# Introduction aux méthodes agiles

# Focus sur XP

Les grands principes uniquement

Merci à tous ceux qui ont rendu leurs cours et exposés disponibles sur le web & dans les livres, voir Biblio.

# Bibliographie


- Exigences et spécifications du logiciel -- © RR - LOG3410 – XPproc, Université de Montreal, 2011
- Dave Elliman, XP, Extreme programming
- Céline ROUDET, Module ITR3 – Génie Logiciel
- Raphaël Marvie, Lean Software Development
- eXtreme Programming (XP) CSE30, University of Sunderland Harry R. Erwin, PhD
- eXtreme Programming & Scrum Practices, Embrace Change, Naresh Jain
- Méthode AGILE Les meilleures pratiques Compréhension et mise en oeuvre, Jean-Pierre Vickoff (Auteur), AgileAlliance

# Où s'appliquent les méthodes agiles?

- Des projets où le risque est important.
- Quand vous ou votre «client» ne savez pas où vous voulez aller.
- Des projets où vous devez apprendre et vous adapter.
- Des projets où les technologies évoluent rapidement.

# Ce qui ne «marche» pas

- Des spécifications «complètes» en premier

Specification will be wrong and so big you will lose heart!
You are writing spec. when knowledge is at a minimum
Requirements always change under your feet.

- Commencer par coder sans «aucun design» du tout

The monkey-at-typewriter approach to making it work
If you don't have a clear model in you mind of how it works then it won't!

# Qu'est-ce qui «marche»?

- La simplicité (K.I.S.S)

Procéder par incréments

- Impliquer le «client»
- Classer les tâches par priorité
- Courtes itérations («Short Sprints»)

2 semaines, c'est le mieux
Evaluer la quantité de travail en cours
Une réunion de «rendu» plannifiée
Dix minutes de mêlée journalière
Aucun changement de spécification au sein d'un sprint

- Appliquer les Design patterns
- Tests Unitaires
- Etre fier de son travail
- Une «vraie» communication dans l'équipe

Pair programming, code walkthrough, etc.

# Le manifeste agile

- Manifestation de 17 figures éminentes du développement logiciel (2001) :
  Certains processus prescrits sont jugés trop lourds «*heavyweight*» …
  Des méthodes plus agiles doivent être employées !

  http://agilemanifesto.org/

- Idées de base:
  - Un produit ne peut pas être entièrement spécifié au départ
  - L'économie est trop dynamique: l'adaptation du processus s'impose.
  - Accepter les changements d'exigences, c'est donner un avantage compétitif au client

- http://agilemanifesto.org/

# Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

**Individuals and interactions** over processes and tools
**Working software** over comprehensive documentation
**Customer collaboration** over contract negotiation
**Responding to change** over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

| | | |
|---|---|---|
| Kent Beck | James Grenning | Robert C. Martin |
| Mike Beedle | Jim Highsmith | Steve Mellor |
| Arie van Bennekum | Andrew Hunt | Ken Schwaber |
| Alistair Cockburn | Ron Jeffries | Jeff Sutherland |
| Ward Cunningham | Jon Kern | Dave Thomas |
| Martin Fowler | Brian Marick | |

# Si vous signez le manifeste, …

… vous privilégiez:

- L'interaction entre les personnes **plutôt que les processus et les outils**
  *« Individuals and interactions over processes and tools »*
- le logiciel fonctionnel **plutôt que la documentation pléthorique**
  *« Working software over comprehensive documentation »*
- la collaboration avec le client **plutôt que la négociation de contrats**
  *« Customer collaboration over contract negotiation »*
- la réaction au changement **plutôt que le suivi d'un plan**
  *« Responding to change over following a plan »*

*That is, while there is value in the items on the right, we value the items on the left more."*

# Développement agile ?

« Les méthodes de développement de type Agile suivent un mode de **développement itératif** et **incrémental**, une **planification de projet évolutive** et encouragent les **retours d'expériences fréquents** du client.

