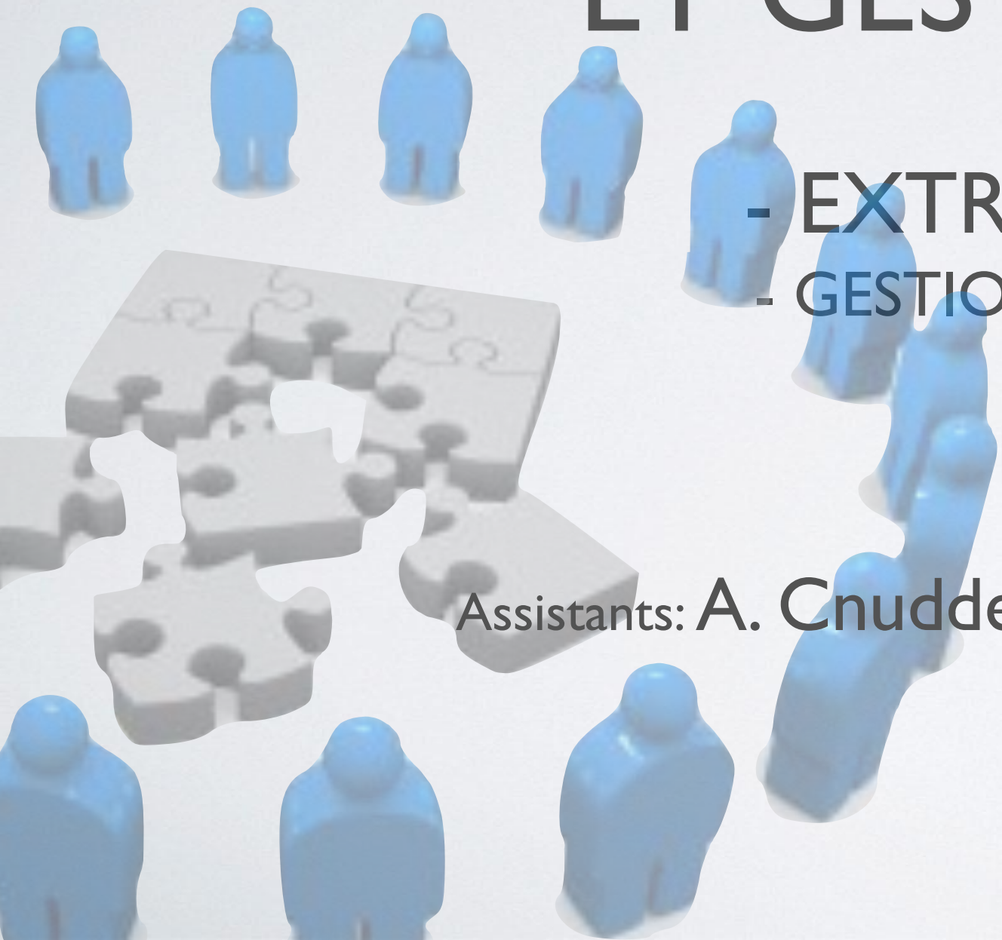
INFOF-307 - GÉNIE LOGICIEL ET GESTION DE PROJETS

