modifications faites sur le GitLab : `pull`

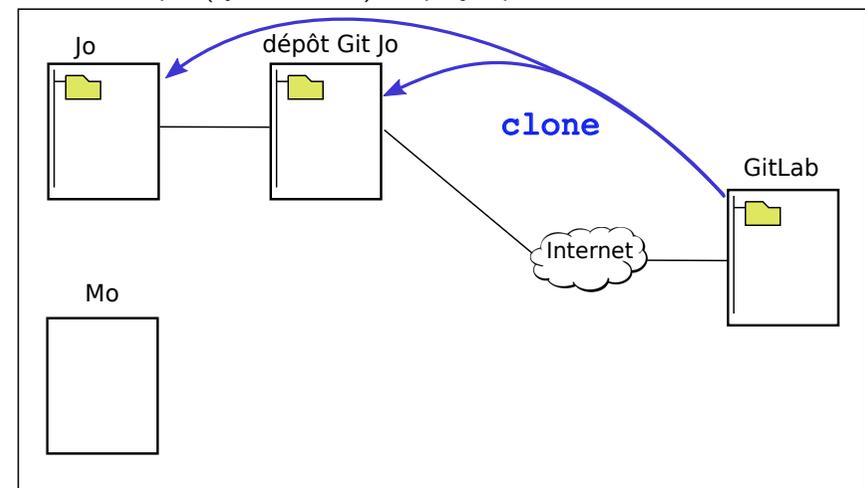
Il était une fois... Jo et Mo avec Git (l'initialisation)

Jo crée un nouveau projet (vide) sur GitLab



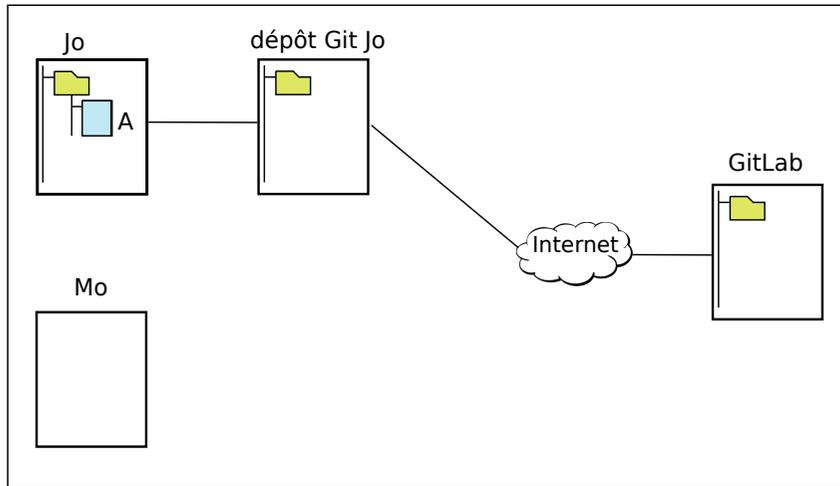
Il était une fois... Jo et Mo avec Git (l'initialisation)

Jo fait une copie (synchronisée) du projet provenant du GitLab



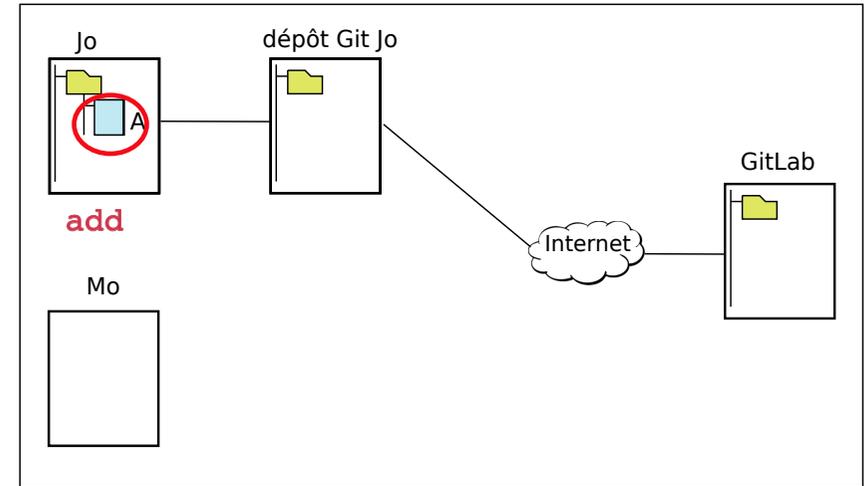
Il était une fois... Jo et Mo avec Git (l'initialisation)

Jo ajoute un nouveau fichier A dans sa copie du projet



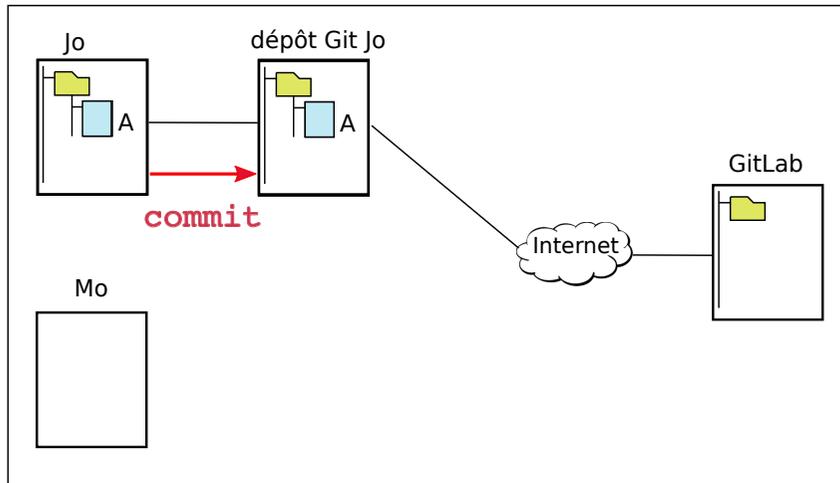
Il était une fois... Jo et Mo avec Git (l'initialisation)

Jo indique les fichiers/répertoires à ajouter à Git (a.k.a. staging)



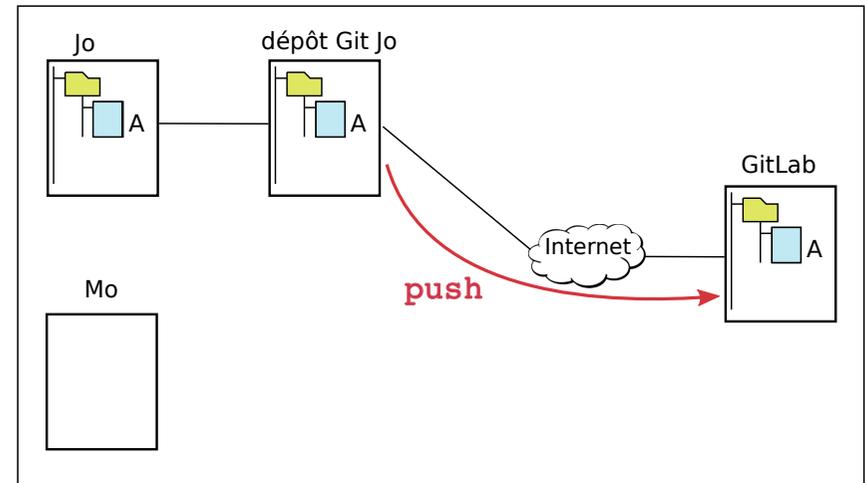
Il était une fois... Jo et Mo avec Git (l'initialisation)

Jo envoie les nouveaux fichiers/répertoires vers **son** dépôt Git



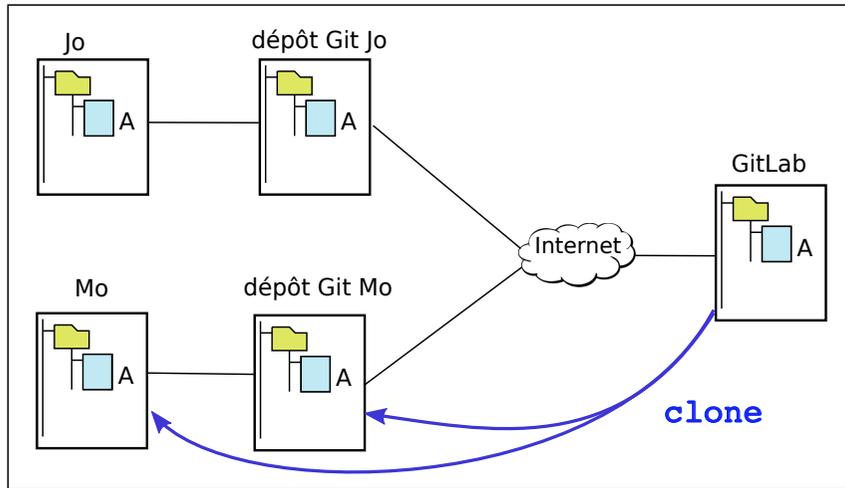
Il était une fois... Jo et Mo avec Git (l'initialisation)

Jo **pousse** ses modifications vers le GitLab



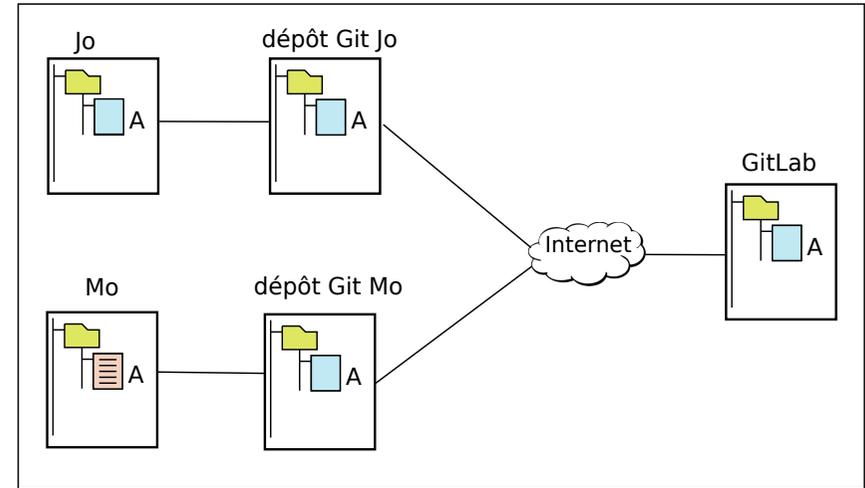
Il était une fois... Jo et Mo avec Git (l'initialisation)

Mo fait une copie (synchronisée) du dépôt sur le GitLab



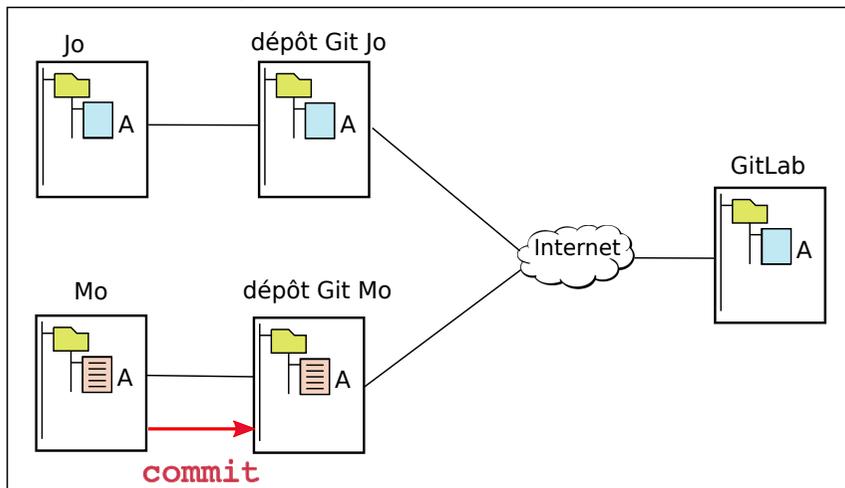
Il était une fois... Jo et Mo avec Git (utilisation)

Si Mo modifie le fichier A et veut envoyer la modification vers GitLab



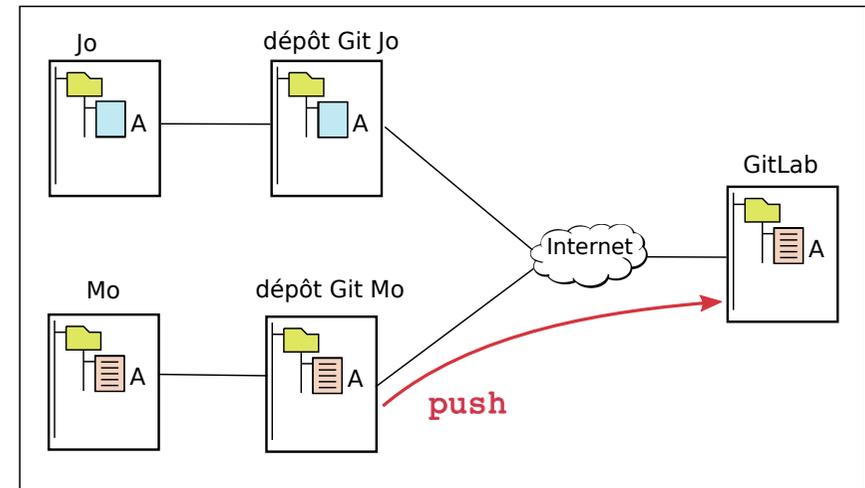
Il était une fois... Jo et Mo avec Git (utilisation)

Mo envoie les fichiers/répertoires modifiés vers son dépôt Git



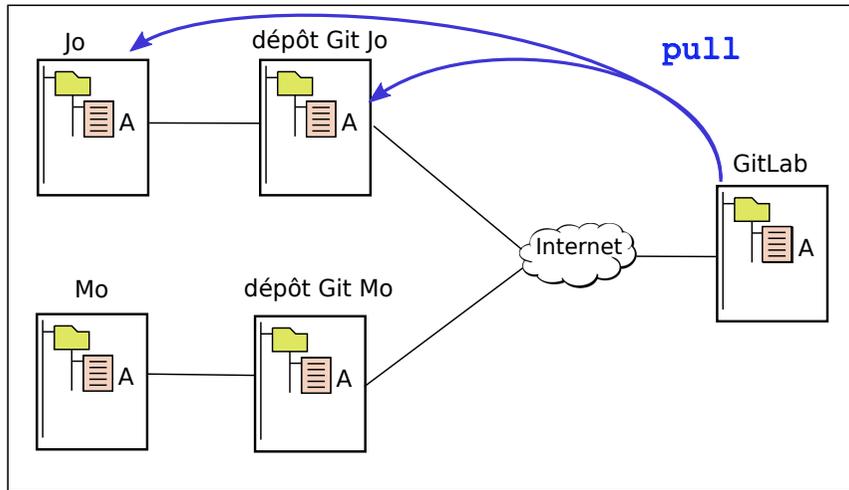
Il était une fois... Jo et Mo avec Git (utilisation)

Mo **pousse** ses modifications vers le GitLab



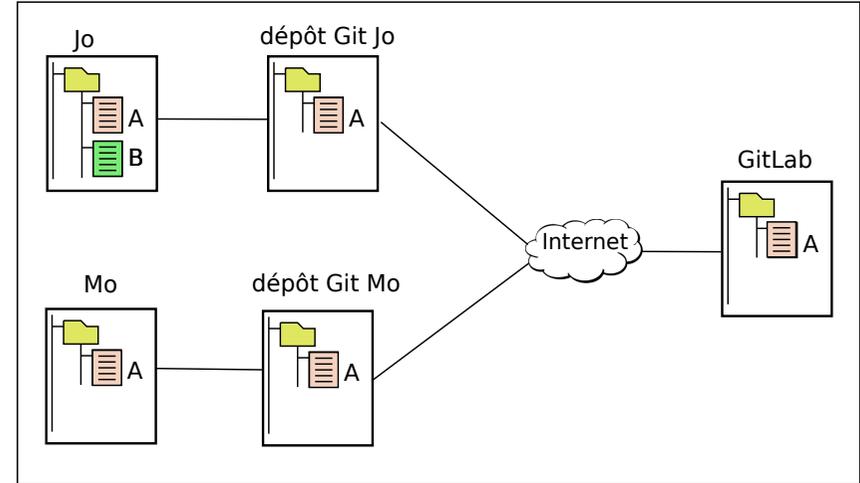
Il était une fois... Jo et Mo avec Git (utilisation)