Elles incluent également toute une série d'autres valeurs et pratiques qui encouragent l'agilité et une réponse aux changements. »

Craig Larman, 2003

# Les 12 principes

1/ Notre première priorité est de **satisfaire le client** en livrant tôt et régulièrement des logiciels utiles.
2/ Le **changement est accepté**, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client.
3/ **Livrer fréquemment** une application fonctionnelle, toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte.
4/ Les experts métier et les développeurs doivent **collaborer quotidiennement** au projet.
5/ Bâtissez le projet autour de **personnes motivées**. Donnez leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail.
6/ La méthode la plus efficace de transmettre l'information est une **conversation en face à face.**
7/ Un **logiciel fonctionnel** est la meilleure unité de mesure de la progression du projet.
8/ Les processus agiles promeuvent un **rythme de développement soutenable**.
Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment.
9/ Une attention continue à **l'excellence technique et à la qualité de la conception** améliore l'agilité.
10/ La **simplicité** - l'art de maximiser la quantité de travail à ne pas faire - est essentielle.
11/ Les meilleures architectures, spécifications et conceptions sont issues d'**équipes qui s'auto-organisent.**
12/ À intervalle régulier, l'équipe **réfléchit aux moyens de devenir plus efficace**, puis accorde et ajuste son comportement dans ce sens.

# Plusieurs méthodes agiles

- *«Agile method»* = méthode agile ou leste !
  Le processus est léger, allégé…
  par opposition au processus lourd
- Variétés:
  - *XP = eXtreme Programming* (1999, Kent Beck) *axée sur la construction d'une application?*
  - *SCRUM : (mêlée du rugby) : met en avant la pratique des réunions quotidiennes*
  - *RUP : ( Rational Unified Process) Vue globale, processus plus adapté aux gros projets.*
  - *DSDM (95) (Dynamic Systems Development Method) : évolution du RAD : spécialisation des acteurs*
  - *…*
- *http://www.agilealliance.com/resources/*

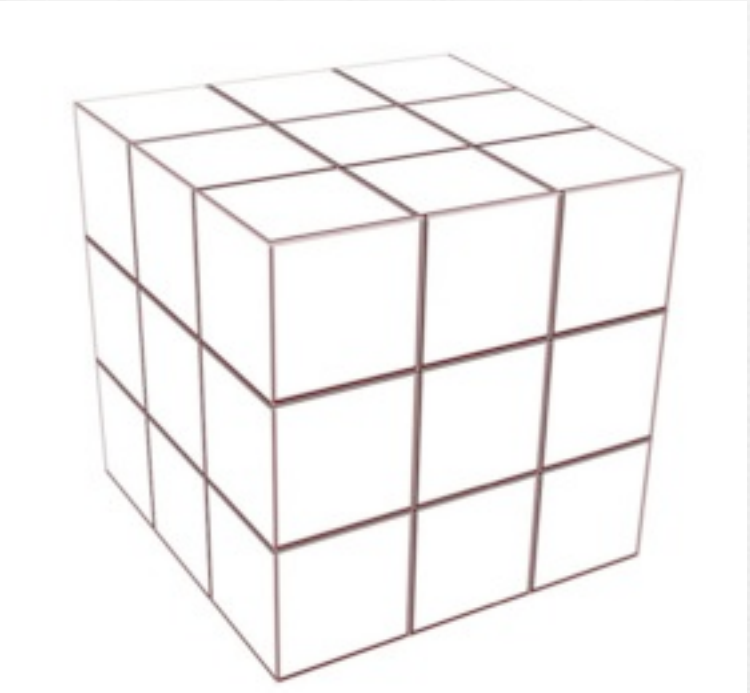
# Extreme Programming XP

Les grands principes uniquement

Merci à tous ceux qui ont rendu leurs cours et exposés disponibles sur le web & dans les livres, voir Biblio.