- EXTREME PROGRAMMING -
- GESTION DE LA QUALITÉ DU CODE -



Pluquet Frédéric

Assistants: A. Cnudde, A. Laval, Y. Molinghen, A. Reynouard

2021-2022

ULB

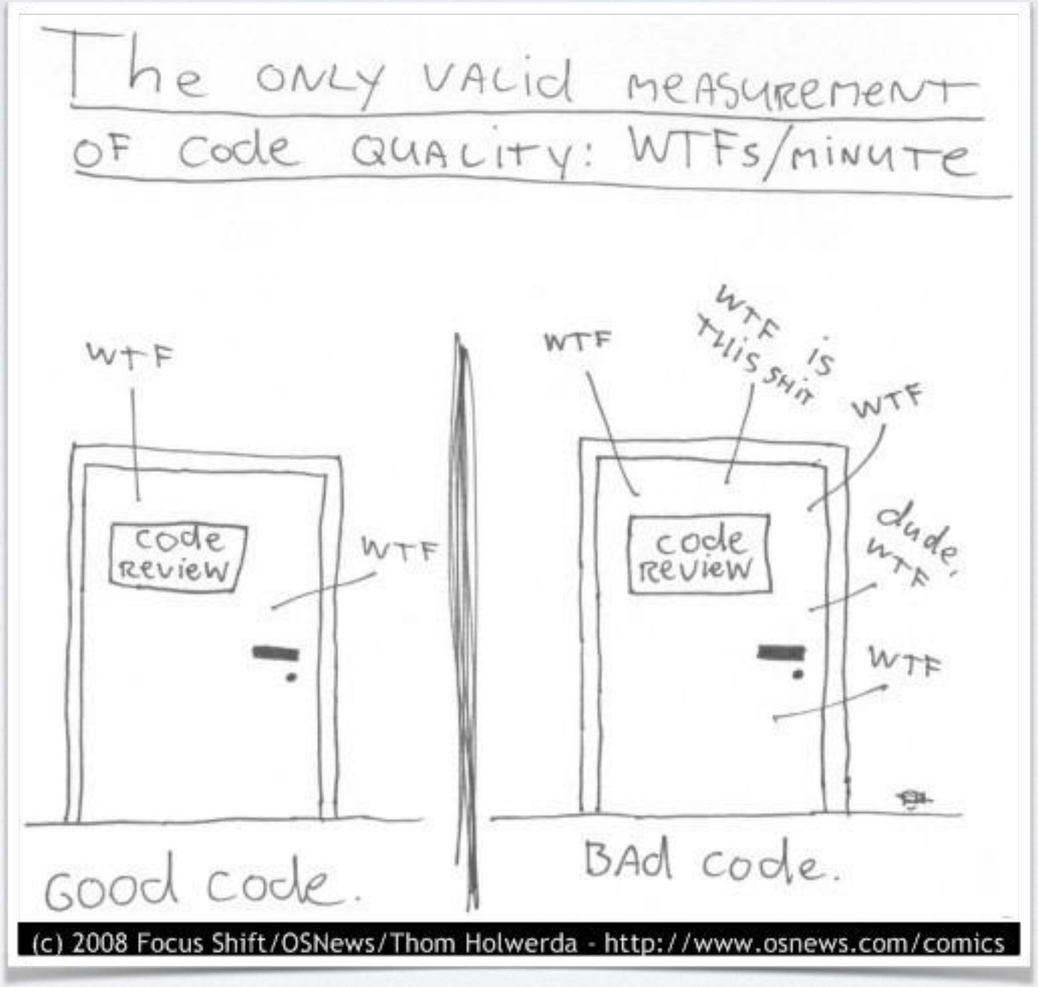
SOMMAIRE

- Introduction
- Gestion de projet
- Gestion de l'équipe
- **Gestion de la qualité du code**
- Autres considérations
- Conclusions

QUALITÉ DU CODE : SOMMAIRE

- **Rappels**
- Quelques concepts supplémentaires liés à XP
- Séances d'analyse live de code