Jo tire/applique les modifications du GitLab vers son dépôt et répertoire



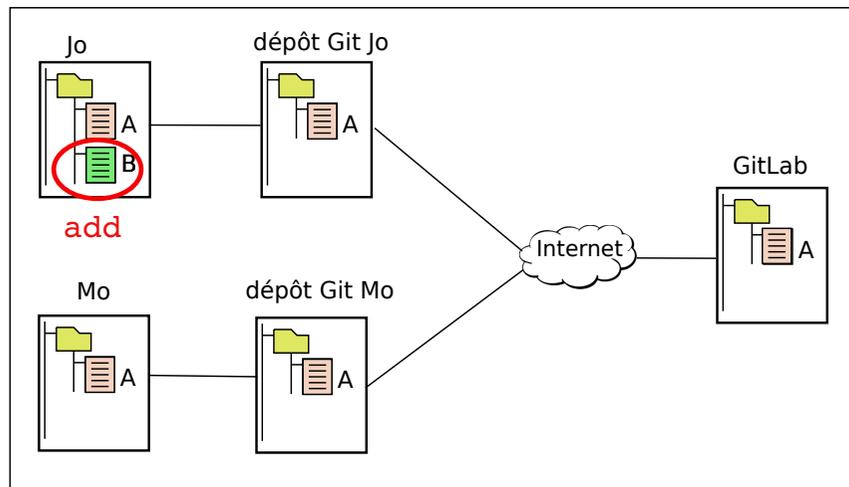
Il était une fois... Jo et Mo avec Git (utilisation)

Si Jo ajoute un fichier B et veut envoyer la modification vers GitLab



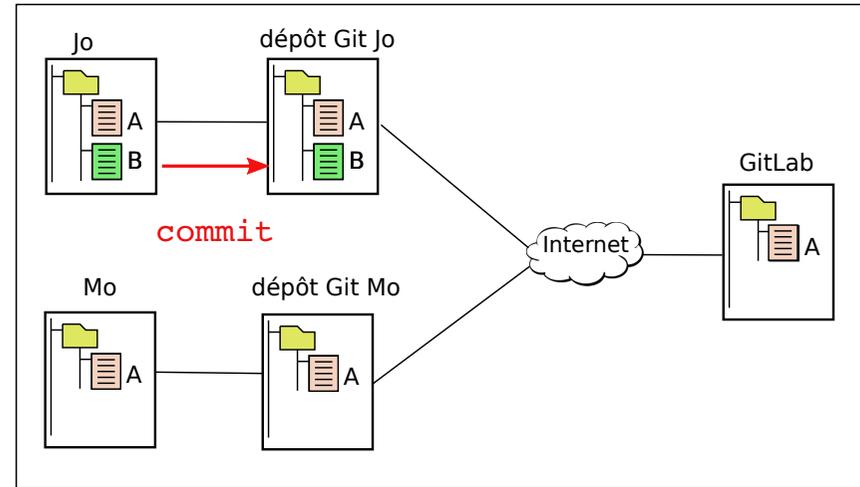
Il était une fois... Jo et Mo avec Git (utilisation)

Jo indique quels sont les fichiers à propager vers Git (a.k.a. staging)



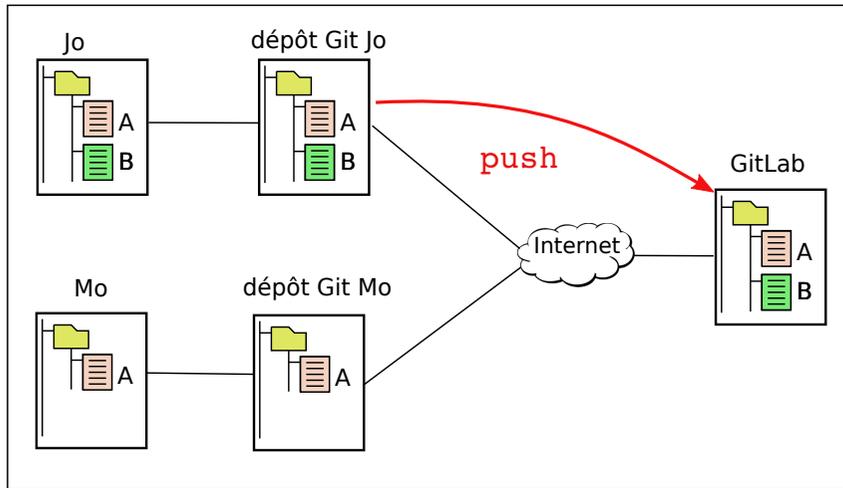
Il était une fois... Jo et Mo avec Git (utilisation)

Jo envoie le fichier supplémentaire vers son dépôt Git



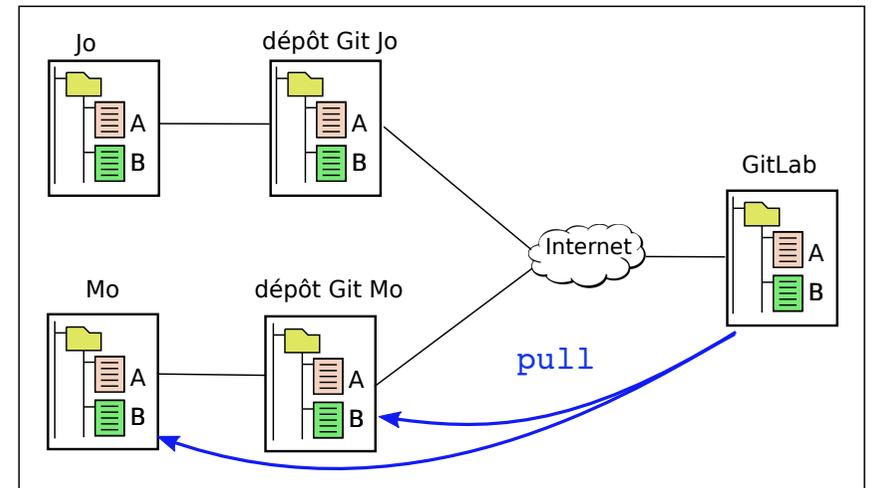
Il était une fois... Jo et Mo avec Git (utilisation)

Jo **pousse** ses modifications vers le GitLab



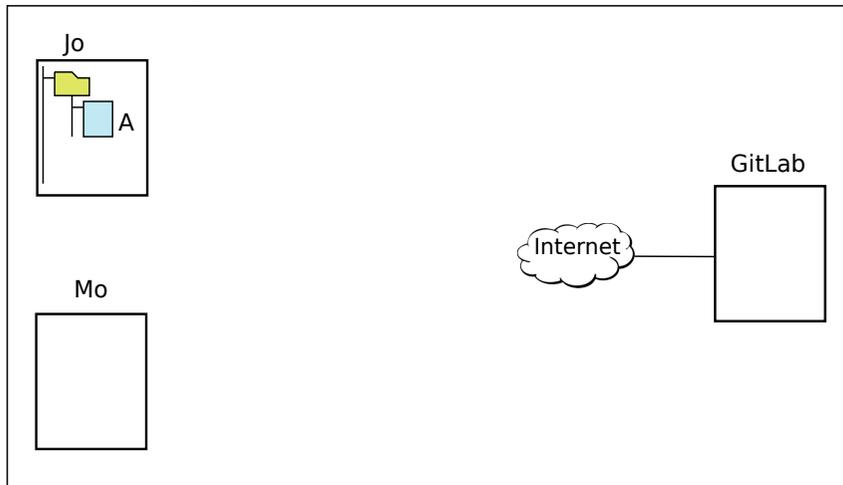
Il était une fois... Jo et Mo avec Git (utilisation)

Mo **tire/applique** les modifications du GitLab vers son dépôt et répertoire



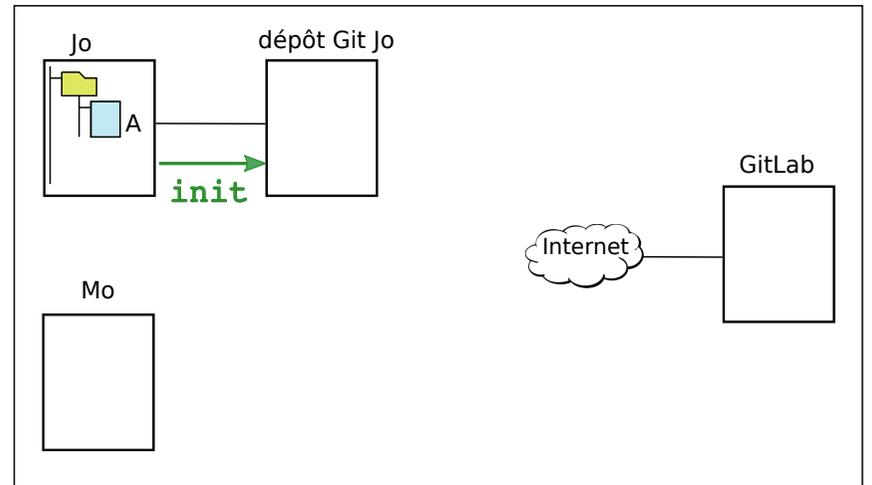
Jo et Mo avec Git (l'initialisation, une autre méthode)

Jo souhaite partager le projet contenu dans son répertoire



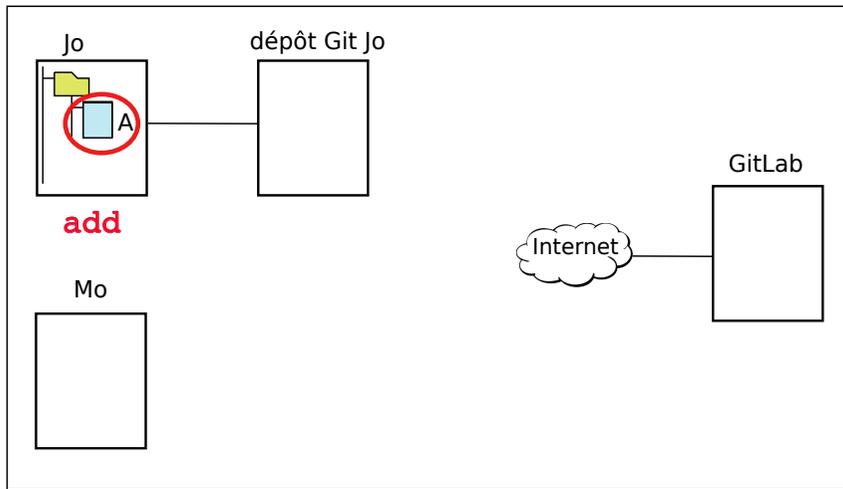
Jo et Mo avec Git (l'initialisation, une autre méthode)

L'initialisation du dépôt de Jo



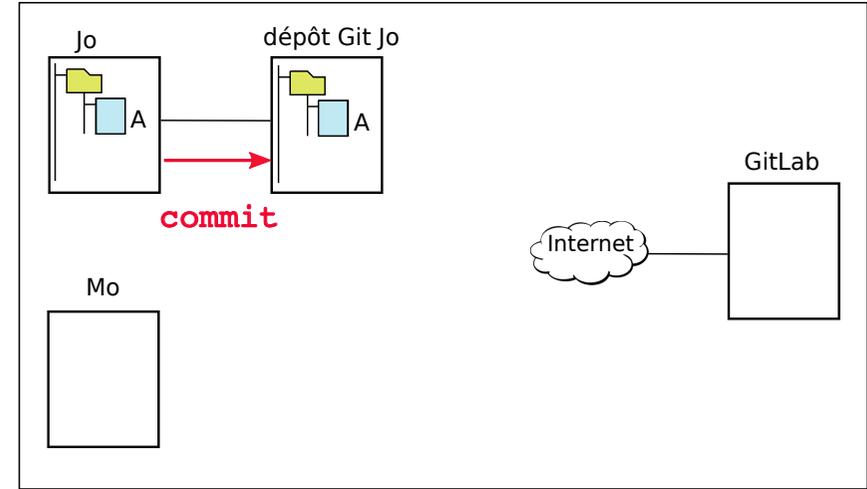
Jo et Mo avec Git (l'initialisation, une autre méthode)

Jo indique quels sont les fichiers/répertoires à ajouter Git (a.k.a. staging)



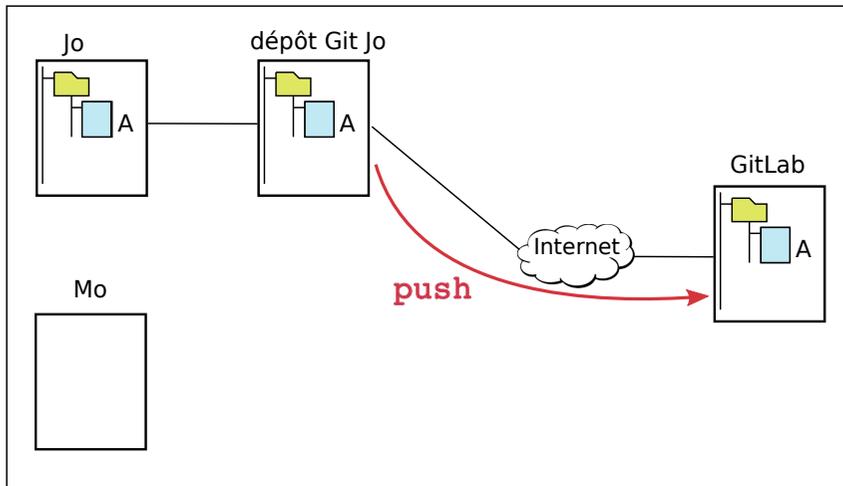
Jo et Mo avec Git (l'initialisation, une autre méthode)

Jo envoie les nouveaux fichiers/répertoires vers **son** dépôt Git



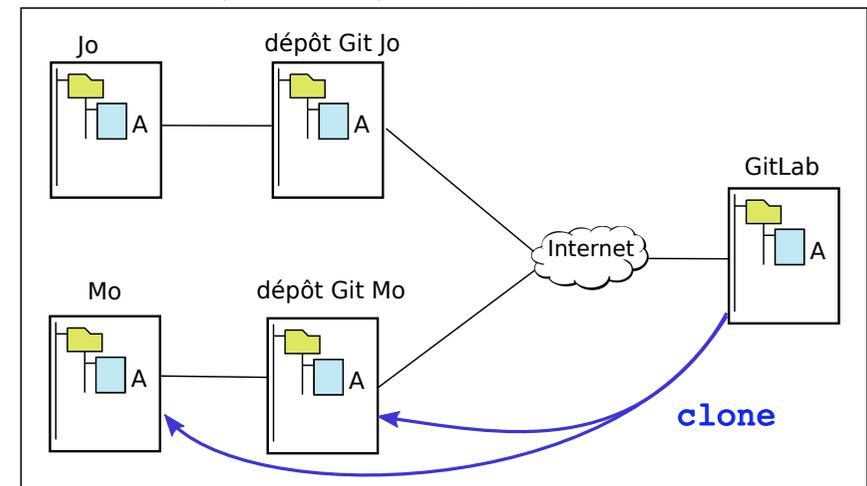
Jo et Mo avec Git (l'initialisation, une autre méthode)

Jo **pousse** ses modifications vers le GitLab



Jo et Mo avec Git (l'initialisation, une autre méthode)

Mo fait une copie (synchronisée) du dépôt sur le GitLab



Jo, Mo et Git, en vrai...