# Les problèmes ciblés par XP

Défauts



# Les problèmes ciblés par XP

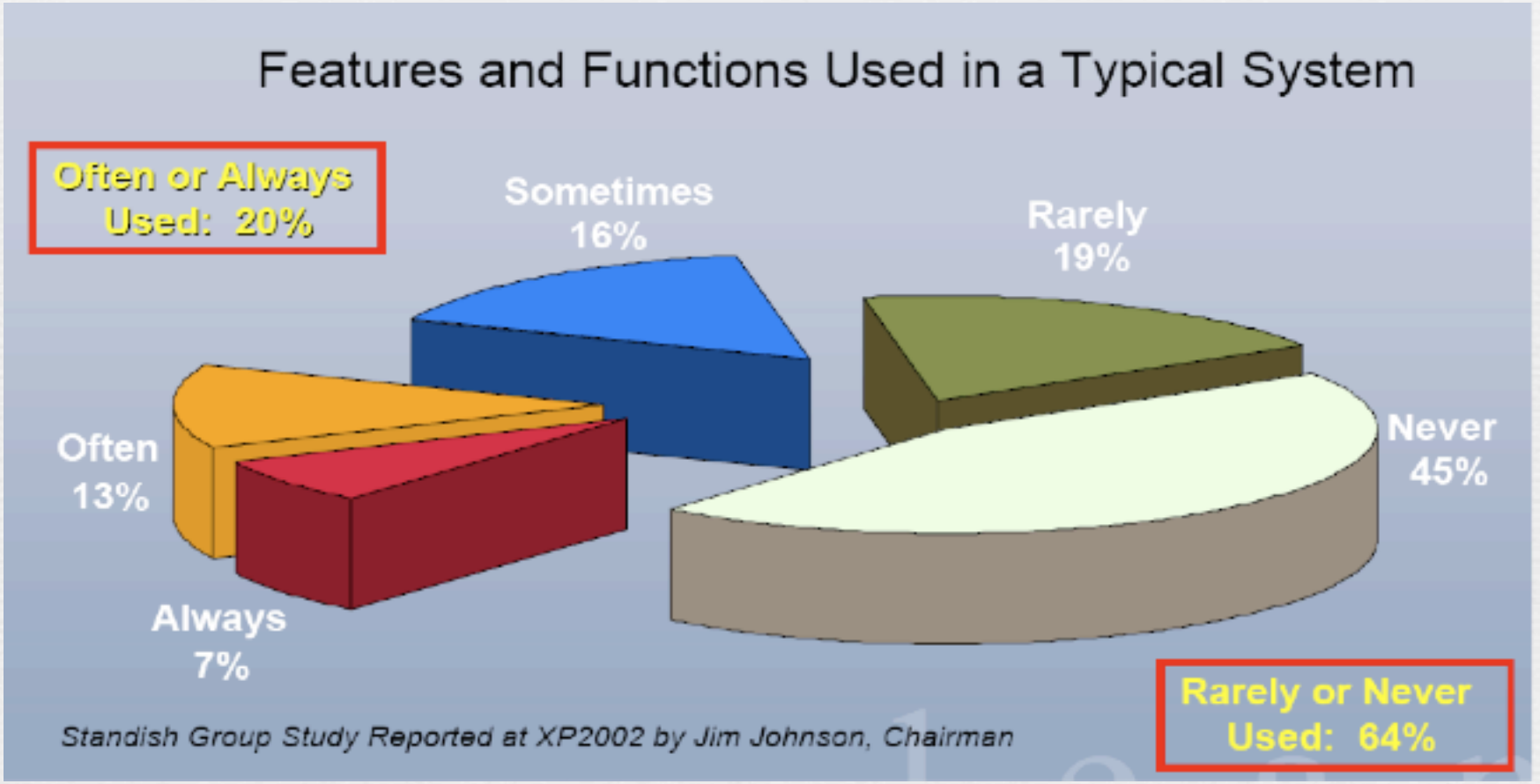
Travail non terminé

# Les problèmes ciblés par XP

Dérives... : des fonctionnalités jamais utilisées

# Les problèmes ciblés par XP

## Dérives... : des fonctionnalités jamais utilisées



# Les problèmes ciblés par XP

# eXtreme Programming (XP)

« Comment mieux travailler avec le client pour nous focaliser sur ses besoins les plus prioritaires et être aussi réactifs que possible ? »
Pour des petits projets (moins de 10 personnes)

# eXtreme Programming (XP)

« Comment mieux travailler avec le client pour nous focaliser sur ses besoins les plus prioritaires et être aussi réactifs que possible ? »
Pour des petits projets (moins de 10 personnes)

**Valeurs d'XP :**

1. Communication
2. Feedback
3. Simplicité
4. Courage

# Les valeurs de l'Extreme Programming

- Communication
  - entre tous les partis
  - «verbal, with a collocated team.»
- «Feedback»
  - tests essentiels & clients
  - «continuous from the customer»
- Simplicité
  - par dessus tout…
  - «always do the simplest thing that can possibly work.»
- Courage
  - «willingness to change».
  - Améliorer (les tests aident!)