QUALITÉ DU CODE

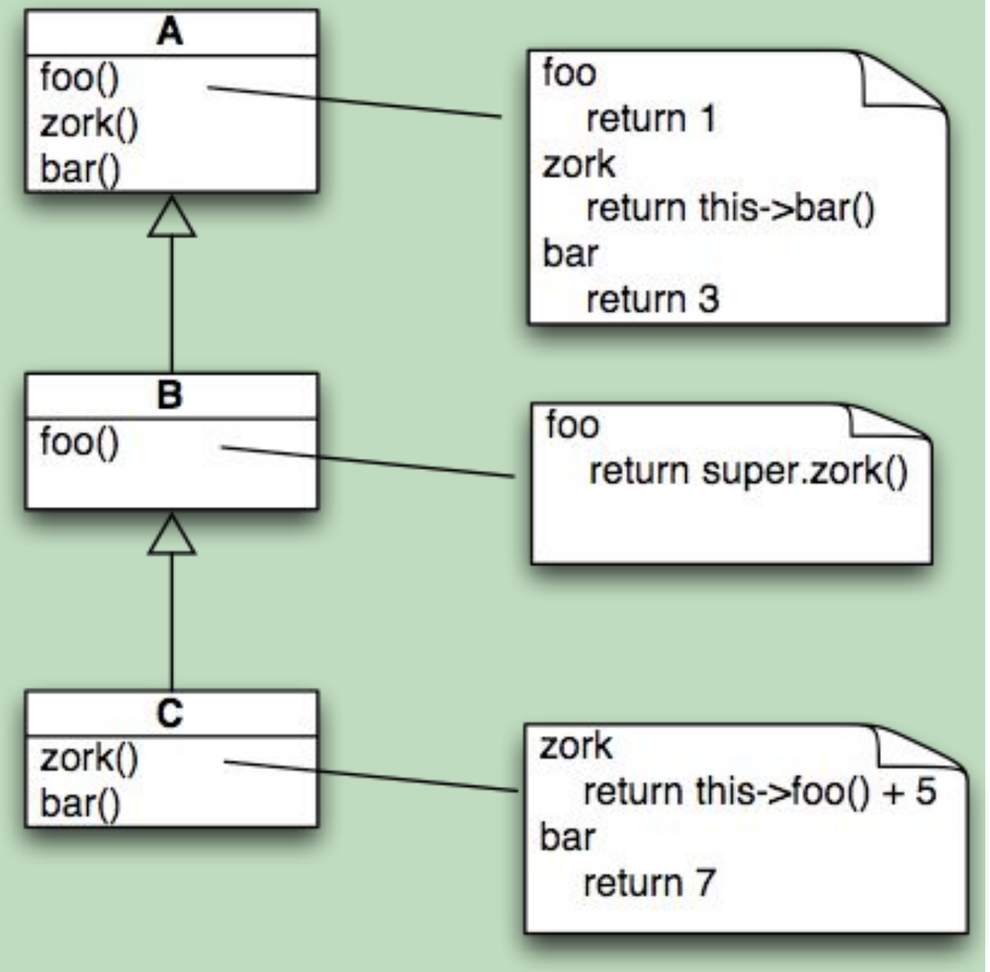


ORIENTÉ OBJET

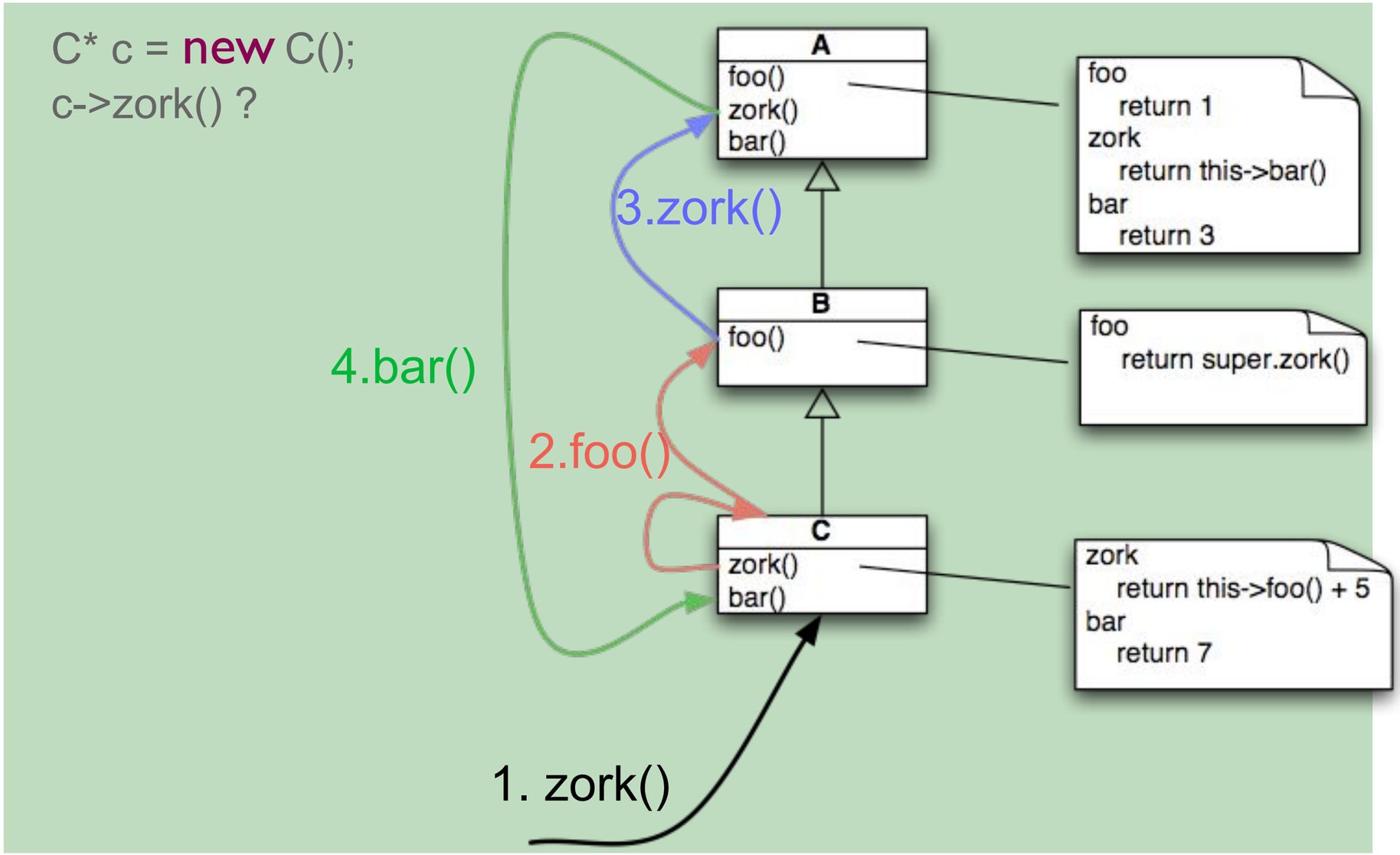
- Objet, Instance
  - Messages
- Classe
  - Attributs
  - Méthodes
  - Abstraite, Interface
- Héritage
- Method Lookup
- Polymorphisme

Follow This 1

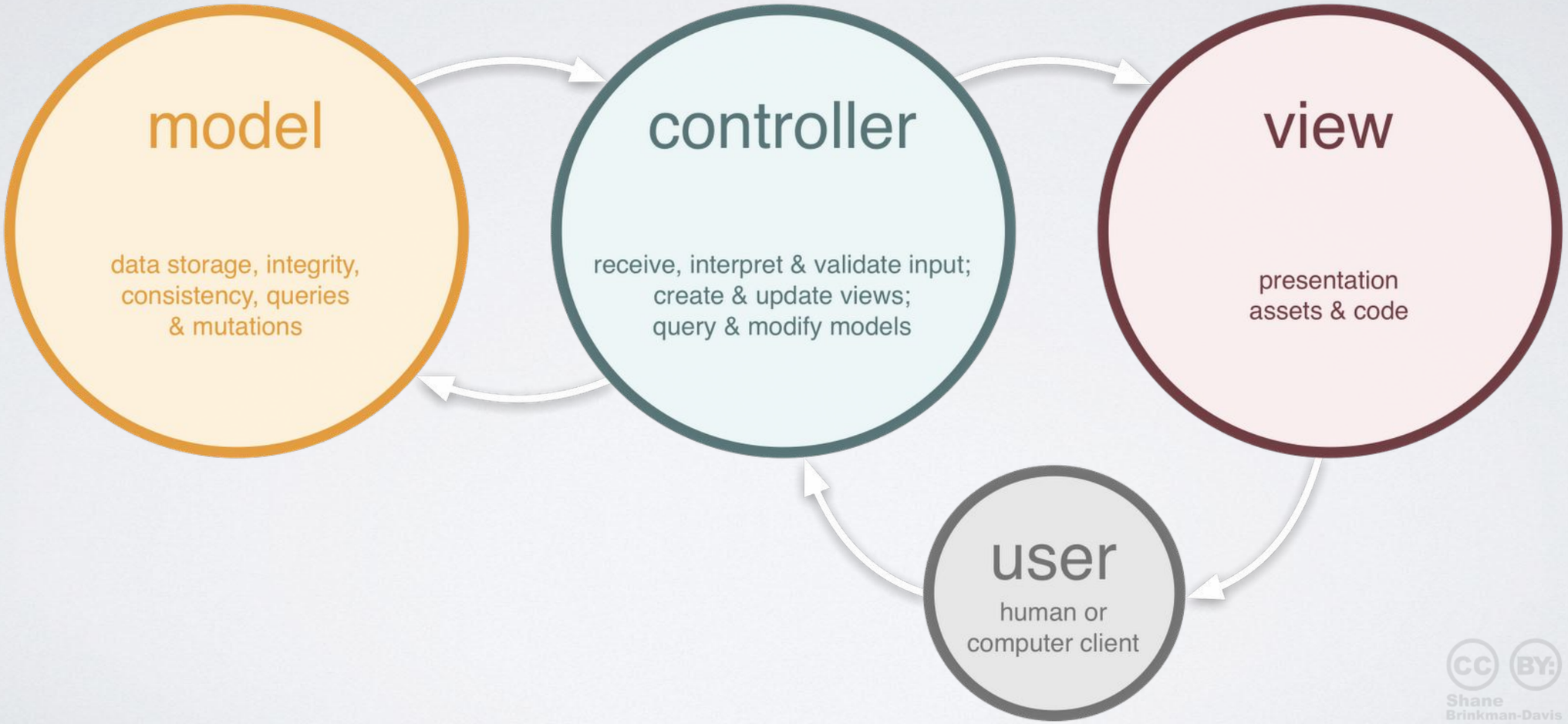
C* c = new C();
c->zork() ?