- Git s'utilise dans un terminal à l'aide de la commande git
- Il existe une multitude de clients graphiques pour Git...
- En TP nous utiliserons : Git en ligne de commande

Remarque 2 (Le GitLab ISTIC)

- ▶ <https://gitlab.istic.univ-rennes1.fr/>
- ▶ *Connectez-vous (au moins une fois) sur le GitLab istic avant le prochain TP!*
- ▶ *Créez un projet et tentez d'y déposer des fichiers*
- ▶ *Le code de vos projets GitLab sera consultables par un navigateur (Démo)*

Jo, Mo et Git, dans le terminal (initialisation)

Exemple 1 (Jo clone un projet gen1 auquel il a accès par le GitLab)

```
% git clone https://gitlab.istic.univ-rennes1.fr/jo/gen1.git
% cd gen1 # le répertoire de travail gen1 a été crée par git
```

Exemple 2 (Jo ajoute le fichier lettre.txt au projet (staging))

```
% git add lettre.txt
```

Exemple 3 (Jo envoie les modifications vers son dépôt Git local)

```
% git commit -m "ajout d'une lettre pour le père Noël"
```

Exemple 4 (Jo pousse ses modifications vers le GitLab)

```
% git push
```

Exemple 5 (Mo clone le projet gen1 auquel il a accès par le GitLab)

```
% git clone https://gitlab.istic.univ-rennes1.fr/jo/gen1.git
% cd gen1
```

Jo, Mo et Git, dans le terminal (utilisation)

On suppose que le répertoire de travail est gen1 (on a fait cd gen1)

Exemple 6 (Mo signale une modification sur lettre.txt (staging))

```
% git add lettre.txt
```

Exemple 7 (Mo envoie les modifications vers son dépôt Git local)

```
% git commit -m "ajout de remerciements"
```

Exemple 8 (Mo pousse ses modifications vers le GitLab)

```
% git push
```

Exemple 9 (Jo obtient les modifications faites sur le GitLab)

```
% git pull
```

Jo, Mo et Git, en vrai... dans le terminal (III)

Quiz 1 (Cette suite de commande échoue ? Oui Non)

```
% git add monFichier
% git commit -m "modifications sur monFichier"
% git add monFichier
% git commit -m "on commite sans avoir modifié monFichier"
```

Quiz 2 (Cette suite de commande détruit définitivement monFichier ?)

```
% rm monFichier
% git commit -m "Suppression de monFichier"
% git push
```

<input checked="" type="checkbox"/>	Oui
<input type="checkbox"/>	Non

Quiz 3 (Cette suite de commande détruit définitivement monFichier ?)

```
% git rm monFichier
% git commit -m "Suppression de monFichier"
% git push
```

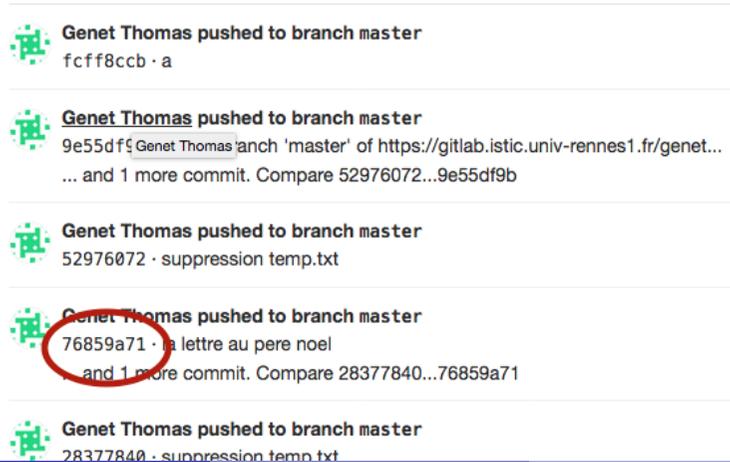
<input checked="" type="checkbox"/>	Oui
<input type="checkbox"/>	Non

Dans "gestionnaire de version", il y a le mot "version"

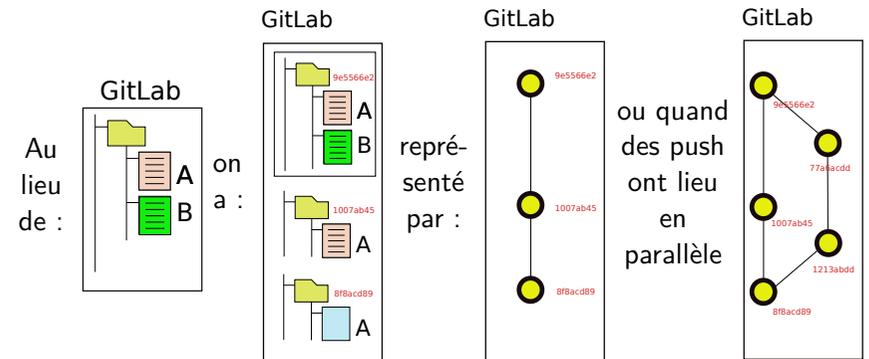
Remarque 3 (En Git chaque version a un identifiant unique)

Git conserve l'historique complet de toutes les versions du projet.

Dans GitLab, l'onglet "Activity" donne les identifiants des versions



Dans "gestionnaire de version", il y a le mot "version"



Dans "gestionnaire de version", il y a le mot "version"



Exemple 10 (Jo replace **tout** son projet dans la version 76859a71)

```
% git reset --hard 76859a71
```

Attention ! on perd les modifications du projet non "commitées" dans Git !

Exemple 11 (Jo restaure le fichier temp1.txt de la version courante)

```
% git checkout temp1.txt
```

Attention ! on perd les modifications sur temp1.txt non "commitées" dans Git !

Dans "gestionnaire de version", il y a le mot "version"

Exercice 1

- 1 Comment récupérer un fichier détruit dans le répertoire local ?
- 2 Comment récupérer un fichier supprimé de la dernière version du GitLab ?

Quiz 4

Initialement, on suppose que le projet est dans la version 52976889 et que l'on tape :

```
% rm temp1.txt  
% git reset --hard 76859a71
```

Comment remettre le projet dans sa forme initiale ?

<pre>% git reset --hard 52976889</pre>	<pre>% git checkout temp1.txt</pre>
V	R

La vie n'est pas un long fleuve tranquille : les conflits

Définition 12 (Conflit de version)

Si deux développeurs font des modifications contradictoires sur un même fichier, un conflit peut apparaître. Il touchera le dernier des deux faisant un **push**. Ce dernier sera le seul à pouvoir résoudre le conflit.

En pratique, si vous suivez les principes de **modularité** présentés aux cours précédents, les conflits de versions devraient être **très rares**, mais... (voir le titre)

Exemple 13 (Exemple de conflit de version)

Jo et Mo modifient différemment la ligne *i* du fichier B de la dernière version. Jo fait son **push**. Ensuite, Mo tente de faire son **push**. Celui-ci échoue car sa version n'est plus à jour. Mo fait un **pull** qui révèle un conflit sur la ligne *i* du fichier B.

Résolution de conflit Git sur un fichier B

- 1 Rechercher les annotations de conflits dans le texte du fichier B :

```
Le texte précédant la partie de texte en conflit.  
<<<<<<< HEAD  
C'est le texte que je voulais proposer.  
=====  
C'est le texte proposé par l'autre développeur.  
>>>>>> e4e1e956f6426c04ab80487f612265fe5c95ece8  
La suite du texte qui n'est pas en conflit.
```

- 2 Résoudre le conflit dans le texte du fichier B :

```
Le texte précédant la partie de texte en conflit.  
C'est le texte que je voulais proposer complété par le  
texte proposé par l'autre développeur.  
La suite du texte qui n'est pas en conflit.
```

- 3 Informer le dépôt de la résolution du conflit sur B :

```
% git add B  
% git commit -m "Conflit sur 2eme ligne résolu"  
% git push
```

Résolution de conflit Git sur un fichier B (II)

Quiz 5 (Ces séquences de commandes sont-elles possible ?)

Sur la machine de Jo	Sur la machine de Mo
% cat temp1.txt Le texte de ce document est très court	% cat temp1.txt Le texte de ce fichier est vraiment très court
% git add temp1.txt	% git add temp1.txt
% git commit -m "modif Jo"	% git commit -m "modif Mo"
% git push	% git push
% git pull	% git pull

V Oui R Non

Si Mo fait son push avant Jo, il aura un conflit V Vrai R Faux



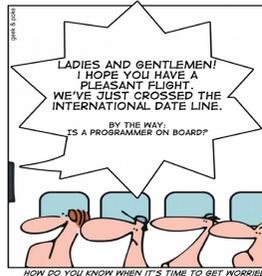
Pr. Jean-Marc Jézéquel
IRISA – Université de Rennes
Campus de Beaulieu
F-35042 Rennes Cedex
e-mail : jezequel@irisa.fr
<http://people.irisa.fr/Jean-Marc.Jezequel>
X @jmjezequel

1 - Introduction

Qu'est-ce que le test et pourquoi tester?



Ariane V (1996) :
Conversion d'un float 64-bit en int 16-bit



Mars Climate Orbiter (1998) :
Comparaison de valeurs dans des unités de mesures différentes



Nissan's Airbag Software Malfunction (2016)
ISTIC - LZGEN

Windows : the Blue Screen of Death

```
A problem has been detected and Windows has been shut down to prevent damage to your computer.
The problem seems to be caused by the following file: kbdhid.sys
MANUALLY_INITIATED_CRASH
If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:
Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use safe mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.
Technical Information:
*** STOP: 0x000000e2 (0x00000000, 0x00000000, 0x00000000, 0x00000000)
*** kbdhid.sys - Address 0x94efd1aa base at 0x94efb000 DateStamp 0x4a5bc705
```

Vocabulaire

Note : Il est possible de remplacer « développeur » par « assistant automatique » (LLM/generative AI)



- ... une **erreur** du développeur introduit ...
 - *Une erreur est une décision inappropriée ou erronée, faite par un développeur, qui conduit à l'introduction d'un défaut.*
- ... un **défaut** dans le système qui provoquera ...
 - *Un défaut est une imperfection dans un des aspects du système qui contribue, ou peut potentiellement contribuer, à la survenance d'une ou de plusieurs défaillances*
 - *Parfois, il faut plusieurs défauts pour causer une défaillance.*
- ... sa **défaillance** à l'exécution.
 - *Une défaillance est un comportement inacceptable présenté par un système.*
 - *La fréquence des défaillances reflète la fiabilité.*

Vocabulaire

« Le test est un processus **manuel** ou **automatique**, qui vise à établir qu'un système **vérifie** les propriétés exigées par sa spécification, ou à **détecter** des différences entre les résultats engendrés par le système et ceux qui sont attendus par la spécification »

Extrait de la norme IEEE-STD729, 1983

Principes

Le Test est le moyen le plus utilisé en pratique pour apporter de la **confiance** concernant le bon fonctionnement d'un logiciel

Le test ne certifie pas le bon fonctionnement

Améliorer la confiance

=

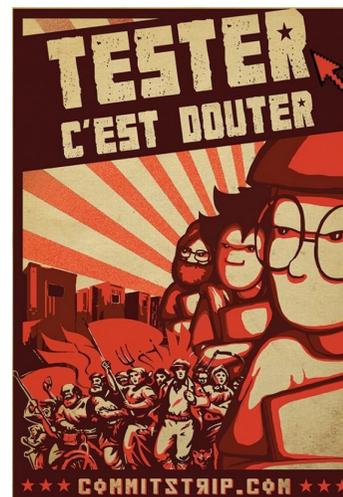
Améliorer la qualité des suites de tests



Principes

Tester pour apporter de la **confiance** concernant le bon fonctionnement du programme

- Fonctionnalité
- Sécurité / intégrité
- Utilisabilité
- Cohérence
- Maintenabilité
- Efficacité
- Robustesse
- Sûreté de fonctionnement
- etc.



Exemple

```
private Label label;           Code correct ■   défectueux? ■
public Button createButton() {
    Button btn = new Button();
    EventHandler handler = event ->
    label.setText("Hello World!");
    return btn;
}
```

Spécification : un clic sur le bouton doit afficher « Hello World ! »

Défaillance : un clic sur le bouton ne produit aucun effet

Défaut : l'instruction `btn.setOnAction(handler)` est manquante

Erreur : le programmeur a oublié d'écrire `btn.setOnAction(handler)`

Test : un test (automatisé ou non) fait en sorte de cliquer sur le bouton et vérifie que « Hello World ! » est bien affiché

2 - Comment tester?

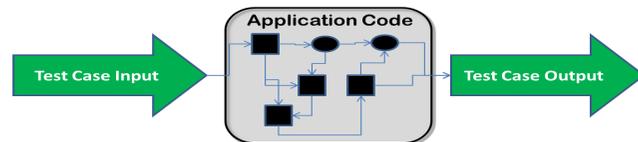
- Test statique
 - relecture / revue de code
 - analyse automatique
 - vérification de propriétés, règles de codage...
- Test dynamique
 - on exécute le programme avec des valeurs en entrée et on observe le comportement

Principes

- Test fonctionnel (test boîte noire)
 - Utilise la description des fonctionnalités du programme



- Test structurel (test boîte blanche)
 - Utilise la structure interne du programme



Test Dynamique

- **Statique** : le code n'est pas exécuté
- **Dynamique** : le code est exécuté et on analyse le résultat de son exécution
- Il existe de nombreuses techniques de test dynamique
 - Test Unitaire (TU)
 - Test d'intégration
 - Test système ou test E2E (end-to-end, de bout en bout)
 - Test d'acceptation
 - Test de montée en charge
 - ...

Test Dynamique

Cas de test

- Un cas de test = une méthode
- Corps de la méthode
 - Configuration initiale
 - Des données de test
 - Exécution du scénario
 - Un oracle
 - il faut construire le résultat attendu
 - ou vérifier des propriétés sur le résultat obtenu
- Une classe de test pour une classe testée
 - Regroupe les cas de test
 - Il peut y avoir plusieurs classes de test pour une classe testée

Test dynamique

Cas de test

- Un cas de test = une méthode
- Corps de la méthode
 - Configuration initiale


```

          @Test
          public void testRemoveRoue() {
              Roue roue = new Roue();
              Velo velo = new Velo();
          
```
 - Exécution du scénario


```

              velo.addRoue(roue);
              velo.removeRoue(roue);
          
```
 - Un oracle de test


```

              assertTrue(velo.getRoues().isEmpty());
          }
          
```

 - Qui compare le résultat observé au résultat attendu

Test dynamique

- Concepts partagés dans la plupart des langages de programmation
- Java : **JUnit** 4 ou 5
 - <http://junit.org/> VS Code intègre ces outils de test
- Python :
 - **unittest** intégré dans la bibliothèque standard de Python
- C#
 - **MSTest** intégré dans l'écosystème de développement avec Visual Studio
- C++
 - GoogleTest (<https://github.com/google/googletest>)
 - CppUnit <http://freedesktop.org/wiki/Software/cppunit/>
 - Cpputest <http://cpputest.github.io/>

3 - Zoom sur JUnit

Test dynamique avec JUnit 4

Package Java : **doit être le même que celui de la classe testée**

```
package test;
```

```
import static org.junit.Assert.*;
import org.junit.Test;
```

Importation de packages nécessaires

```
public class TestNombre {
```

Définition de la classe de test : *TestNomClasse* Ou *NomClasseTest*

```
@Test
void test() {
    fail("Not yet implemented");
}
```

Annotations pour marquer la méthode comme une méthode de test

Code du test

Test dynamique avec JUnit

Code Java à tester dans dossier « src/fr/istic/monProjet »

```
package fr.istic.monProjet;

public class Nombre {
    int valeur;
```

Code de test dans le dossier « test/fr/istic/monProjet »

```
public Nombre(int val) {
    valeur = val;
}

public int multiplier(int val) {
    return valeur*val;
}
```

```
package fr.istic.monProjet;
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
```

```
public class TestNombre {
    @Test
    void testMultiplier() {
        Nombre nb = new Nombre(2);
        assertEquals(6, nb.multiplier(3));
    }
}
```

Création d'un objet à tester

Vérification d'une propriété

Test dynamique avec JUnit dans VS Code

Lancer les tests à l'aide de



The screenshot shows the VS Code interface with a Java file named 'PointTest.java' open. The code includes package declarations, imports for JUnit, and a test method 'testDiv()' that uses 'assertEquals'. The Test Results panel on the right shows the test results, including a failed test 'testDiv()' with the message 'Not yet implemented'. A blue arrow points from the JUnit icon in the top left to the test runner button in the left sidebar.

Test dynamique avec JUnit

Les assertions

Permettre de vérifier des propriétés sur l'objet à tester

`assertEquals(valAttendue, valObservée);` Vérifie que *valObservée* et *ValAttendue* sont égales

`assertNull(obj);` Vérifie que *obj* est nul

`assertNotNull(obj);` Vérifie que *obj* n'est pas nul

`assertFalse(bool);` Vérifie que *bool* est faux

`assertTrue(bool);` Vérifie que *bool* est vrai

...