# Communication
## Verbale, et équipe collocalisée

- Contacts réguliers avec le client
- User stories (Use Cases ... )
- Mêlée journalière
- Programmation en Paires – do it in pairs
- Tests Unitaires– make test cases –
- Experimenter – try it!

# Communication
## Verbale, et équipe collocalisée

- Contacts réguliers avec le client
- User stories (Use Cases ... )
- Mêlée journalière
- Programmation en Paires – do it in pairs
- Tests Unitaires– make test cases –
- Experimenter – try it!

- *Réduit* les risques d'incompréhension;
- Cependant les détails sont perdus s'ils ne sont pas écrits.

.

# «Feedback»
# du code et du client

- Consulter régulièrement le client
- Tester vos codes

# «Feedback»
# du code et du client

- Consulter régulièrement le client
- Tester vos codes

*Allez demander au client qui est assis là-bas :*

*Même (surtout?) en conception:*

- *Construire un prototype de premier ordre*
- *Testez-le avec le client*
- *Ajustez-le jusqu'à ce qu'il soit conforme à vos attentes.*
- *Un peu de rétroaction peut remplacer beaucoup de travail analytique.*

# Simplicité
## Verbale, et équipe collocalisée

- “Quelle est la chose la plus simple qui est utile?”
- Construit là !

- You Ain’t Gonna Need It (YAGNI).

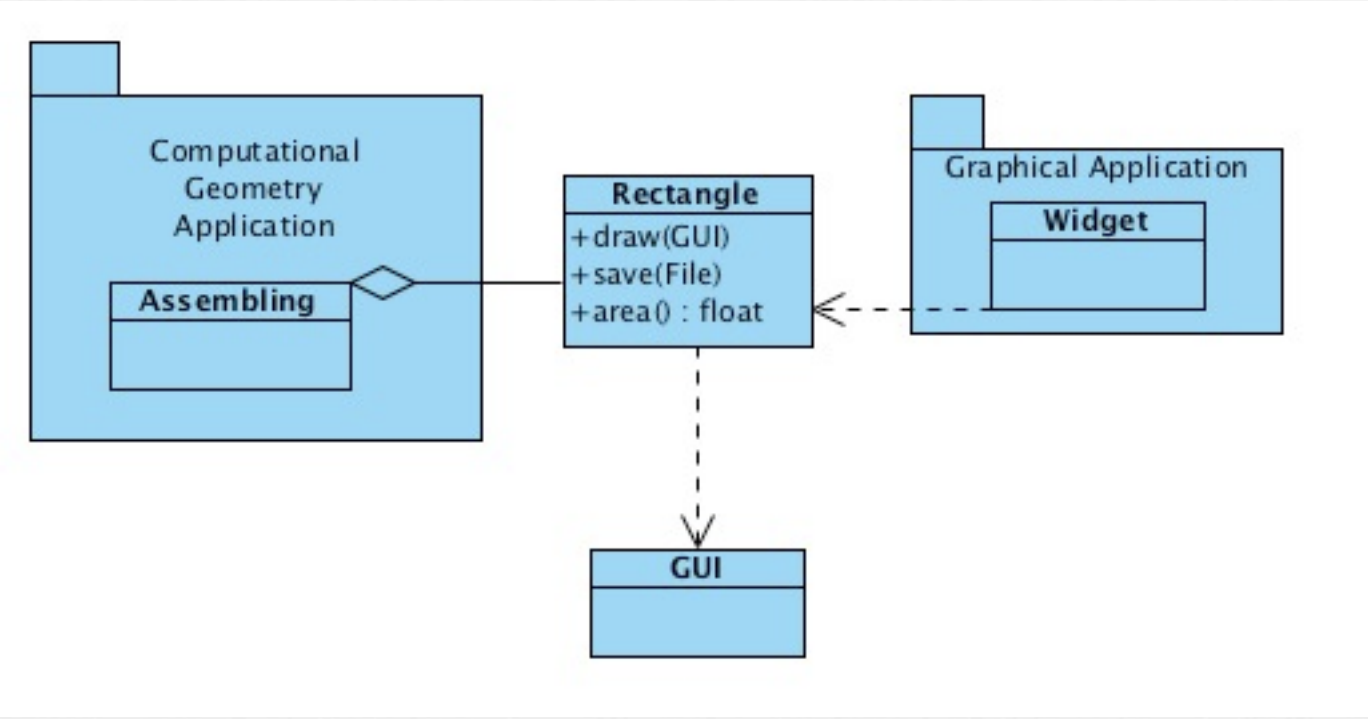
- Absolute simplicity isn’t always the best solution, so be as simple as possible but not too simple.