Follow This 1 : Solution



ARCHITECTURE



Observer and MVC

MVC = Model View Controller

Framework (not a pattern) to build applications with a GUI

- Uses several design patterns
  - Composite, Observer, Strategy, Factory, Bridge, ...
- Big

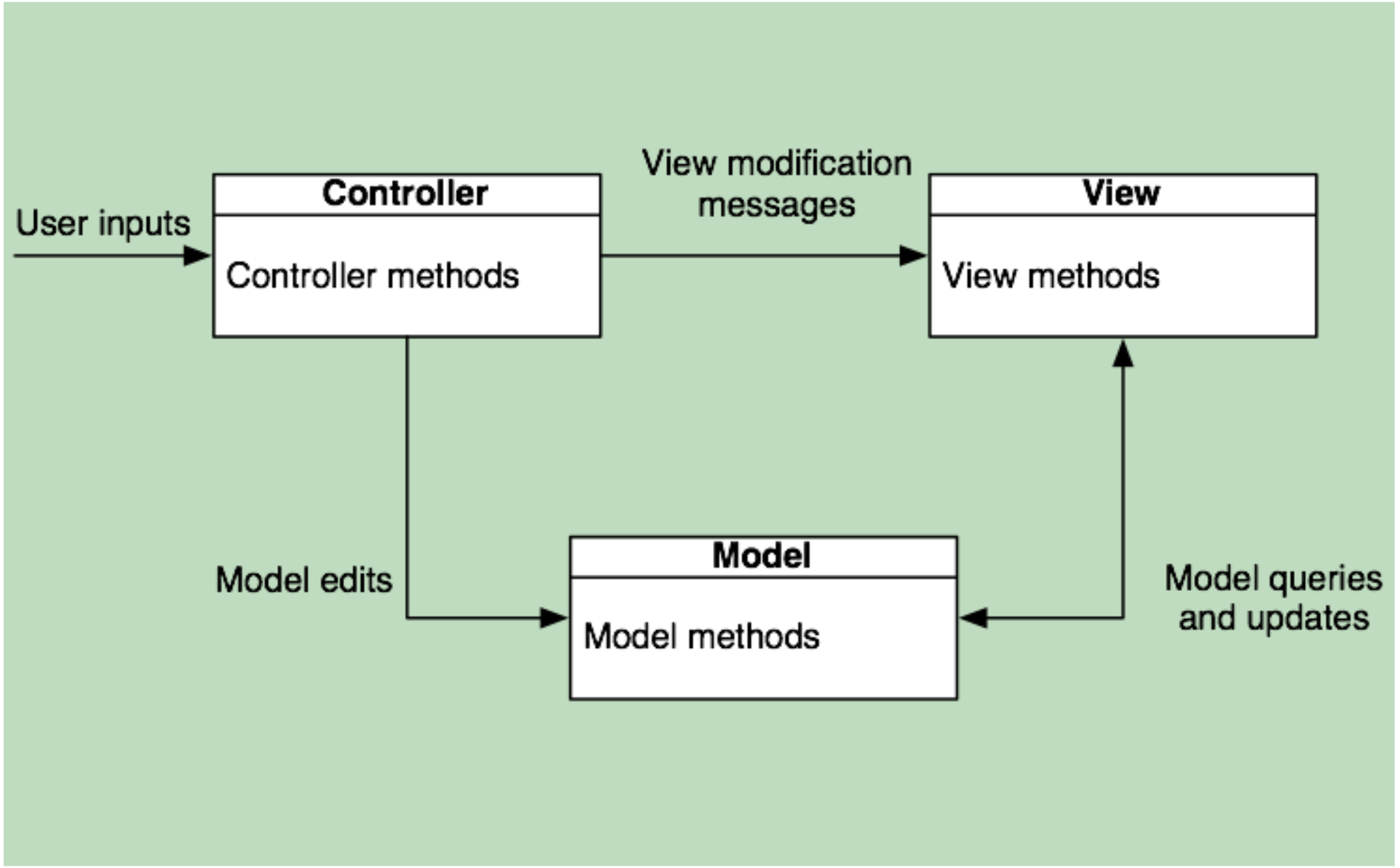
Slide 636 d'Analyse et Méthodes - ULB

Model-View-Controller

How to build a GUI application in such a way that:

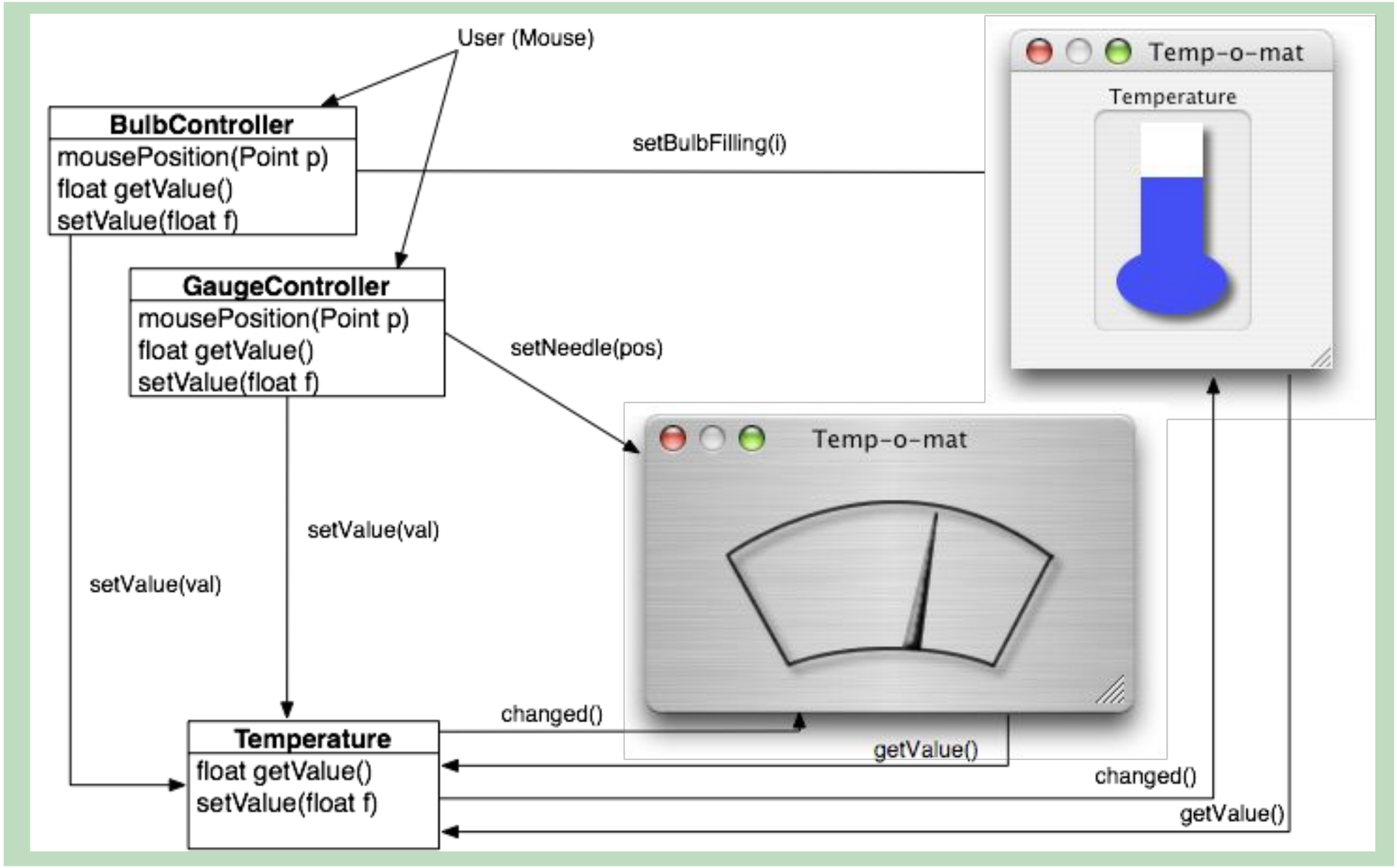
- there is a clearly separated model that implements your actual domain knowledge
- this model can be viewed by several GUI interfaces that have to remain consistent with the model
- nothing of this should be hardcoded

MVC basic structure



Slide 638 d'Analyse et Méthodes - ULB

MVC Application Example



Slide 640 d'Analyse et Méthodes - ULB

CODE

- Convention standard de nommage
- Longueur des méthodes
- Éviter le code dupliqué
  - Utiliser le refactoring
- Éviter les nombres magiques (magic numbers)

CODE

- Deux principes de GRASP:
  - Haute cohésion
  - Couplage faible
- Loi de Demeter

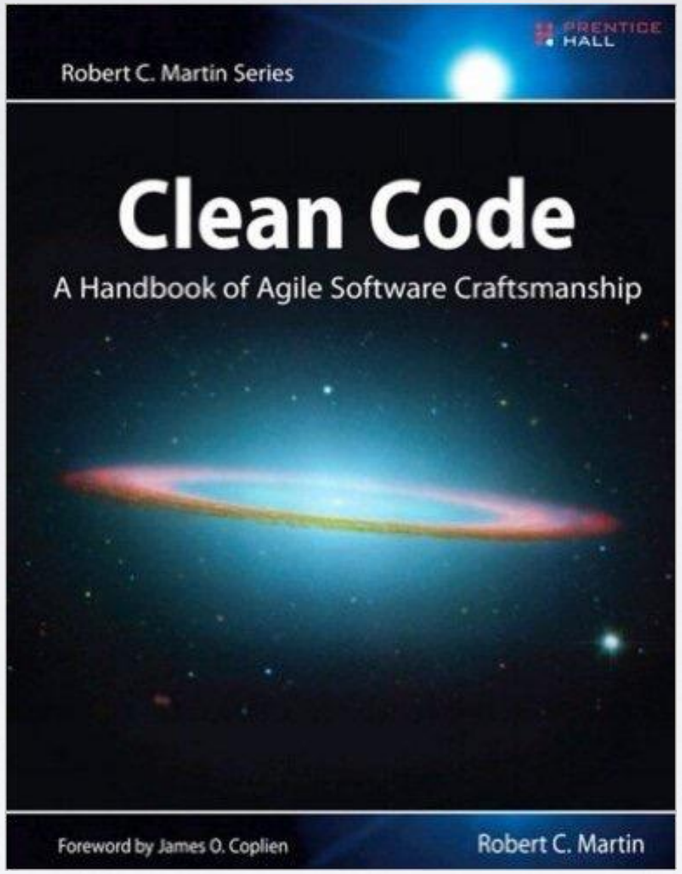
The law of Demeter

You are only allowed to send messages to:

- an argument passed to you
- an object you create
- self, super

Avoid global variables

Avoid objects returned from message sends other than self



https://www.amazon.fr/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882

QUALITÉ DU CODE : SOMMAIRE

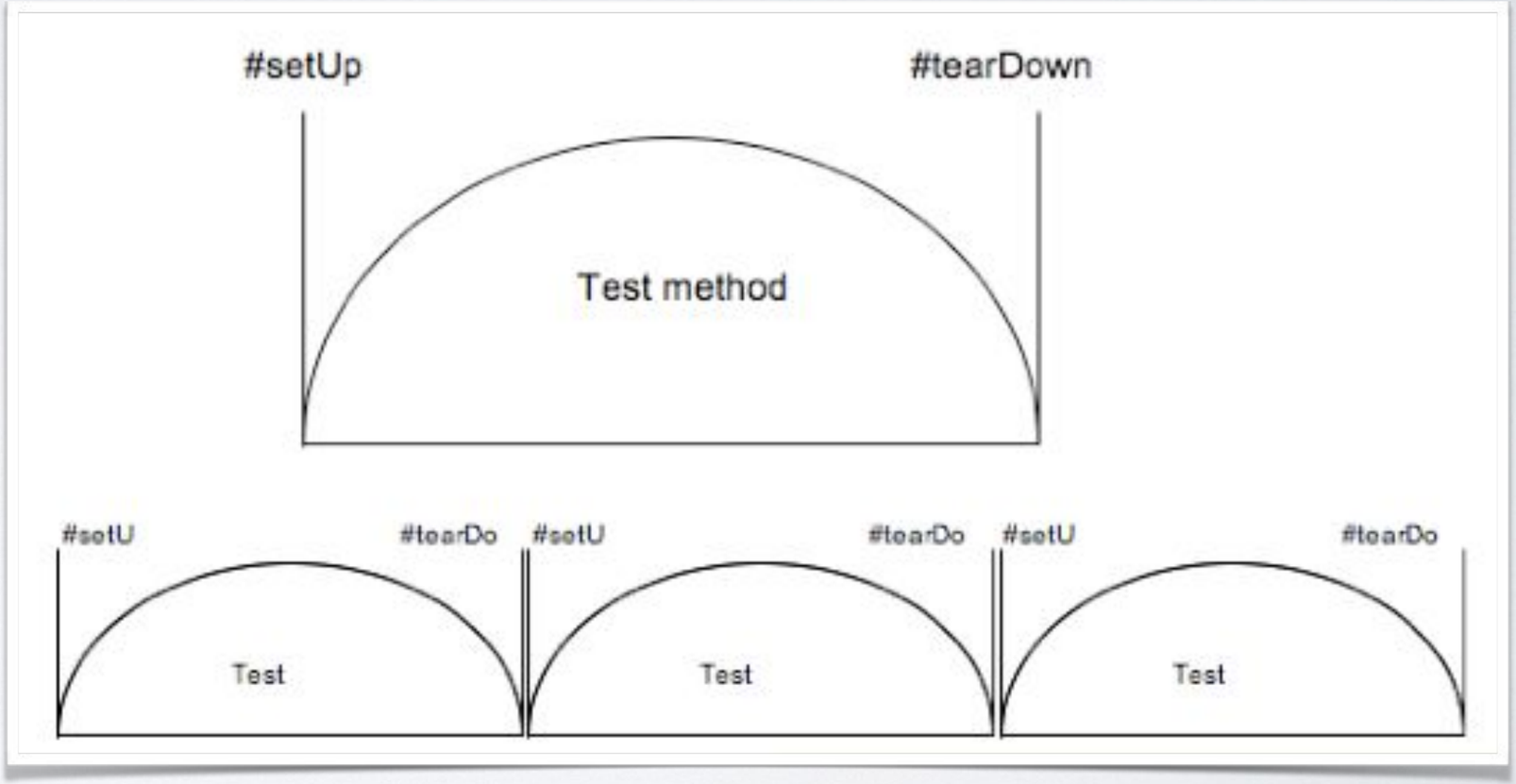
- Déjà discuté lors des analyses live de code
- Rappels
- **Quelques concepts supplémentaires liés à XP**

PROGRAMMATION

- Développement piloté par les tests
  - Un test est **toujours automatisé**
    - Exécutable à n'importe quel moment sans l'intervention d'un humain
    - Peut être exécuté à souhait

TESTER
TESTS D'UNITÉ

- Créer le contexte
- Envoyer le stimulus
- Vérifier le résultat



TEST FIRST !