Test dynamique avec JUnit

- L'annotation `@Test` identifie un test
- Pas de `main` : chaque opération annotée par `@Test` est exécutée comme un programme
- Un test contient des **assertions**
- Un test possède un **oracle**
 - Oracle = objet (personne, instruction, etc.) connaissant le résultat attendu du test
 - Les assertions définissent l'oracle d'un test, exemple :
 - `assertEquals(6, nb.multiplier(3));`

4 – Test Nominal VS Test de Robustesse *Test aux Limites*

Test dynamique avec JUnit

On doit tester différents aspects :

- Le bon fonctionnement d'une opération en mode nominal
 - *On peut avoir plusieurs opérations de tests visant à couvrir l'ensemble des fonctionnalités*
- La bonne gestion des cas non standards :
 - *Valeurs non valides (erreur correctement gérée)*
 - *Très élevées*
 - *Bornes des valeurs*
 - *Etc.*

Question :
Résultat
du test?

```
double x = 9999999999999999.;
double y = x + 1.0;
assertTrue(y - x == 1.0);
```

Pass



Fail



```
double x = 9999999999999999.;
double y = x + 1.0;
assertTrue(y - x == 1.0);
```

- Attention avec l'égalité entre nombres flottants
 - Pas que en Java!
 - Car c'est un problème de la norme IEEE 754 implémentée par les processeurs
- Java et JUnit
 - `v1==v2 => Double.compare(v1, v2) == 0`
 - `assertEquals(x, y, 0.0001);`

Test dynamique avec JUnit

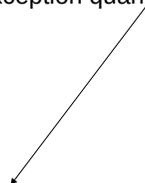
```
public class Nombre {
    double valeur;
```

```
    public Nombre(int val) {
        valeur = val;
    }
```

```
    public double multiplier(double val) {
        return valeur*val;
    }
```

```
    public double diviser(double v) throws IllegalArgumentException{
        if(Double.compare(v, 0.0) == 0)
            throw new IllegalArgumentException();
        return valeur/v;
    }
}
```

Exception quand divise par 0



Rappel exception Java

Exception = Erreur gérée dans le code
Le 'throws' déclare les exceptions lancées par l'opération

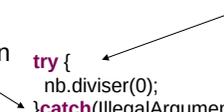
```
public double diviser(double v) throws IllegalArgumentException {
    if(Double.compare(val, 0.0)==0)
        throw new IllegalArgumentException();
    return valeur/val;
}
```

Le 'throw' lance une exception

'try' permet d'encadrer du code pouvant lever une exception

'catch' définit le code à exécuter en cas d'exception levée

```
try {
    nb.diviser(0);
} catch (IllegalArgumentException e) {
    e.printStackTrace();
}
```



Test dynamique avec JUnit

```
public class TestNombre {
    @Test
    void testMultiplier() {
        Nombre nb = new Nombre(2);
        assertEquals(6.0, nb.multiplier(3.0), 0.0001);
    }
```

Plusieurs tests par classe

1 test = 1 scénario

```
    @Test
    void testDiviser() {
        Nombre nb = new Nombre(6);
        assertEquals(2.0, nb.diviser(3.0), 0.0001);
    }
```

Teste que division ok



```
    @Test
    void testDiviserPar0() throws IllegalArgumentException {
        Nombre nb = new Nombre(2);
        assertThrows(IllegalArgumentException.class, () -> nb.diviser(0));
    }
}
```

Teste que division par 0 lève une exception



Test dynamique avec JUnit

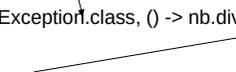
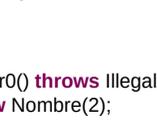
JUnit permet de tester que des erreurs sont bien gérées

'assertThrows' permet de définir l'exception qui doit être levée

puis on donne l'exception

```
@Test
void testDiviserPar0() throws IllegalArgumentException {
    Nombre nb = new Nombre(2);
    assertThrows(IllegalArgumentException.class, () -> nb.diviser(0));
}
```

On provoque l'erreur



4 – Préludes et Postludes de Tests

Setup & Tear Down

Test dynamique avec JUnit

```
public class TestNombre {
    @Test
    void testMultiplier() {
        Nombre nb = new Nombre(6);
        assertEquals(2.0, nb.multiplier(12.0), 0.01);
    }

    @Test
    void testDiviser() {
        Nombre nb = new Nombre(6);
        assertEquals(2.0, nb.diviser(3.0), 0.01);
    }

    @Test
    void testDiviserPar0() throws IllegalArgumentException {
        Nombre nb = new Nombre(6);
        assertThrows(IllegalArgumentException.class, () -> nb.diviser(0));
    }
}
```

Souvent, initialisation identique dans différents tests : **Redondant**

ISTIC - L2GEN

Test dynamique avec JUnit

Test Fixture

```
public class TestNombre {
    Nombre nb;

    @Before
    void setUp() {
        nb = new Nombre(6);
    }

    @Test
    void testMultiplier() {
        assertEquals(18., nb.multiplier(3.), 0.);
    }

    @Test
    void testDiviser() {
        assertEquals(2., nb.diviser(3.), 0.);
    }

    @Test
    void testDiviserPar0() throws IllegalArgumentException {
        assertThrows(IllegalArgumentException.class, () -> nb.diviser(0));
    }
}
```

Déclaration d'1 attribut *nb*

L'annotation *@Before* permet de définir 1 opération appelée avant chaque test: **test fixture** Utilisée pour initialiser des valeurs.

Initialisation de *nb*

Utilisation de *nb*

ISTIC - L2GEN

Tear Down

- Action à faire après chaque test
 - Par exemple relâcher une ressource, fermer un fichier...

```
@After
public void tearDown() {
    System.out.println("This is the tearDown() method that runs after each testcase");
}
```

Test dynamique

- JUnit permet de d'écrire différents types de tests dynamiques
 - Les slides précédents faisaient référence à du test unitaire
 - Test d'intégration
 - Test système (E2E)
 - Test d'acceptation
 - Test de robustesse
 - Test de performance
 - Test de non régression
 - Archiver et automatiser les tests
 - ...
- Mais ce sera pour plus tard...



33

5 - TDD : Test Driven Design

Développement dirigé par les tests

Principe du TDD

- Pour ajouter une fonctionnalité X à un logiciel:
 1. Ecrire les tests T montrant que X fonctionne
 2. Vérifier que le test échoue (X n'est pas encore développée) => Permet de vérifier que le test est valide!
 3. Ecrire le code nécessaire pour passer T (et pas plus)
 4. Vérifier que les tests T passent, sinon retourner en 3
 5. Si nécessaire, remanier le code pour le simplifier

ISTIC - L2GEN

35

Exercice TDD

- Ecrire les tests pour une fonction **add(s:String):Int** qui additionne les nombres contenus dans une chaîne de caractères.
 - pour une chaîne vide, la fonction rend 0
 - pour une chaîne contenant un seul nombre, la fonction rend ce nombre
 - pour une chaîne contenant deux nombres séparés par une virgule, la fonction rend la somme
 - etc.

```
@test
public void testAdd0() {
    //...
}
@test
public void testAdd1() {
    //...
}
```

ISTIC - L2GEN

36

TDD : Conclusion

- **Tests Simples et Rapides**

- L'accent est mis sur la création de tests simples et rapides pour maintenir une suite de tests efficace.

- **Communication Améliorée**

- Les tests servent également de documentation vivante, améliorant la communication au sein de l'équipe de développement.

- **Inconvénients :**

- Peut être couteux si fait sans discernement
- Peut entraîner une confiance excessive dans la stabilité du logiciel
 - certains problèmes peuvent subsister au niveau des interactions entre composants.

Vous le pratiquerez au prochain TP!



Pr. Jean-Marc Jézéquel
IRISA – Université de Rennes
Campus de Beaulieu
F-35042 Rennes Cedex
e-mail : jezequel@irisa.fr
<http://people.irisa.fr/Jean-Marc.Jezequel>
X @jmjezequel

1 - Introduction



CommitStrip.com

Qualité du process vs qualité du produit

- Qualité du process de fabrication
 - Ensemble de normes ISO 9001
 - Capability Maturity Model (CMM)
 - Etc.
- Qualité du produit = qualité du logiciel
 - Facteurs contribuant à la qualité du logiciel

Facteurs de qualité externes d'un logiciel (1/3)

- Validité
 - Aptitude à réaliser les tâches définies par sa spécification
Spécifications informelles => objectif difficile à atteindre
- Robustesse
 - Aptitude à fonctionner même dans des conditions anormales
Réaction non catastrophique à des situations non prévues par la spécification, forcément incomplète
- Extensibilité
 - Facilité d'adaptation d'un logiciel aux changements (correctifs ou évolutifs) de spécification
éviter les logiciels château de cartes

Facteurs de qualité externes d'un logiciel (2/3)

- Réutilisabilité
 - Aptitude à être réutilisé en partie pour de nouvelles applications
Cesser de réinventer, re-coder, et surtout, re-tester
- Compatibilité
 - Aptitude des logiciels à pouvoir être combinés entre eux
Assemblage de briques de base
- Efficacité
 - Bonne utilisation des ressources du matériel: processeurs, mémoires, communications, énergie, etc.
A ne pas confondre avec temps réel...

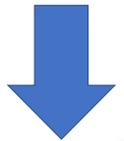
Facteurs de qualité externes d'un logiciel (3/3)

- Portabilité
 - Facilité avec laquelle un produit logiciel peut-être adapté à différents environnements matériels et logiciels
 - *souvent incompatible avec efficacité optimale...*
- Vérifiabilité
 - Facilité de préparation des procédures de validation (tests), de recette et de certification
- Ergonomie
 - Facilité avec laquelle les utilisateurs d'un composant peuvent apprendre à l'utiliser et à en tirer le meilleur part
documentation à jour (=> extraite du code : Javadoc)

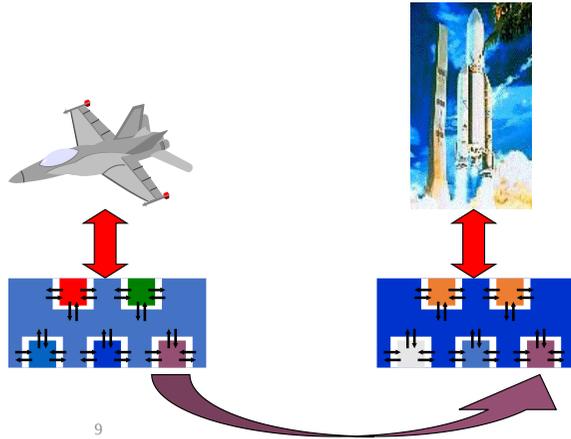
2 – Conception par Contrat

Principe de l'approche objet

- Structurer les systèmes autour des objets
 - Plutôt qu'autour des fonctions
- Obtenir des systèmes modulaires et maintenables
 - Par assemblage de briques de base
 - Composition d'éléments simples pour obtenir des systèmes sophistiqués
- Favorise :
 - masquage d'information (abstraction)
 - encapsulation (facilite les modifications à portée locale)
 - dissociation interface/implantation=> composant réutilisab



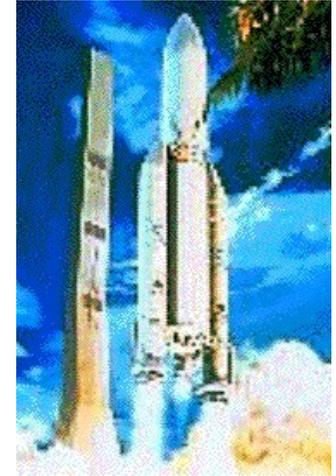
Validité inter-composants : Peut-on (ré)-utiliser un composant?



9

Ariane 501 Vol de qualification Kourou, ELA3 -- 4 Juin 1996,12:34 UT

- H0 -> H0+37s : nominal
- Dans SRI 2:
 - BH (Bias Horizontal) > 2¹⁵
 - convert_double_to_int(BH) fails!
 - exception SRI -> crash SRI2 & 1
- OBC disoriented
 - Angle attaque > 20°,
 - charges aérodynamiques élevées
 - Séparation des boosters



10

Ariane 501 : Vol de qualification Kourou, ELA3 -- 4 Juin 1996,12:34 UT

- H0 + 39s: auto-destruction (coût: 500M€)



Pourquoi ? (cf. IEEE Comp. 01/97)

- Pas une erreur de programmation
 - Non-protection de la conversion = décision de conception ~1980
- Pas une erreur de conception
 - Décision justifiée vs. trajectoire Ariane 4 et contraintes TR
- Problème au niveau du test d'intégration
 - Comme toujours, aurait pu être détecté...
 - Mais gigantesque espace de test vs. ressources limitées
 - En plus, SRI inutile à cette étape du vol!

Pourquoi? (cf. IEEE Computer 01/97)

- Réutilisation dans Ariane 5 d'un composant de Ariane 4 ayant une contrainte « cachée » !
 - Restriction du domaine de définition
 - Précondition : $\text{abs}(\text{BH}) < 32768.0$
 - Valide pour Ariane 4, mais plus pour Ariane 5

Spécification = contrat entre un composant et ses clients

- Dans la vie réelle, différents types de contrats
 - Du « Contrat social » de Jean-Jacques Rousseau au “cash & carry”
- De même, plusieurs types de contrats dans un monde réparti



Quatre niveaux de contrats logiciels

- Élémentaire (syntaxique)
 - le programme compile...
- Comportemental (fonctionnel)
 - pré et post conditions
- Synchronisations
 - e.g. *path expressions*, etc. [McHale]
- Qualité de service (quantitative)
 - Négociation dynamique possible

Cf. IEEE Computer
July 1999

Représentation des contrats en UML avec OCL (Object Constraint Language)

- Typage par signature des méthodes insuffisant
 - besoin de pouvoir exprimer des restrictions
 - valeurs d'entrées et de sorties
 - besoin de préciser la sémantique
 - ce que fait une méthode (le quoi) sans entrer dans le détail comment
 - Préserve le masquage d'information : Indépendance vis-à-vis de l'implantation
- Inspirée par la notion de Type Abstrait de Données:
Spécification = Signature +
 - Préconditions (conditions sous lesquelles une méthode peut être appelée)
 - Postconditions (propriétés garanties par une méthode)
 - Invariants de classe (vrai à l'entrée et à la sortie des méthodes)

- Directement dans le modèle
 - notation entre { } **accrochée à un élément de modèle**



- Dans un document séparé, en précisant le **contexte**
 - Invariants = Propriétés vraies pour l'ensemble des instances de la classe
 - dans un état stable, chaque instance doit vérifier les invariants de sa classe

contexte Compte inv:
solde >= plancher

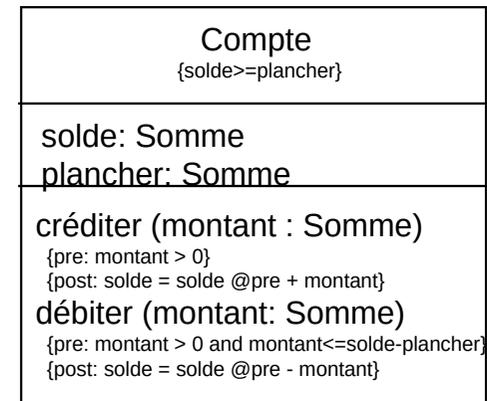
Précondition: *ce qui doit être respecté par le client*

- Spécification des conditions nécessaires pour qu'un client soit autorisé à appeler une méthode
 - exemple: montant > 0
- Notation en UML
 - {«precondition» OCL boolean expression}
 - Abbreviation: {pre: OCL boolean expression}

Postcondition: *Ce qui doit être assuré par l'implantation*

- Spécification de ce qui sera vrai à la complétion d'un appel valide à une méthode
 - exemple: solde = solde @pre + montant
- Notation en UML
 - {«postcondition» OCL boolean expression}
 - Abbreviation: {post: OCL boolean expression}
 - Opérateur pour accéder à la valeur « d'avant »:
 - OCL expression @pre

Etre abstrait et précis avec UML



Conception par Contrat

- Assertions des contrats :
 - jouent un rôle crucial dans la séparation nette des responsabilités dans un système modulaire
 - contrat entre l'appelant d'une méthode (le client) et l'implantation de la méthode (le contractant) :

Pourvu que le client appelle la méthode dans des conditions où l'invariant de classe du contractant et la précondition de la méthode sont respectés, alors le contractant promet que lorsque la méthode terminera, le travail spécifié dans la postcondition sera effectué, et l'invariant de classe sera respecté.