**YAGNI** (anglicisme, acronyme anglais de *You ain't gonna need it*) est un principe d'extreme programming qui déclare que les programmeurs ne devraient pas ajouter de fonctionnalité à un logiciel tant que celle-ci n'est pas absolument nécessaire[1]. Ron Jeffries recommande par ailleurs : « mettez toujours en œuvre les choses quand vous en avez effectivement besoin, pas lorsque vous prévoyez simplement que vous en aurez besoin »[2]. **Wikipedia**

# Principes de Simplicité

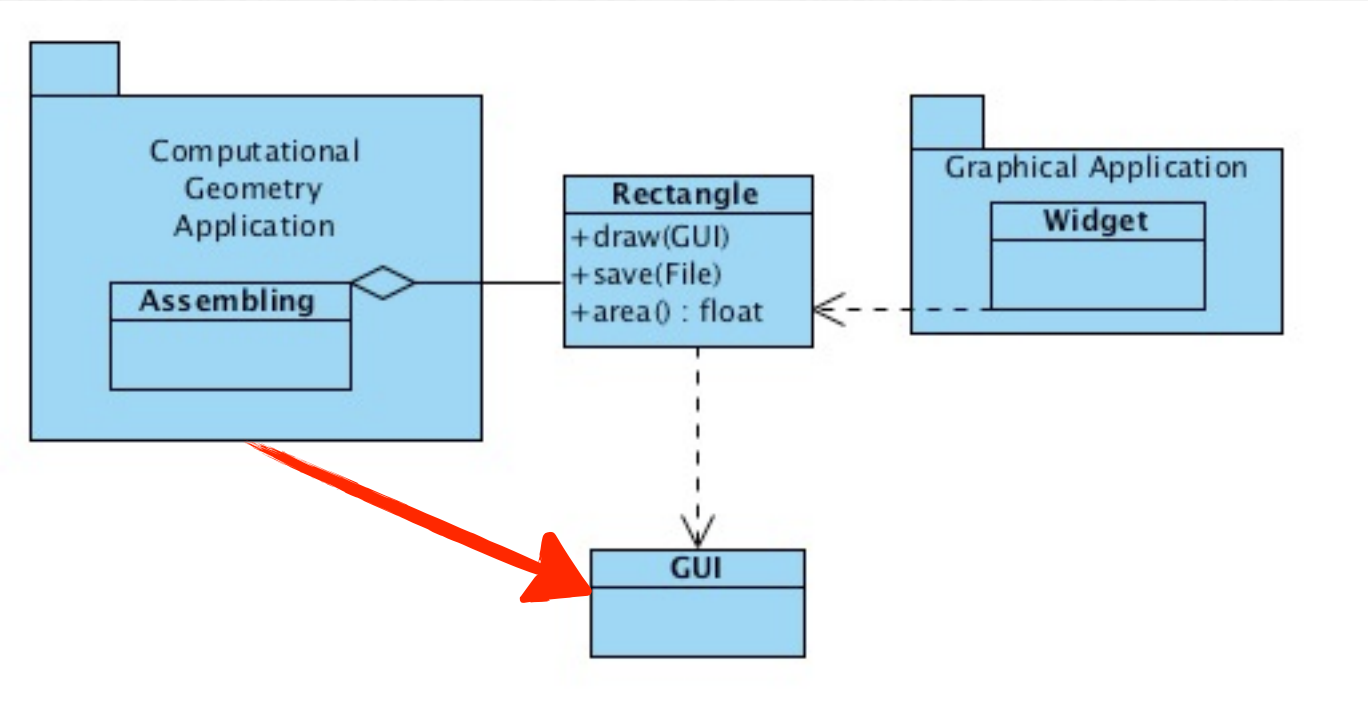
- Pas de codes complexes "intelligents"- sauf si c'est absolument essentiel
- Les classes portent une unique responsabilité.
- Les fonctions ne font qu'une et une seule chose (philosophie Unix)
- Le code devrait être évident et contenir un minimum de commentaires vraiment utiles!
  - Les commentaires sont souvent "évidents" et polluent le code
  - Les commentaires peuvent être dangereux - Les développeurs modifient les codes et laissent les commentaires devenus obsolètes.
  - ..... ATTENTION A RESPECTER CEPENDANT LES REGLES DONNEES.....