- Une règle à respecter en XP:
  - Un code peut être écrit que si il a été testé avant !
- Permet de concevoir du code testable
  - Si on le fait en fin de projet, les parties du système peuvent être trop enchevêtrées
- Eviter la régression
- Attention: il vous faudra du courage pour y arriver :)

CONCEPTION SIMPLE

- «Conception» en XP = toute réflexion qui touche à la structure d'un système plutôt qu'à la substance
- En XP: pas de bon design, mais des designs adaptés pour les résultats attendus

CONCEPTION SIMPLE

- Pas d'investissements spéculatifs
  - Spéculations comme à la Bourse: si on a mal spéculé, on risque de perdre beaucoup !
  - Toujours dans l'idée de changements continus (besoins, techniques, ...)
  - A la spéculation, XP préfère le rendement.
  - «Do The Simplest Thing That Could Possibly Work»
  - «You Aren't Gonna Need It»

CONCEPTION SIMPLE

- Simplicité **!=** Facilité
  - Souvent plus difficile de faire du code simple !
  - Ex: si on a une méthode qui effectue une opération récursive sur des fichiers contenus dans un répertoire et que l'on a besoin d'effectuer une autre opération avec le même parcours
    - Facilité: copier/coller et modifier l'opération => perte de temps si découverte d'une erreur dans la méthode originale

CONCEPTION SIMPLE

- Qualités du code:
  - Tous les tests doivent être exécutés avec succès
  - Chaque idée doit être implémentée clairement et isolément
  - Tout doit être écrit une et une seule fois
  - Le nombre de classes, de méthodes et de lignes de code doit être minimal

REFACTORING

- Qu'est-ce que c'est ?
- Pourquoi est-ce nécessaire ?
- Exemples
- Support
- Obstacles au refactoring

REFACTORING

- The process of changing a software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure [Fowl99a]
- A behaviour-preserving source-to-source program transformation [Robe98a]

REFACTORINGS CLASSIQUES

Class refactoring	Method Refactoring	Attribute Refactoring
ajouter une (sous-)classe à une hiérarchie	ajouter une méthode à une classe	Ajouter une variable à une classe
renommer une classe	renommer une méthode	renommer une variable
supprimer une classe	supprimer une méthode	supprimer une variable
	descendre une méthode	descendre une variable
	monter une méthode	monter une variable
	ajouter un paramètre à une méthode	créer des accesseurs
	déplacer une méthode	
	extraire du code dans une méthode	

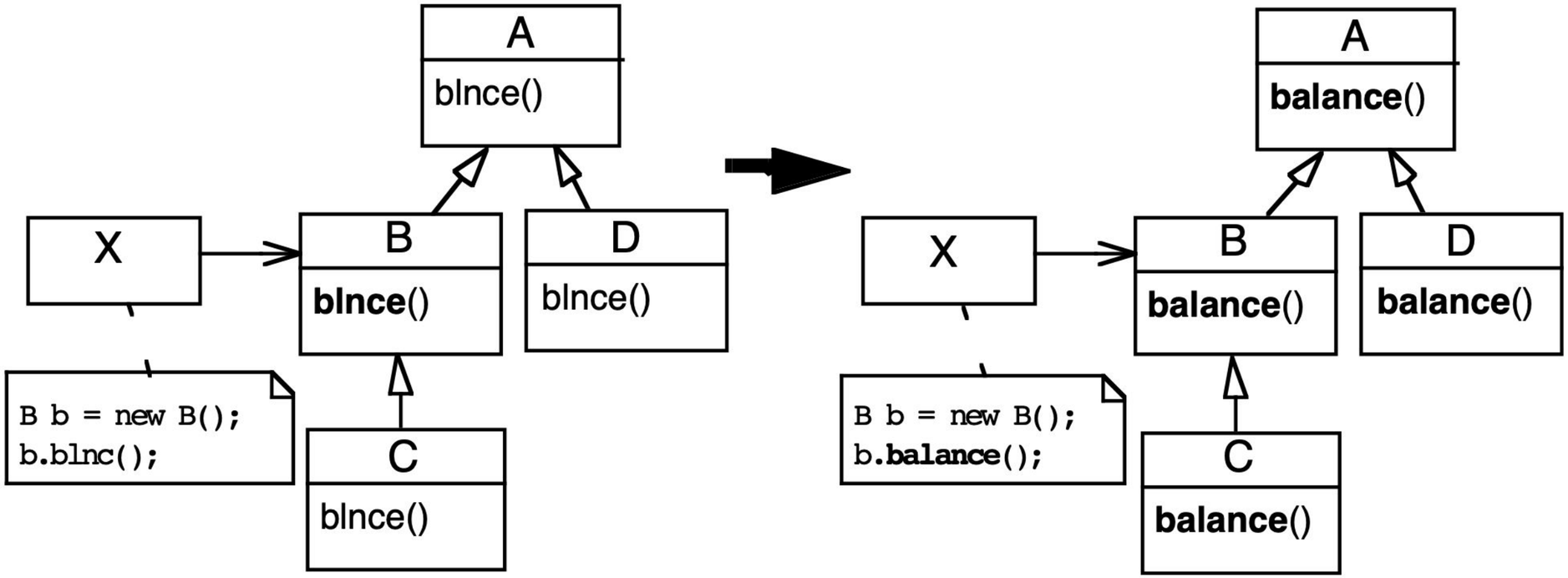
POURQUOI REFACTORISER ?

- “Grow, don’t build software” (Fred Brooks)
  - Certains diront qu’un bon design permet d’écrire du bon code sans qu’aucune refactorisation soit nécessaire...
- En réalité :
  - Extrêmement difficile d’avoir un design parfait directement
  - Vous ne pouvez pas comprendre complètement le domaine du programme
  - Vous ne pouvez pas comprendre complètement les demandes des clients
  - Vous ne pouvez pas vraiment planifier comment le système va évoluer

POURQUOI REFACTORISER ?