Contrats en Java

- Pas de support direct en Java
 - Support direct en Eiffel, Scala, C#/Spec#
 - Support par annotations en Python (PyContracts), Kotlin, JavaScript
- Pour la **spécification** : dans le Javadoc
- Pour l'**implantation** : utilisation de *assert* et *exceptions*
- Pour aller plus loin : Java Modeling Language (JML)
 - Cf cours MEF de Sandrine Blazy

Contrats en Java : Spécification en Javadoc

```

/**
 * ...
 * @param montant : quantité à créditer sur le compte
 * @requires
 *   montant > 0
 * @ensures
 *   solde = solde@pre + montant
 */
public void créditer(int montant) {
}
    
```

← précondition

← postcondition

Compte <small>(solde=plancher)</small>
solde: Somme plancher: Somme
créditer (montant: Somme) <small>(pre: montant > 0)</small> <small>(post: solde = solde @pre + montant)</small> débiter (montant: Somme) <small>(pre: montant > 0 and montant <= solde - plancher)</small> <small>(post: solde = solde @pre - montant)</small>

Contrats en Java : Implantation

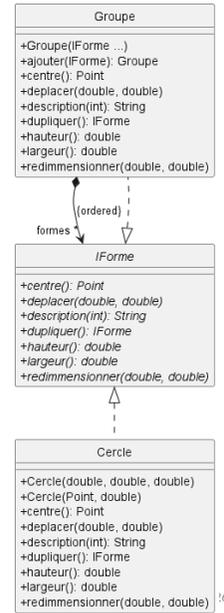
- Invariant
 - Faire une méthode : `boolean invariant(){}`
 - Augmentable dans les sous-classes par redéfinition avec appel à `super.invariant();`
- Précondition
 - Lancer une exception. Par ex. pour `créditer(int montant):`
 - `if (montant < 0) throw new IllegalArgumentException("montant < 0");`
- Post-condition
 - `assert invariant() && solde == oldSolde + montant : "Postcondition : solde incorrect";`
 - Simulation du `@pre` en utilisant une variable locale
- **Limitation** : héritage à la main
 - Si on redéfinit une méthode, il faut ré-écrire son contrat

Intérêt pratique des contrats

- Spécification, documentation
 - *Pas un outil pour la tolerance aux fautes*
- Aide au Test
 - Les post-conditions jouent le role des **Oracles** dans les tests unitaires
 - Usage systématique => plus jamais de debugging
- Permet d'attribuer facilement la responsabilité d'un bug lors de l'integration
 - Précondition violée : problème dans le code du client
 - Postcondition violée : problème dans le code du fournisseur

Exercice

- Contrat de
 - `IForme.deplacer()`?
 - `IForme.redimensionner(dx,dy)`?
 - Quid si $dx \leq 0$?
 - Quid des figures symétriques comme Cercle ou Losange si $dx \neq dy$?
 - `IForme.dupliquer()`



3 – Qualité des tests

Quand s'arrêter d'écrire des tests ?
 Quels sont les différents types de tests unitaires ?
 Quelles données utilisées ?

De la difficulté de la validation intra-composant

Acquérir une valeur positive n
 Tant que $n > 1$ faire
 si n est pair
 alors $n := n / 2$
 sinon $n := 3n + 1$
 Sonner alarme;

- Prouver que l'alarme est sonnée pour tout n?
 - Revient à prouver que la suite $n_{i+1} = n_i \text{ pair} ? n_i / 2 : 3n_i + 1$ converge
 - C'est la conjecture de Syracuse...
- Indécidabilité de certaines propriétés
 - problème de l'arrêt de la machine de Turing...

- **Recours au test**
 - ici, si machine 32 bits, $2^{31} = 10^{10}$ cas de tests
 - **5 lignes de code => 10 milliards de tests !**

Couverture de code

- Quand s'arrêter d'écrire des tests ?
- Couverture de code par les tests :
 - méthode d'aide pour savoir quels tests écrire (quels scénarios)
 - Critère de qualité des tests
 - Plusieurs stratégies : les critères de couverture
 - Couverture des lignes de code
 - Couverture des instructions
 - Couverture de branches
 - Couverture de conditions

Couverture de code

- Critères de couverture : couverture des lignes de code
 - Proportion des lignes de code exécutées lors d'un test
 - $n = 0 \Rightarrow 40\%$
 - $n = 4 \Rightarrow 80\%$
 - $n = 7 \Rightarrow 100\%$
- Objectif
 - toutes les lignes de code sont couvertes par au moins un test
- Problème de ce critère : ni nécessaire, ni suffisant

Acquérir une valeur positive n
 Tant que $n > 1$ faire
 si n est pair
 alors $n := n / 2$
 sinon $n := 3n + 1$

Limite de la couverture des instructions

- L'unique test couvre le code de la méthode *multiplier*
 - ... mais ne vérifie presque rien (sauf d'éventuelles exceptions)
 - => les assertions sont nécessaires pour vérifier des propriétés
- => **couverture de 100 % ne veut pas dire code sûr**
- Il existe des critères de couverture plus sophistiqués, mais toujours sans garantie sur le code
 - Couverture des instructions
 - Couverture de branches
 - Couverture de conditions

```
public double multiplier(double val) {
    return valeur*val;
}

public class TestNombre {
    Nombre nb;

    @BeforeEach
    void setUp() {
        nb = new Nombre(2);
    }

    @Test
    void testToutPourri() {
        nb.multiplier(0);
    }
}
```

Données de test : les classes d'équivalence

- Partitionner les données d'entrée en classes d'équivalence
 - telles que pour chaque classe, le programme a le même comportement
 - $n \leq 0$
 - $n = 2^k$
 - $n = 2^k + 1$
 - Valeurs extrêmes :
 - Integer.MAX_VALUE, Integer.MIN_VALUE
- Donc pas nécessaire d'écrire **tous** les tests
 - Pour chaque classe, prendre une valeur représentative et écrire un test avec
 - $n = 0$
 - $n = 4$
 - $n = 7$
 - ...

Acquérir une valeur positive n
 Tant que $n > 1$ faire
 si n est pair
 alors $n := n / 2$
 sinon $n := 3n + 1$

4 – Problème de l’oracle

Metamorphic testing & Snapshot testing

Tester un diagramme Camembert

```
public class CamembertTest {
    static final double delta = 0.000001;
    Camembert c;

    @Before
    public void setUp() {
        c = new Camembert(110,110, 100);
    }

    @Test
    public void testAjouterSecteur() {
        c.ajouterSecteur("red", 0.15);
        assertEquals(0.15*360, c.getArc(),delta);
        c.ajouterSecteur("blue", 0.2);
        assertEquals(0.35*360, c.getArc(),delta);
        c.ajouterSecteur("green", 0.65);
        assertEquals(360.0, c.getArc(),delta);
    }

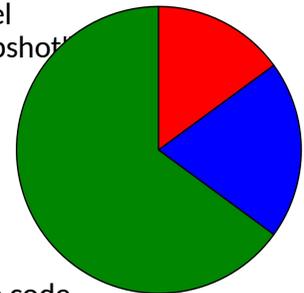
    @Test
    public void testEnSVG() { // Oracle ????
```

Problème de l’oracle

- Parfois l’oracle est difficile à spécifier
 - Données complexes
 - Etat de l’objet testé « opaque » (ie non facilement accessible)
- 2 possibilités
 - Snapshot testing
 - Metamorphic testing

Snapshot testing

- Technique de test logiciel où le comportement actuel d'une application est enregistré sous forme de "snapshot" et comparé aux futures exécutions
 - Executer testAjouterSecteur(), sauver le résultat
 - Vérifier visuellement que c’est OK
 - Copier manuellement le svg dans
 - Public static final String camembertSVG = ...;
- Les variations inattendues dans le snapshot peuvent indiquer des changements non intentionnels dans le code source, aidant ainsi à détecter les régressions potentielles au fil du développement.



```
@Test
public void testEnSVG() {
    testAjouterSecteur();
    assertEquals(camembertSVG, c.svg());
}
```

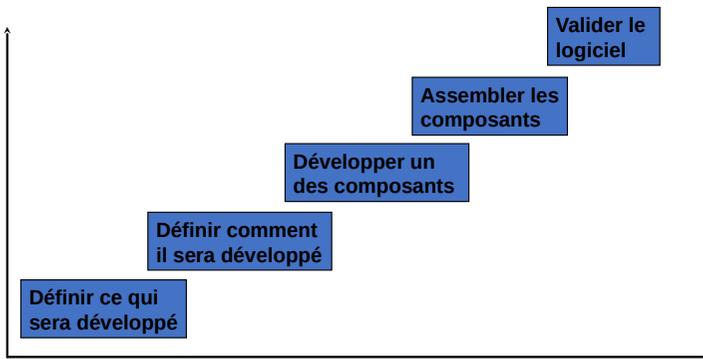
Test Métamorphique

- **Approche de Test** : Évaluation du comportement du système en fonction de transformations spécifiques appliquées à ses entrées.
- **Évaluation des Transformations** : Examine comment les sorties évoluent avec des transformations définies telles que modifications, perturbations, ou permutations des données d'entrée.
- **Objectif** : Identifier des comportements indésirables dans des situations où les résultats sont difficiles à prédire, mais où les relations entre les données sont mieux comprises.
- Exemple
 - `forme.deplacer(30,20).deplacer(-30,-20)` doit remettre l'objet forme dans son état initial

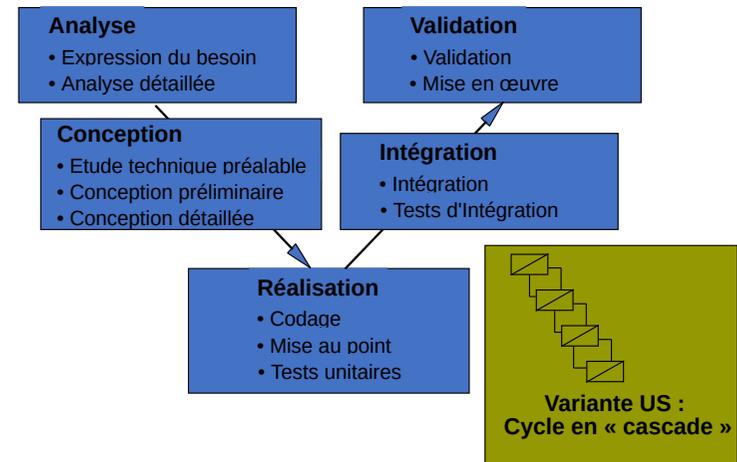


Pr. Jean-Marc Jézéquel
IRISA – Université de Rennes
Campus de Beaulieu
F-35042 Rennes Cedex
e-mail : jezequel@irisa.fr
<http://people.irisa.fr/Jean-Marc.Jezequel>
X @jmjezequel

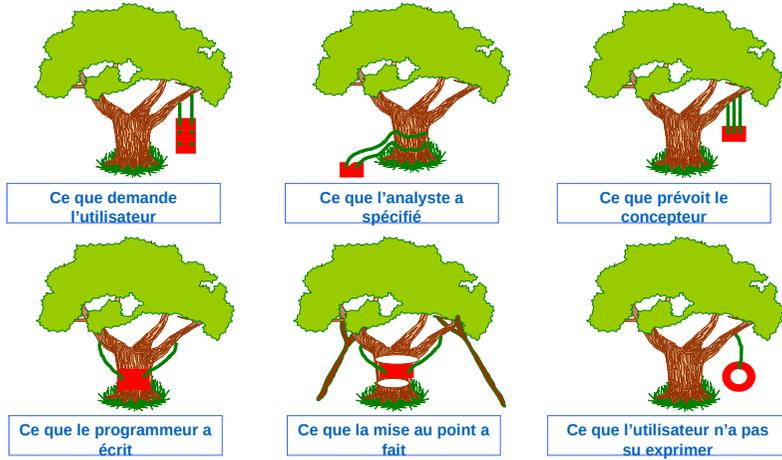
1 - Introduction



- L'organisation de ces activités et leur enchaînement définit le *cycle de développement* du logiciel



Problèmes avec le processus classique...



Problèmes du processus classique

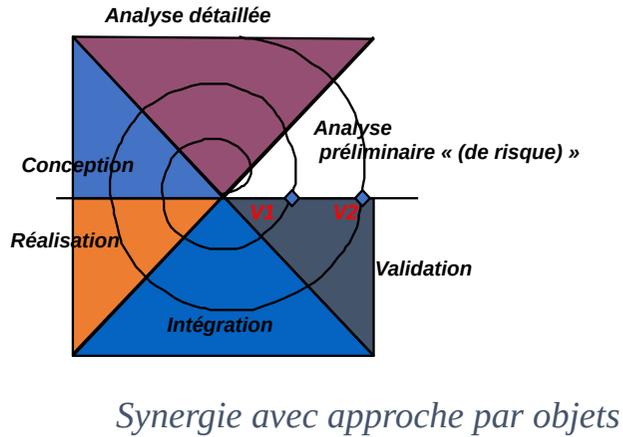
- Organisation « industrielle » héritée du XIX^{ème} siècle
 - rassurant pour les managers
 - hiérarchie malsaine dans les rôles
 - antinomie : Coplien 's organizational pattern
 - *Architects Also Implement*
- cycle management <> cycle développement
- linéarité implicite
 - temps d 'approbation des documents => effet tampon
 - coût de la (non-) modification d 'un document « final »
 - irréaliste pour un projet innovant, donc à risques

Causes principales d'échecs des projets

- Des objectifs de projet irréalistes ou non articulés
- Estimation inexacte des ressources nécessaires
- Des exigences système mal définies
- Mauvaise communication de l'état d'avancement du projet
- Risques non gérés
- Mauvaise communication entre les clients, les développeurs et les utilisateurs
- Utilisation d'une technologie immature
- Incapacité à gérer la complexité du projet
- Des pratiques de développement peu rigoureuses
- Mauvaise gestion des projets
- Jeux politiques des parties prenantes
- Pressions commerciales

2 – Processus itératifs

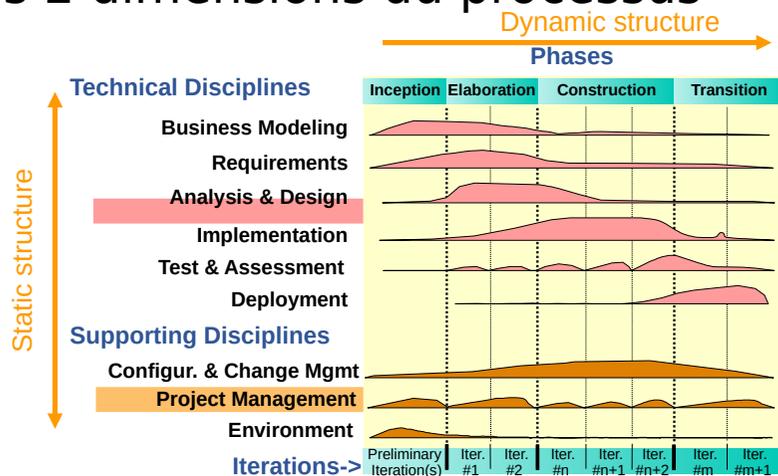
Barry Boehm, 1985



Intérêts du cycle de vie en « spirale »

- Bien adapté au développements innovants
 - les progrès sont tangibles : c'est du logiciel qui « tourne » et pas seulement des kilos de documents
 - possibilité de s'arrêter « à temps », i.e. avant que l'irréalisabilité du projet ait créée un gouffre financier
- Moins simple à manager
 - difficile à gérer en situation contractuelle
 - mal contrôlé => on retombe dans le *hacking*
- Production des incréments asservie sur 2 parmi 3 :
 - période (e.g. release toutes les 2 semaines)
 - fonctionnalités (releases découpés suivant use-cases)
 - niveau de qualité (problème de la mesure)

Les 2 dimensions du processus



Phases du développement itératif



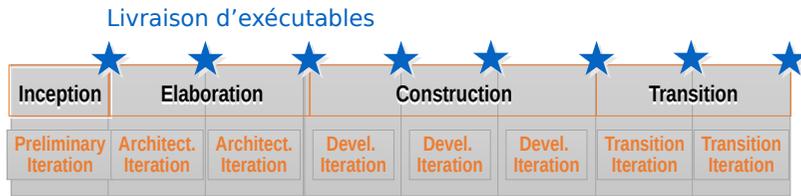
Inception: définition de la portée du projet

Elaboration: planification du projet, spécification des fonctionnalités, architecture de base

Construction: réalisation du produit

Transition: transfert du produit vers les utilisateurs

Itérations



Une **itération** est une séquence d'activités avec un plan bien établi et un critère d'évaluation, résultant en la livraison d'un logiciel *exécutable*.