# Simplicité => Responsabilité Unique




http://www.objectmentor.com/resources/articles/srp.pdf

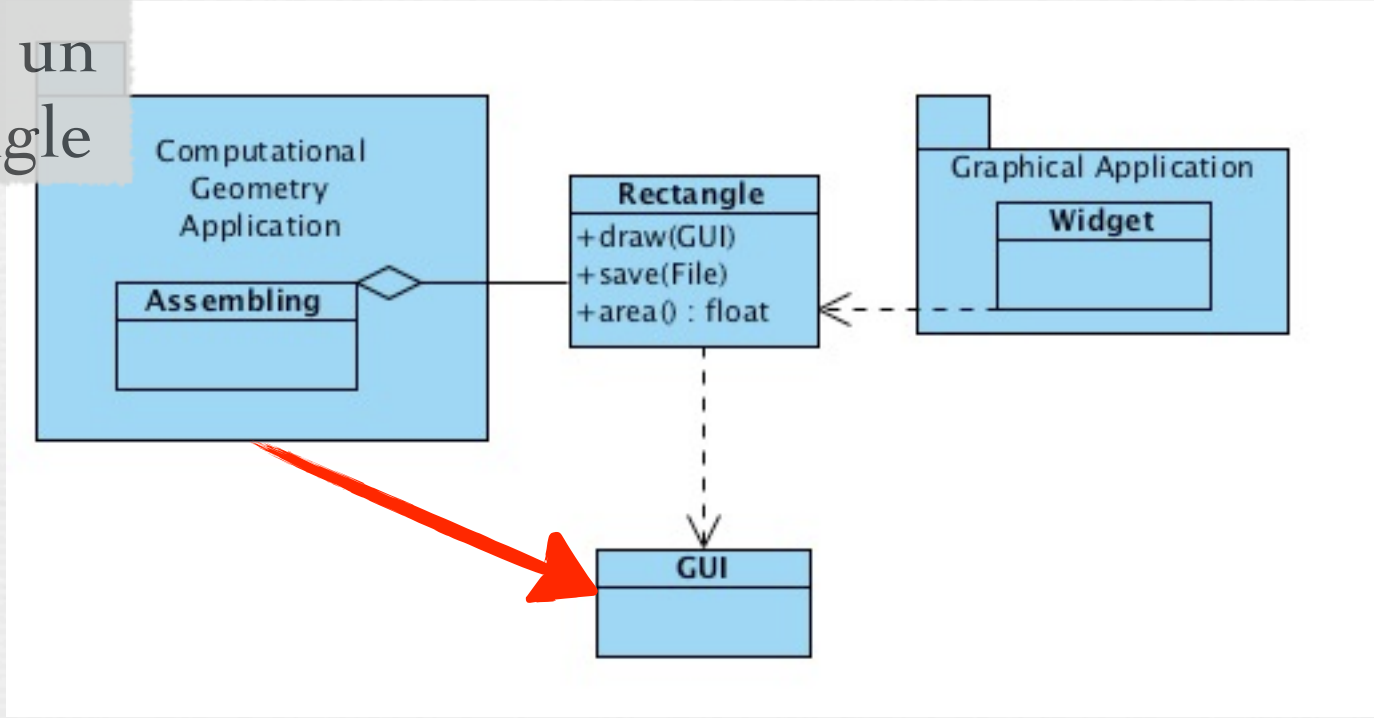
# Simplicité => Responsabilité Unique




http://www.objectmentor.com/resources/articles/srp.pdf