- Le refactoring vous aide à
  - Manipuler le code dans un environnement sûr
    - Le comportement est préservé
  - Recréer une situation où l'évolution est possible
  - Comprendre le code existant
- Un code doit être maintenu !
  - C'est une façon sûre de le faire

EXEMPLE : RENOMMER UNE MÉTHODE



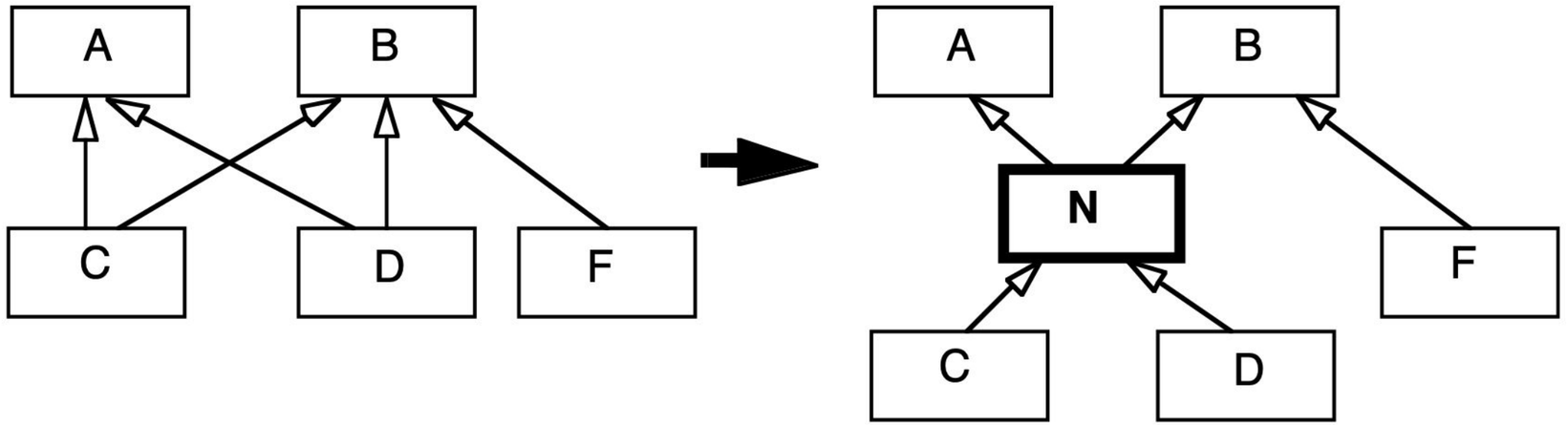
RENOMMER UNE MÉTHODE MANUELLEMENT

- Vérifier si une méthode n'existe pas dans la classe et les sous/super-classes avec le même «nom»
- Parcourir toutes les implémentations de la méthode (définitions)
- Parcourir tous les appels de la méthode
- Editer et renommer toutes les implémentations
- Editer et renommer tous les appels
- Tester

RENOMMER UNE MÉTHODE

- Préconditions
  - aucune méthode avec la signature de la nouvelle méthode dans la hiérarchie qui contient la méthode
  - [Java] la méthode n'est pas un constructeur
- Post Conditions
  - La méthode a un nouveau nom
  - Toutes les méthodes relevantes dans la hiérarchie ont également le nouveau nom
  - Tous les appels de ces méthodes ont été modifiés
- Autre considération
  - Langages typés statiquement./dynamiquement => scope du renommage

EXEMPLE : AJOUTER UNE CLASSE



AJOUTER UNE CLASSE

- Préconditions
  - Aucune classe ou variable globale avec le même nom dans le même scope
  - Les sous-classes sont toutes des sous-classes de toutes les super-classes
  - [Smalltalk] il doit y avoir au moins une super-classe
- Postconditions
  - la nouvelle classe est ajoutée dans la hiérarchie avec les super-classes comme super-classes et les sous-classes comme sous-classes
  - la nouvelle classe a un nom de classe
  - les sous-classes héritent de la nouvelle classe et plus des super-classes

SUPPORTS

- Encore envie de le refactoriser à la main ?
- Peut être automatisé
  - plus simple
  - plus sûr
- Quels sont les outils nécessaires au refactoring ?

OUTILS POUR LE REFACTORING

Changement efficace	Preuve d'échec
Refactoring Tools - transformation d'un programme source-to-source - préserve le comportement **>> Améliore la structrure**	*Regression Testing* - Répétition des tests écrits - Pas d'intéraction utilisateur - Déterministe **>> Vérifie les dégats p.r. aux précédentes versions**
Development Environment - Edit-Compile-Run rapide - Intégré dans l'environnment **>> Commode**	*Configuration & Version Management* - suivi de plusieurs versions - suivi de qui à fait quoi **>> Permet de revenir aux versions précédentes**

OBSTACLES POUR LE REFACTORING

- Problème de performance
  - «Refactoring will slow down the execution»
- Problèmes culturels
  - «Nous vous payons pour ajouter des fonctionnalités, pas pour améliorer la beauté du code !»
- Touche pas, ça fonctionne !

OBSTACLES POUR LE REFACTORING

- Le développement est toujours sous contraintes
  - Le refactoring prend du temps
  - Le refactoring après la livraison ?
  - La méthodologie doit prendre cela en compte, comme les tests !

LE MYTHE DE LA PERFORMANCE

- Ne pensez pas qu'un software clairement écrit est lent !
- Si seulement 10% d'un système consomme 90% des ressources => ne s'occuper que des 10%
  - Le refactoring aide à localiser les parties à changer
  - Le refactoring vous permet de vous concentrer sur l'optimisation
- Toujours utiliser un profiler sur votre système «lent» pour vous guider dans vos efforts d'optimisation
  - Ne jamais optimiser directement !

LE MYTHE DE LA PERFORMANCE

- Kent Beck's Directives :
  - Make it work
  - Make it right
  - Make it fast

DES QUESTIONS ?