Phases et itérations : 2 exemples

- Petit projet de commerce électronique
 - Intégration à un mainframe
 - 5 personnes
- Grand projet d'infrastructure
 - Gros travail d'architecture nécessaire
 - 20 personnes

	No. of Iterations				Project Length	Iteration Length
	Inception	Elaboration	Construction	Transition		
e-business	0.2	1	3	1	3-4 months	2-3 weeks
infrastructure	1	3	3	2	9-12 months	5-7 weeks

3 – Méthodes agiles

Exemple de Scrum

Points clés de l'Agilité

- Développer seulement ce qui est nécessaire
- Minimiser la paperasserie
- flexibilité
 - besoins, plan, utilisation des ressources, etc...
- Apprendre de ses erreurs précédentes
- Réévaluer les risques régulièrement
- Établir des critères de progrès
 - objectifs et mesurables
- Automatiser

Scrum

- Scrum est une **méthode agile** pour la gestion **itérative** de projet
 - exploitant la créativité des membres de l'équipe de développement.
- On peut distinguer trois phases :
 - La phase initiale au cours de laquelle les fonctionnalités du système sont listées et une architecture logicielle générale est définie
 - Suit une série de "sprints", chaque sprint correspondant à un incrément du système
 - La phase de terminaison du projet développe les derniers artefacts (manuel d'utilisation ...) et tire les leçons apprises durant le développement.

Terminologie Scrum (1/2)

Terme Scrum	Definition
Équipe de développement	Un groupe auto-organisé de développeurs de logiciels, qui ne devrait pas dépasser 7-10 personnes, responsables du développement du logiciel et d'autres documents essentiels du projet.
Incrément de produit potentiellement livrable	Incrément logiciel fourni à partir d'un sprint. « potentiellement livrable » signifie qu'il est dans un état fini et qu'aucun travail supplémentaire, tel que des tests, n'est nécessaire pour l'incorporer dans le produit final.
Product backlog	Liste d'éléments à faire auxquels l'équipe Scrum doit s'attaquer. Il peut s'agir de définitions de fonctionnalités pour le logiciel, d'exigences logicielles, de récits utilisateur (<i>user stories</i>) ou de descriptions de tâches supplémentaires nécessaires, telles que la définition de l'architecture ou la documentation utilisateur.
Product owner	Une personne (ou éventuellement un petit groupe) dont le travail consiste à identifier les caractéristiques ou les exigences du produit, à les hiérarchiser pour le développement et à examiner en permanence le backlog du produit pour s'assurer que le projet continue de répondre aux besoins critiques de l'entreprise. Le Product Owner peut être un client, mais aussi un chef de produit dans une société de logiciels ou un autre représentant des parties prenantes.

Terminologie Scrum (2/2)

Terme Scrum	Definition
Scrum	Une réunion quotidienne de l'équipe Scrum qui passe en revue les progrès et hiérarchise le travail à faire ce jour-là. Idéalement, il devrait s'agir d'une courte réunion en face à face qui inclut toute l'équipe.
Scrum Master	Le Scrum Master est chargé de s'assurer que le processus Scrum est suivi et guide l'équipe dans l'utilisation efficace de Scrum. Il est responsable de l'interface avec le reste de l'entreprise et de s'assurer que l'équipe Scrum n'est pas détournée par des interférences extérieures. Les développeurs Scrum sont catégoriques sur le fait que le Scrum Master ne doit pas être considéré comme un chef de projet. D'autres, cependant, ne trouvent pas toujours facile de voir la différence.
Sprint	Une itération de développement. Les sprints durent généralement de 2 à 4 semaines.
Velocity	Estimation de l'effort de backlog produit qu'une équipe peut couvrir en un seul sprint. Comprendre la vitesse d'une équipe l'aide à estimer ce qui peut être couvert dans un sprint et fournit une base pour mesurer l'amélioration des performances.

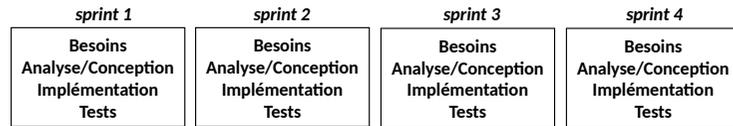
Processus logiciel SCRUM

- Une phase d'idéation
- Une succession de sprints de taille fixe
- Chaque *sprint* produit un logiciel fonctionnel
- Chaque *sprint* réalise toutes les activités de développement logiciel
- Les *sprints* sont groupés en *Releases* (de taille fixe)

Processus logiciel SCRUM (*sprints* et *releases*)

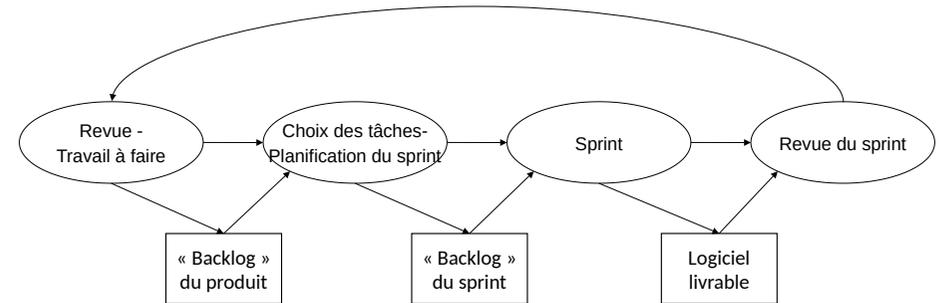


Cycle séquentiel



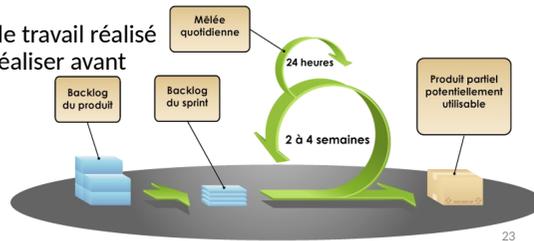
Cycle SCRUM d'une release

Cycle de vie d'un sprint SCRUM



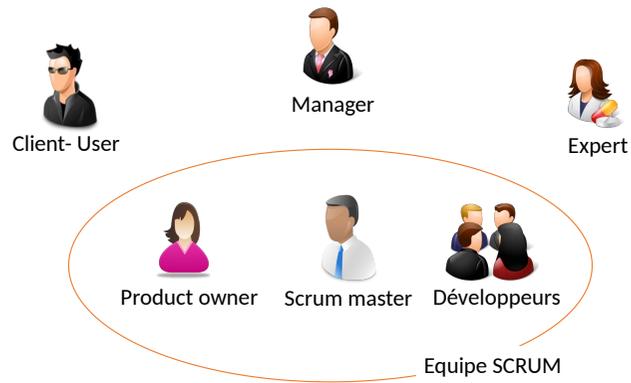
Sprint

- Chaque release est décomposée en *sprints* au cours de l'activité de planification de la release (la première fois « sprint zéro »)
- Un sprint est une période de développement de taille fixe
 - Durée fixe (en moyenne 2 à 4 semaines), équipe stable
- Un sprint se décompose en un ensemble de tâches
 - **Backlog**: carnet, liste ordonnée de « choses » à faire (*user stories*, tâches)
- Mêlée journalière (Scrum)
 - réunion d'équipe pour faire le point sur le travail réalisé depuis le début du sprint et le travail à réaliser avant la fin du sprint
- Chaque sprint termine par :
 - une revue du produit
 - une rétrospective sur le processus



4 - Les acteurs de Scrum

Rôles



L'équipe

- Composition:
 - 1 Product Owner
 - 1 Scrum Master
 - 2 à 7 développeurs
 - Le product owner et le Scrum master peuvent aussi prendre le rôle de développeur
- Principes
 - Auto-organisation
 - Pluridisciplinarité
 - Stabilité
 - Valeurs communes

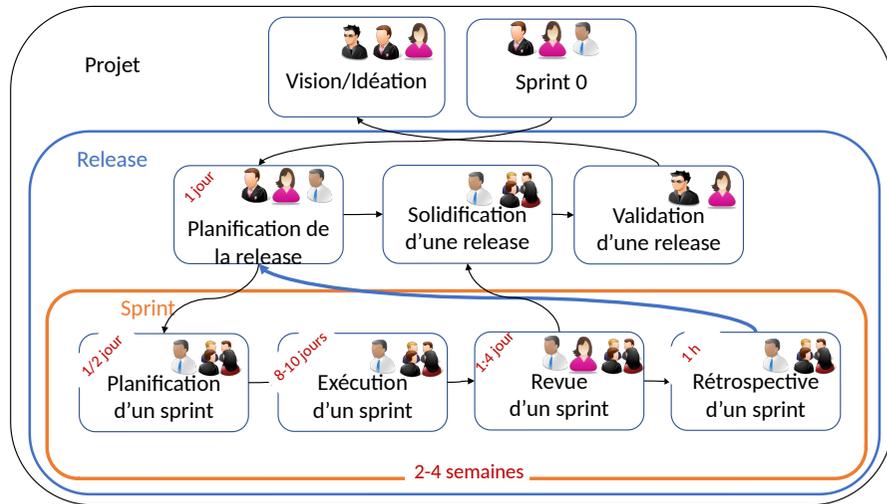
Product owner

- Responsabilités
 - Fait partager la vision globale du produit
 - Gère le backlog du produit (liste ordonnée des « choses » à faire)
 - Définit les priorités
 - Accepte ou rejette les *Releases* (livrables)

Scrum master

- Responsabilités
 - n'est pas « le chef », mais un facilitateur
 - Motive l'équipe
 - Fait appliquer les bonnes pratiques de Scrum
 - Gère les obstacles

Processus logiciel Scrum



ISTIC- L2GEN



Client- User



Manager



Product owner



Scrum master



Développeur

29

5 - Les objets de Scrum

Feature, Story, Tâche, Backlog

Feature

- Une *feature* (*fonctionnalité*) est un service ou une fonctionnalité du produit à développer
- Elle se décompose en *stories* (histoires) de tailles différentes. On distingue :
 - les *stories* complexes (épopées) qui seront affinées en *stories* plus simples
 - Les *stories* atomiques qui ne se décomposent pas et sont réalisées dans les *sprints*

ISTIC- L2GEN

31

Story

- Une *story* est une exigence du système à développer, formulée en une ou deux phrases dans le langage de l'utilisateur.
- Les *Stories* émergent au cours d'ateliers de travail menés avec le Métier, le Client et/ou les Utilisateurs.
- On distingue :
 - *Story* fonctionnelle
 - *Story* technique
 - Correction de bug
 - Remboursement de la « dette technique »

ISTIC- L2GEN

32

Story

- Les 3C
 - Carte : l'histoire est courte et sa description tient sur une carte (demi-page)
 - Conversation : l'histoire est définie avec les gens du métier
 - Confirmation : l'histoire est confirmée par des tests d'acceptation rédigés au même moment que celle-ci
- Workflow de la story
 - Idée d'une story rédigée sur une Carte (1/2 feuille)
 - Conversation dirigée par le *product owner* qui inclut les gens du métier
 - L'équipe apporte sa Confirmation que la story est prête
 - L'équipe réalise la story
 - Le *product owner* apporte sa Confirmation que la story est finie

Description type d'une story

- Plan type
 - **En tant que** <acteur>, **je veux** <un but> [**afin de** <une justification>]
 - **En tant que** client, **je veux** pouvoir accéder au site de ma banque **afin de** gérer mon compte sur Internet
- Priorité
- Nombre de points
 - Représente le niveau de difficulté intrinsèque, en fonction de la taille et de la valeur métier
 - Corrélées en moyenne, mais pas toujours (en fonction d'une forte valeur métier, de la réutilisation possible de composants ...)
- Conditions d'acceptation
 - **Etant donné** <le contexte> **quand je** <événement> **alors** <résultat>
 - **Etant donné** que je suis sur la page de connexion et que j'ai entré un login et un mot de passe dans le formulaire et que le login et le mot de passe correspondent à un utilisateur enregistré, **quand je** clique sur le bouton "Se connecter" **alors** j'arrive sur la page d'accueil du site.

Post-it de story



Tâche

- A l'exécution, une story se décompose en tâches

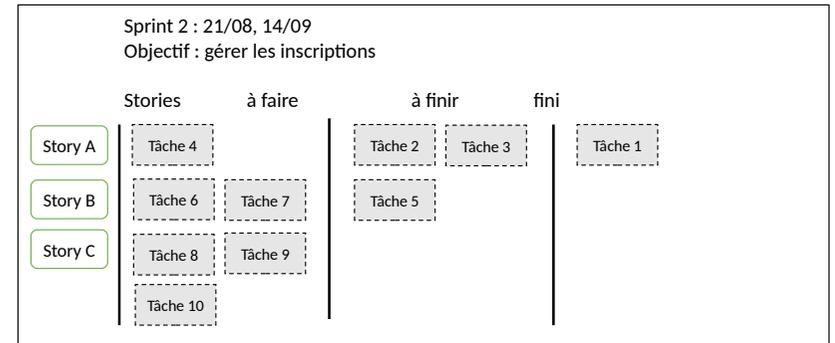
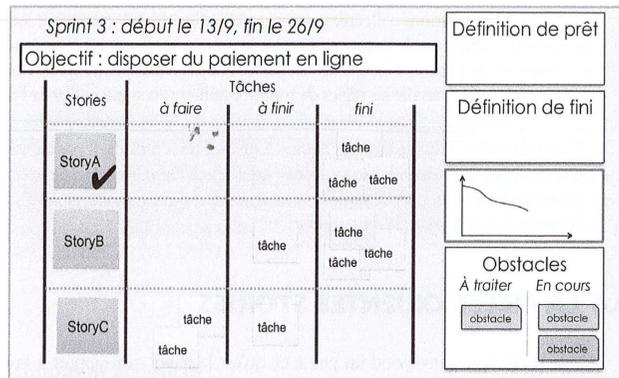


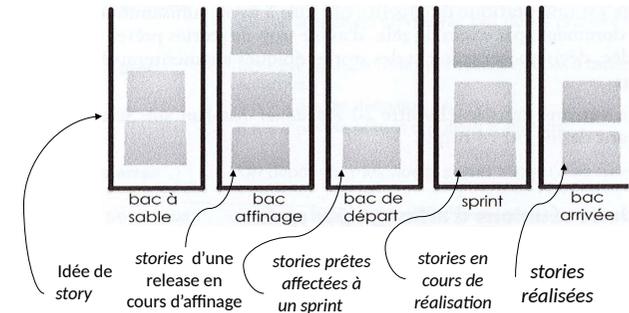
Tableau de la Story

- Le tableau de la story décrit son état.



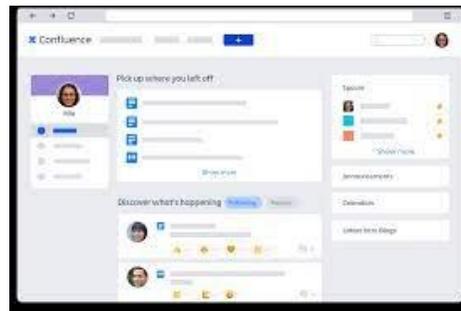
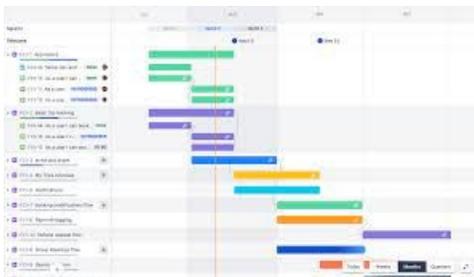
Le backlog (carnet de route)

- Liste ordonnée des choses (stories) à faire
- En pratique, plusieurs (sous-)backlogs
 - On peut distinguer le backlog de produit et le backlog de sprint

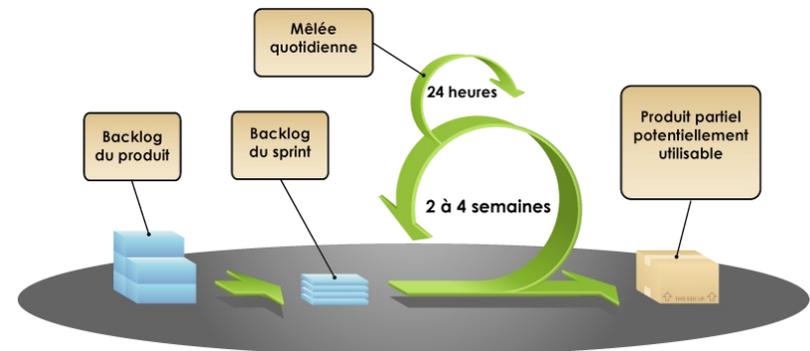


Outils SCRUM

- Jira, Confluence...



Backlog et Sprint

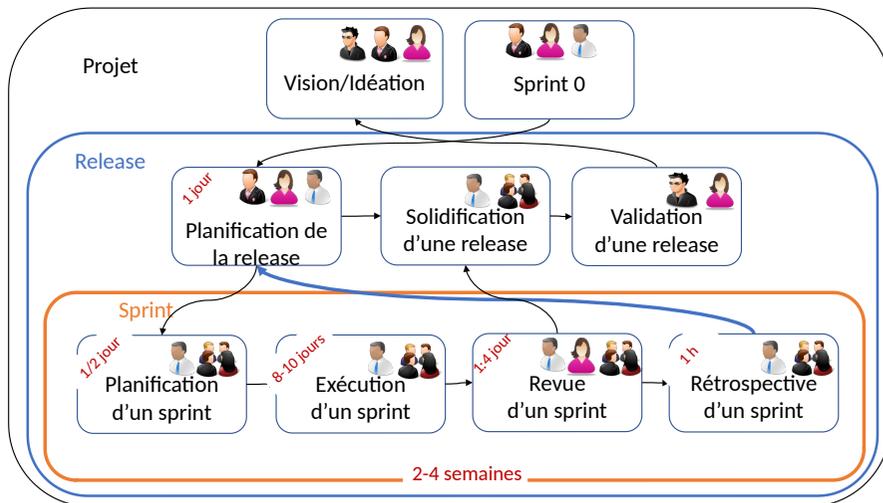


La mêlée quotidienne (Scrum)

- Courte : ~15mn
- Bilan: mise à jour du tableau de la story
 - Qu'est-ce que j'ai fait hier ?
 - Qu'est que je vais faire aujourd'hui ?
 - Quels sont les obstacles que j'ai rencontrés ?
- Objectif :
 - Rythmer le sprint (stories finies, prêtes)
 - Recenser les obstacles

6 – Gestion de projet Scrum

Processus logiciel Scrum



- Client- User
- Manager
- Product owner
- Scrum master
- Développeur

Idéation



- Définition d'une vision commune
 - Identification des *features*
 - Impact mapping
 - Identification des parties prenantes
 - Acteurs
- Création d'un *backlog* de haut niveau du produit
 - Tableau ordonné des features
 - (éventuellement Story map haut niveau)

Impact mapping

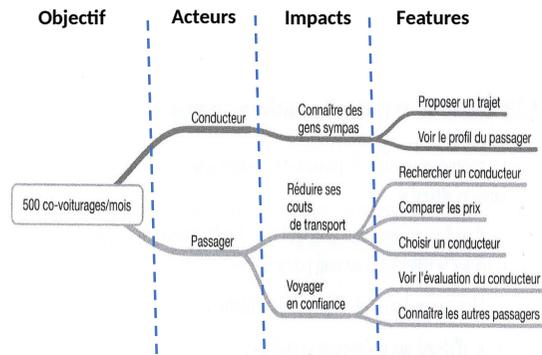


Figure 14.3 – Une carte qui donne la vision pour un site de co-voiturage

Sprint « Zéro »

- Affinage du backlog
 - Story mapping (feature stories)(partiel)
 - Description des stories (description, conditions d'acceptation)
 - Ordonnement des stories
 - Planification de la première release
 - Approvisionnement du bac d'affinage
 - Approvisionnement du bac de départ du sprint 1
- Peut durer plusieurs journées

Workflow d'une feature

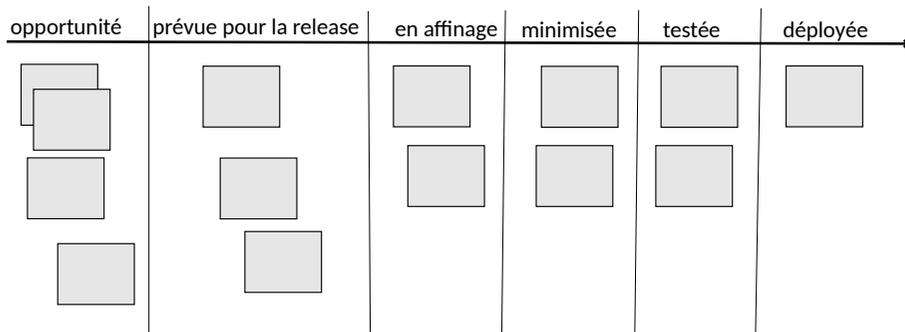


Tableau des features

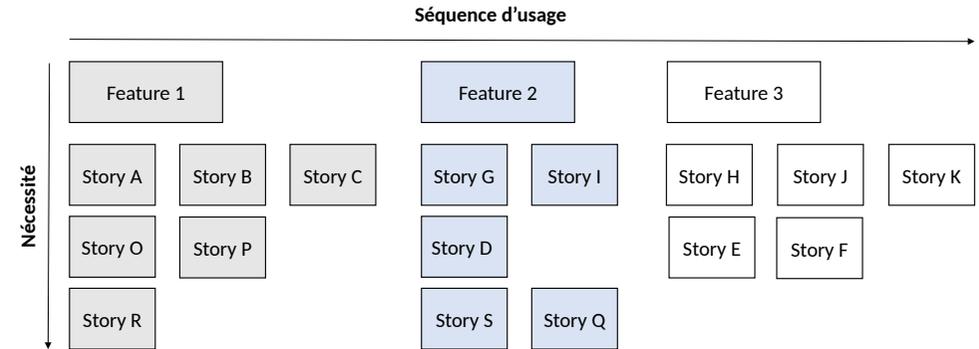
Workflow d'une feature

- Opportunité : idée de feature qui présente une opportunité pour le produit
- Prévue pour la release : l'étude d'opportunité a abouti, et la feature est prévue pour la release en cours
- En affinage : initialement sous forme d'épique, est décomposée en stories
- Minimisée : *Minimal Marketable Feature*
- Testée : ... avec des stories finies
- Déployée : utilisable par des utilisateurs qui peuvent fournir du feedback

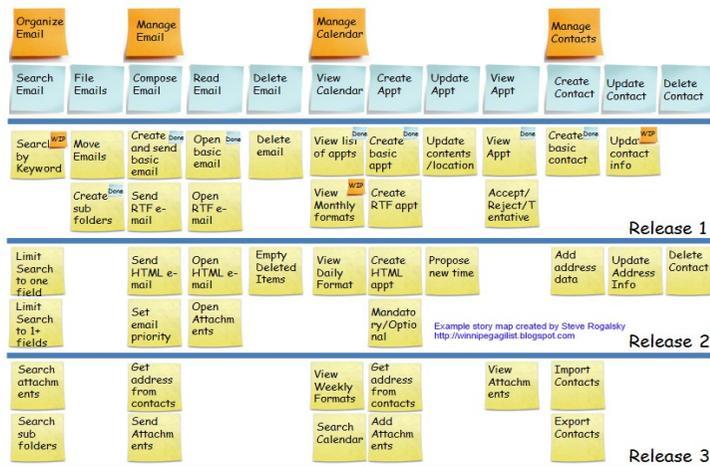
Story mapping

- Ordonnancement des features
- Décomposition en stories
- (Organisation des releases)

Story map : décomposition d'une feature en stories



Exemple



Estimation des points d'une story

- Difficulté intrinsèque
 - Taille et valeur métier
 - Corrélées en moyenne, mais pas toujours (en fonction d'une forte valeur métier, de la ré-utilisation de composants ...)
- Planning Poker (source Wikipedia)
 - Les participants s'installent autour d'une table, placés de façon que tout le monde puisse se voir.
 - Le responsable de produit explique à l'équipe un scénario utilisateur (*user story*).
 - Les participants posent des questions au responsable de produit, discutent du périmètre du scénario, évoquent les conditions de satisfaction qui permettront de le considérer comme "terminé".
 - Chacun des participants évalue la complexité de ce scénario, choisit la carte qui correspond à son estimation et la dépose, face vers le bas, sur la table devant lui.
 - Au signal du facilitateur, les cartes sont retournées en même temps.
 - S'il n'y a pas unanimité, la discussion reprend.
 - On répète le processus d'estimation jusqu'à l'obtention de l'unanimité.
 - Une procédure optimisée consiste, après la première "donne", de demander aux deux acteurs ayant produit les évaluations extrêmes d'expliquer leurs points de vue respectifs. Ces explications achevées et comprises de tous, une nouvelle estimation est produite et c'est alors la moyenne arithmétique de ces estimations qui est prise en compte.

Planification des *releases*

- Affiner les risques, les incertitudes en fonction des retours de la revue et de la rétrospective du dernier sprint
- Ajuster la vélocité de l'équipe (capacité de travail de l'équipe en nombre de points de story)
- Affiner, (re-)planifier le(s) prochain(s) sprint(s)
 - Définition de prêt et fini
 - Nombre de points
- Affiner, (re-)planifier la future release

Planification d'un *Sprint*

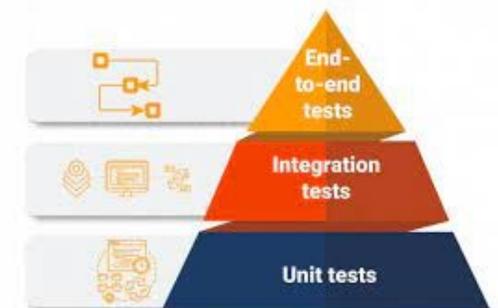
- Confirmer les stories prêtes
 - Définition de prêt et fini
- Evaluer la nombre de points d'une story
 - Ponts de récit ou journée idéale (homme/jour)
- Organisation de l'essaiage (plusieurs stories en parallèle, répartition des ressources)
- Décomposition des stories en tâches
- Affectation des tâches aux développeurs

Exécution d'un *sprint*

- Conception, réalisation et test des *stories*
- Organisation, affectation des tâches
- Inclut les *sprints* journaliers, dont les mêlées (Scrums)

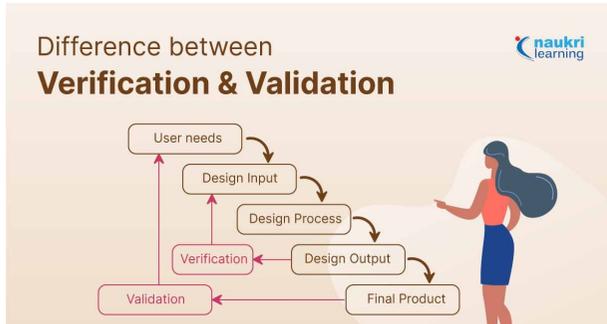
Solidification d'une *release*

- Documentation technique détaillée expliquant l'architecture
 - Par exemple à l'aide de diagrammes UML
- Tests de qualité de services
 - performances, coût,
 - robustesse, sécurité ...
- End to End Testing



Validation d'une release

- Test d'usage avec le client, des utilisateurs
 - Démonstration du produit
 - Bref manuel utilisateur expliquant comment installer et utiliser l'application



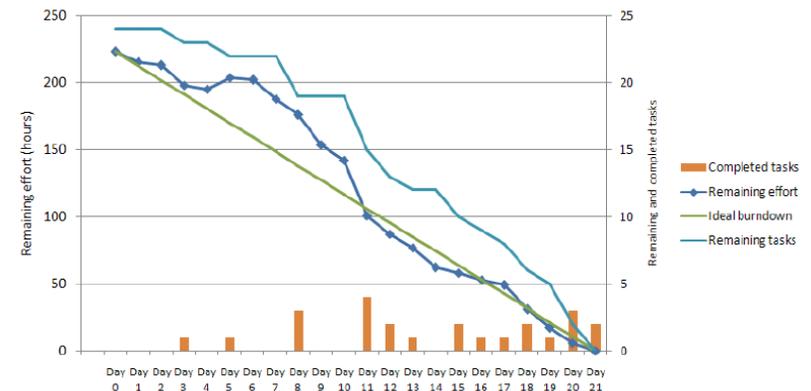
Do the product *right*
VS.
Do the *right* product

7 - Suivi de projet Scrum

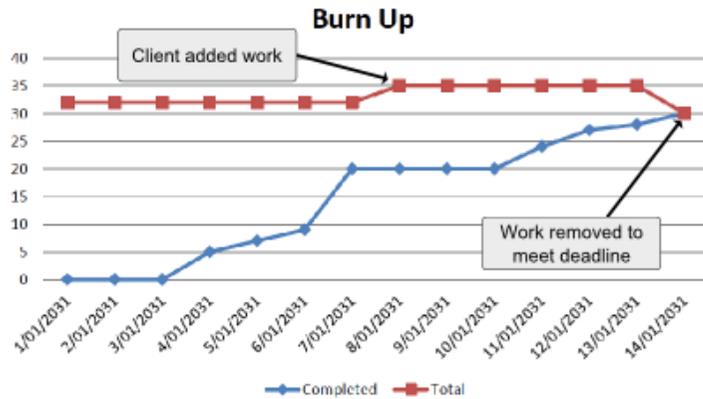
Indicateurs

- Sprint
 - *Burndown* de *sprint* (orienté reste à faire)
 - *Burnup* de *sprint* (orienté ce qui a été déjà fait)
- Release
 - *Burndown* de *release*
 - *Burnup* de *release*
- Equipe
 - Vitesse (capacité de l'équipe)
 - Suivi des obstacles
 - Perturbations exogènes ou endogènes qui perturbent le bon déroulement du sprint
 - Peut de générer de nouvelles tâches (dette) et/ou leur réorganisation

Burndown graphe



Burnup graphe



ISTIC- L2GEN

61

Vélocité

- Estime la capacité de l'équipe en nombre de points de stories par sprint
- Utilisé pour la planification de la release
- Affinée à la fin de chaque sprint
- Tendance à la stabilité

ISTIC- L2GEN

62

8 – Revue de projet Scrum

Revue d'un *sprint*

- Démonstration de chaque story finie
- Collecte du feedback
- Evaluation du niveau de réalisation de l'objectif
- Evaluation de l'impact du travail réalisé et décision d'une release (livraison) ou pas

ISTIC- L2GEN

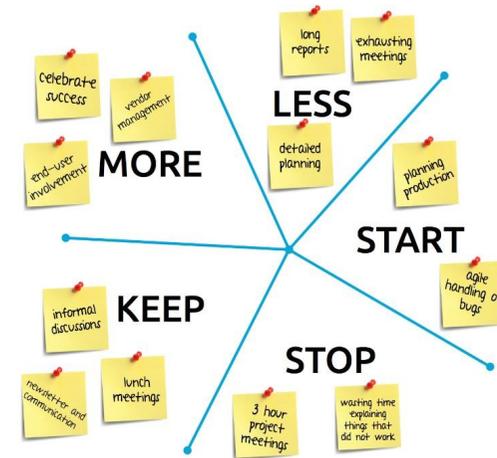
64

Rétrospective d'un *sprint*

- Collecter les information sur le sprint passé
 - Ce qui c'est bien passé, moins bien passé
- Identifier les choses à améliorer
- Décider d'améliorer certaines choses

- Combinée à la revue, de courte durée

Rétrospective de sprint : « Starfish »



9 – Retour sur les LLM

Large Language Models

ChatGPT, Copilot...

(section basée sur les slides de O. Barais)

ChatGPT ?

- ChatGPT =
 - ChatBot (mais qui marche) produit par la compagnie OpenAI
 - 30 novembre 2022
 - 1 million d'utilisateurs en 5 jours
 - 100 millions fin janvier 2023
 - 1.16 milliard en mars 2023
 - Fondé sur un LLM = *Large Model Language*
 - Tuné sur des dialogues humains
 - Interactions fluides
 - « safe »
 - Son job de chaque instant : **prédire le prochain mot à écrire** via des stats

Votre question

+

Le début de sa réponse



Quel prochain mot ???

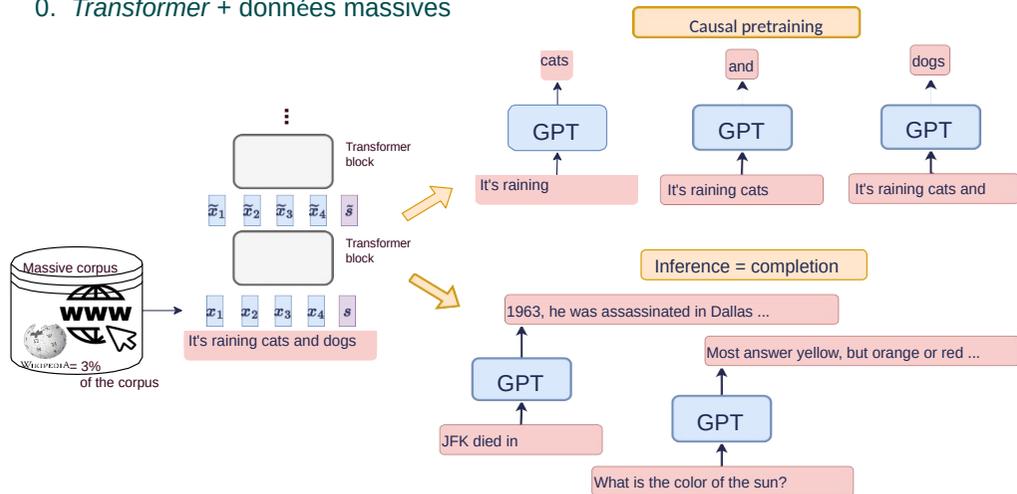
Détails gore sur comment ça marche ?

Improving Language Understanding by Generative Pre-Training

Alec Radford OpenAI, Karthik Narasimhan OpenAI, Tim Salazar OpenAI, Bryce Steiniger OpenAI

<https://www.youtube.com/watch?v=7ell8KEbhJo&t=1s>

0. Transformer + données massives

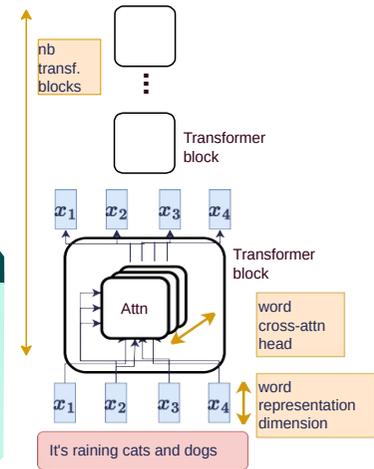


1. Toujours plus!

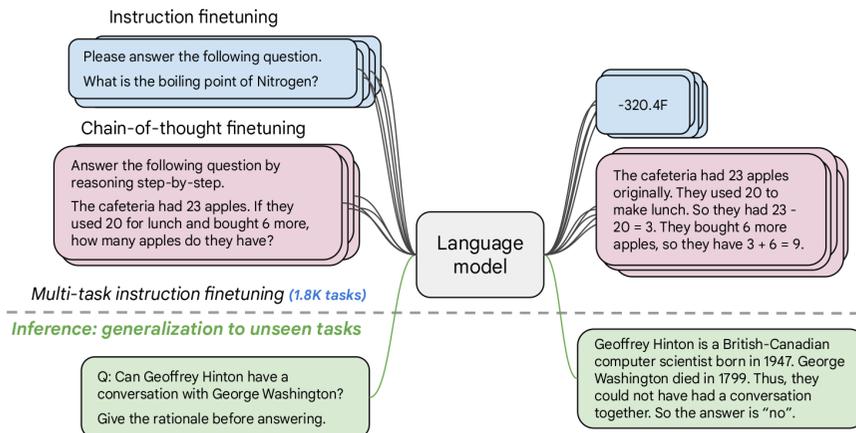
- + de mots en entrée [500 → 2k, 32k]
- + de dimensions dans l'espace des mots [500-2k → 12k]
- + de têtes d'attention [12 → 96 (dim 128)]
- + de blocks/couches [5-12 → 96]

175 Milliards de paramètres... Ça fait

- ▶ $1.75 \cdot 10^{11} \Rightarrow 300 \text{ Go} + 100 \text{ Go}$ (stockage des données en inférence) $\approx 400\text{Go}$
- ▶ GPU NVidia A100 = 80Go de mémoire (=20k€)
- ▶ Coût pour (1) entrainement : 4.6 Millions d'€



2. L'affinage sur différentes tâches de raisonnement (±) complexe

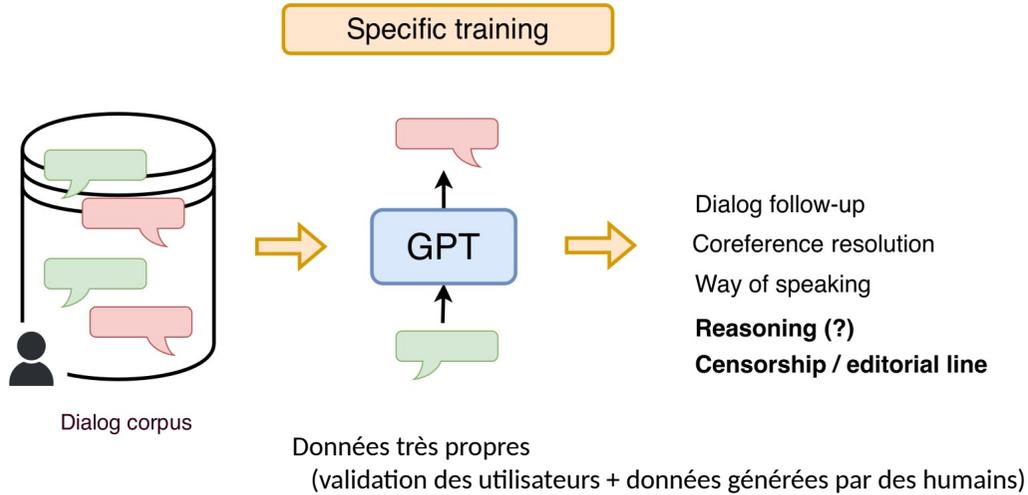


2. L'affinage sur différentes tâches de raisonnement (±) complexe

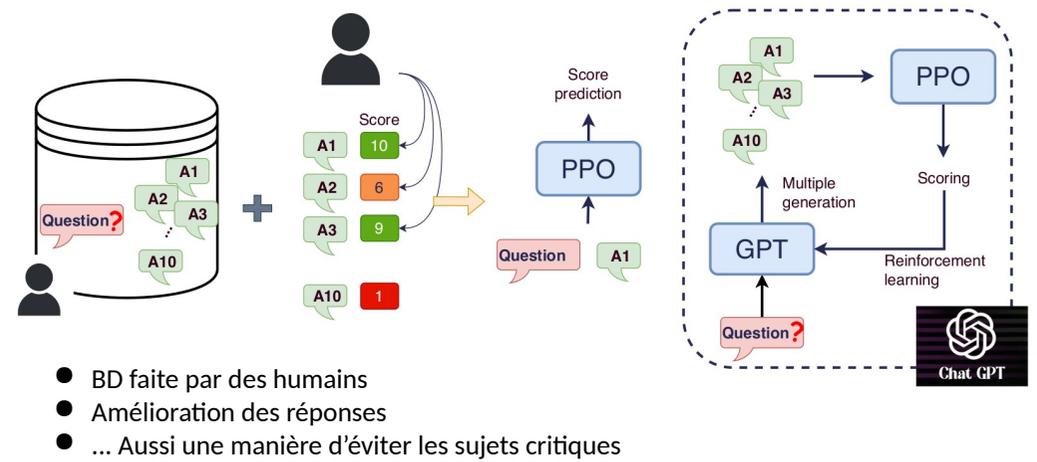
<p>TO-SF</p> <p>Commonsense reasoning Question generation Closed-book QA Adversarial QA Extractive QA Title/context generation Topic classification Struct-to-text ...</p> <p>55 Datasets, 14 Categories, 193 Tasks</p>	<p>Muffin</p> <p>Natural language inference Code instruction gen. Program synthesis Dialog context generation</p> <p>Closed-book QA Conversational QA Code repair ...</p> <p>69 Datasets, 27 Categories, 80 Tasks</p>	<p>Natural Instructions v2</p> <p>Cause effect classification Commonsense reasoning Named entity recognition Toxic language detection Question answering Question generation Program execution Text categorization ...</p> <p>372 Datasets, 108 Categories, 1554 Tasks</p>		
<p>CoT (Reasoning)</p> <p>Arithmetic reasoning Commonsense Reasoning Implicit reasoning ...</p> <p>Explanation generation Sentence composition ...</p> <p>9 Datasets, 1 Category, 9 Tasks</p>	<p>MMLU</p> <p>Abstract algebra College medicine Professional law Sociology Philosophy ...</p> <p>57 tasks</p>	<p>BBH</p> <p>Boolean expressions Tracking shuffled objects Dyck languages Navigate Word sorting ...</p> <p>27 tasks</p>	<p>TyDiQA</p> <p>Information seeking QA</p> <p>8 languages</p>	<p>MGSM</p> <p>Grade school math problems</p> <p>10 languages</p>

Mots-clés = tâches => apprendre à structurer

3. Suivi de dialogue



4. Suivi de dialogue & amélioration des réponses



Pffff ça marche pas ce truc...Il hallucine...

Question : Que peux-tu me dire d'Alexandre Termier ?

Réponse : Alexandre Termier est un chercheur français en informatique et en intelligence artificielle. Il est notamment connu pour ses travaux dans le domaine de l'intelligence artificielle appliquée aux jeux, en particulier aux échecs. Termier a obtenu son doctorat en informatique à l'Université Joseph Fourier à Grenoble, en France. **FAUX**

2 problèmes ici:

- Demander des faits à ChatGPT = ERREUR. Ce n'est pas son job. Utilisez Google !!!
- Utiliser ChatGPT gratuit (modèle 3.5) = il hallucine plus

OP Que peux-tu me dire d'Alexandre Termier ?

Finished browsing

Alexandre Termier est professeur à l'Université de Rennes 1, au sein du laboratoire d'informatique IRISA. Il est également le chef du groupe de recherche LACODAM de l'INRIA à l'IRISA, qui se concentre sur le Data Mining et l'Intelligence Artificielle. Termier est également membre associé du Laboratoire d'Informatique de Grenoble (LIG), au sein du groupe Scalable Information Discovery and Exploitation¹. [Lien vers SA page](#)

Au fil de sa carrière, Alexandre Termier a participé à de nombreux projets de recherche. Quelques exemples de son travail incluent:

1. "Anomaly Detection in Streams with Extreme Value Theory" (Détection d'anomalies dans les flux avec la théorie des valeurs extrêmes), en collaboration avec Alban Siffer, Pierre-Alain Fouque et Christine Largouët, publié à KDD 2017². [Lien vers DBLP](#)

En activant le plugin « Bing » : il peut lire et résumer sa page web. Demander des faits devient envisageable.

VRAI ;!)

Quelques utilisations pour des informaticiens

- Répondre à des questions de prog plus vite que StackOverflow
- Trouver et expliquer des erreurs dans des codes
 - *Et parfois en rajouter ;-)*
- Écrire du code à partir d'une spécification plus ou moins précise
- Traduire du code d'un langage à un autre
- **Préf Assistant pédagogique d'info interactif, patient, toujours dispo**

Création de code

OP Tu es un expert en C#/WPF avec plus de 10 ans d'expérience. Ta mission est d'ajouter un composant graphique WPF à une application de gestion d'emploi du temps. Le rôle de ce composant est d'afficher un emploi du temps. Il est basé sur un Canvas. Sa donnée d'entrée est un emploi du temps pour une semaine type.

On a d'abord une List<Cours>, qui est la liste des cours.

Chaque Cours a une propriété Intitule qui donne une chaîne courte (entre 3 et 4 caractères).

Et la liste des créneaux de la semaine pour ce cours, représentée par la propriété suivante :

Dictionary<DayOfWeek, List<CreneauHoraire>> SemaineType

Où CreneauHoraire a les propriétés suivantes :

```
public TimeSpan HeureDebut { get; set; }
public TimeSpan HeureFin { get; set; }
```

Création de code suite

Le dictionnaire donne pour chaque jour une liste des créneaux affectés à ce jour. Et les heures de début et fin du créneau sont données par l'objet CreneauHoraire.

Le Canvas qui représente l'emploi du temps a la spécification suivante :

- son arrière plan est blanc
- il y a un grille gris pâle, où les lignes verticales séparent les jours ouvrés de la semaine (du lundi au vendredi), et les lignes horizontales représentent les heures, une par heure, entre 7h et 19h. L'heure est écrite en petit à gauche au dessus de chaque ligne, le jour est écrit en haut au milieu de chaque colonne.
- un créneau de cours est un rectangle coloré qui prend toute la largeur de la colonne, et l'espace vertical correspondant à sa durée.
- il est placé dans la colonne correspondant à son jour, et verticalement de son heure de début à son heure de fin. Attention les créneaux ont des minutes (ex: 18h15), qui doivent être prises en compte pour son placement correct.
- l'intitulé du cours est écrit au milieu du rectangle coloré.
- une couleur arbitraire est assigné à chaque cours

Création de code suite



Pour réaliser ce composant WPF, on va utiliser principalement un contrôle `Canvas` et les contrôles `Rectangle` et `TextBlock` pour afficher les créneaux.

Commençons par définir la classe Cours et la classe CreneauHoraire.

```
csharp Copy code
public class CreneauHoraire
{
    public TimeSpan HeureDebut { get; set; }
    public TimeSpan HeureFin { get; set; }
}

public class Cours
{
    public string Intitule { get; set; }
    public Dictionary<DayOfWeek, List<CreneauHoraire>> SemaineType { get; set; }
}
```

On va utiliser ensuite un `UserController` pour représenter le composant d'emploi du temps.

.....blablabla...

Création de code Résultat :



Université de Rennes Réécriture de code

OP

J'ai cette structure de données :

Dictionary<TDemiSemestre, Dictionary<DaysOfWeek, List<CreneauHoraire>>>

Que je parcoure avec cette boucle :

```
foreach (DaysOfWeek jour in Sem_SemaineType[DemiSemestre].Keys)
{
    foreach (CreneauHoraire creneau in Sem_SemaineType[DemiSemestre][jour])
    {
        if (creneau.Flottant) return true;
    }
}
```

pour vérifier si l'un des CreneauHoraire a la propriété Flottant à true.
Pourrais tu me transformer cette boucle en construction linq plus courte ?

Université de Rennes Réécriture de code



Bien sûr, vous pouvez utiliser LINQ pour simplifier cette boucle en une seule ligne. Voici comment vous pouvez le faire :

```
csharp
bool flottantExists = Sem_SemaineType[DemiSemestre]
    .SelectMany(pair => pair.Value)
    .Any(creneau => creneau.Flottant);
```

Dans cet exemple, `SelectMany` est utilisé pour "aplatir" les listes de `CreneauHoraire` dans une seule séquence, puis `Any` est utilisé pour vérifier si au moins un `CreneauHoraire` a la propriété `Flottant` à `true`.

Notez que cette version renvoie `false` si `Sem_SemaineType[DemiSemestre]` est vide ou si aucun `CreneauHoraire` n'a la propriété `Flottant` à `true`, tout comme votre version avec les boucles.



Tout n'est pas rose au pays de ChatGPT

- Hallucinations -> on a vu, et ça arrivera toujours!
- Longueur de contexte limitée
 - « mémoire » de 32k tokens max (disons 20k mots) avec GPT-4
 - -> si conversation longue oublie le début, donc la consigne !
 - Très problématique quand écriture collaborative d'un code
 - On donne une spec, il donne un code
 - On trouve une erreur ou un cas nouveau à gérer -> question -> correction
 - On itère une ou deux fois => Ses réponses se mettent à oublier des corrections précédemment effectuées, voire la consigne initiale !
- Manque d'explicabilité/interprétabilité
- Manque de stabilité/prédictibilité
- Manque de transparence
- Manque de confidentialité : interdit dans certaines entreprises/projets

Bottes de 7 lieues si on comprend ce qu'on fait
Déconnecteur de cerveau sinon

Je vous encourage et je m'attends à ce que vous **utilisiez** l'IA (ChatGPT et/ou Copilot) dans ce cours
Apprendre à utiliser l'IA est une compétence émergente et il est important de savoir l'utiliser en génie logiciel

Soyez toutefois conscient des **limites** de ChatGPT/Copilot/... : <https://youtu.be/R2fjRbc9Sa0>

- Si vous fournissez des questions (*prompts*) en fournissant peu d'effort, vous obtiendrez des résultats de faible qualité. Affinez vos *prompts* afin d'obtenir de bons résultats. **Cela demande du travail.**
- Ne vous fiez jamais à ce qu'il dit. **Gardez un esprit critique. Vous serez responsable de toute erreur** ou omission fournie par l'outil. Il vaut mieux utiliser l'IA sur des sujets que vous comprenez.
- L'IA est un outil que vous **devez reconnaître avoir utilisé**. Veuillez utiliser les commentaires de votre code pour expliquer à quelles fins vous avez utilisé l'IA et quels prompts vous avez utilisés pour obtenir les résultats. Ne pas le faire constitue une violation des politiques d'honnêteté académique.
- Vous devez **comprendre** le code que propose l'IA. Vous pouvez d'ailleurs lui demander de vous l'expliquer. Lors de votre évaluation individuelle **tout code que vous n'êtes pas capable d'expliquer entrainera une note de 0.**
- Réfléchissez au moment où cet outil est utile. Ne l'utilisez pas s'il n'est pas approprié au contexte du travail à fournir, **au minimum ayez un usage raisonné en lien avec l'impact environnemental associé.**