# Simplicité => Responsabilité Unique

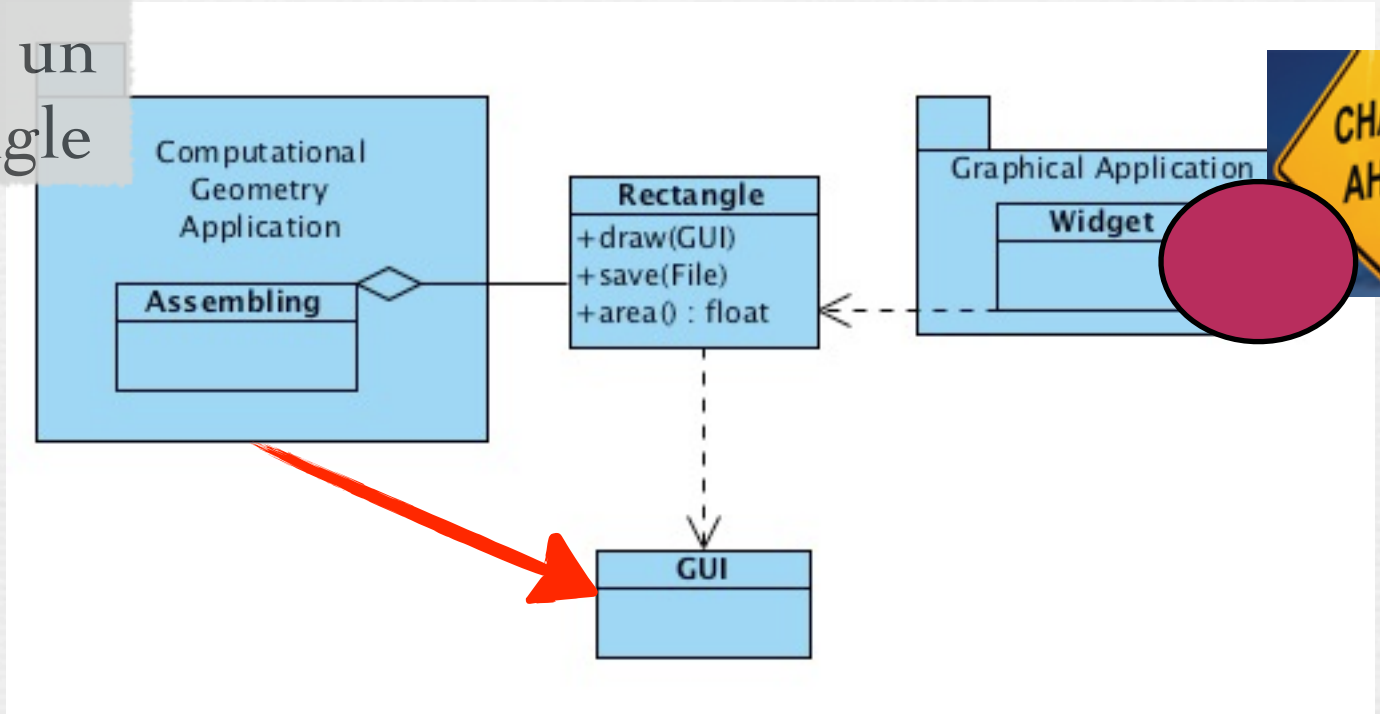
Ne dessine jamais un rectangle




http://www.objectmentor.com/resources/articles/srp.pdf

# Simplicité => Responsabilité Unique

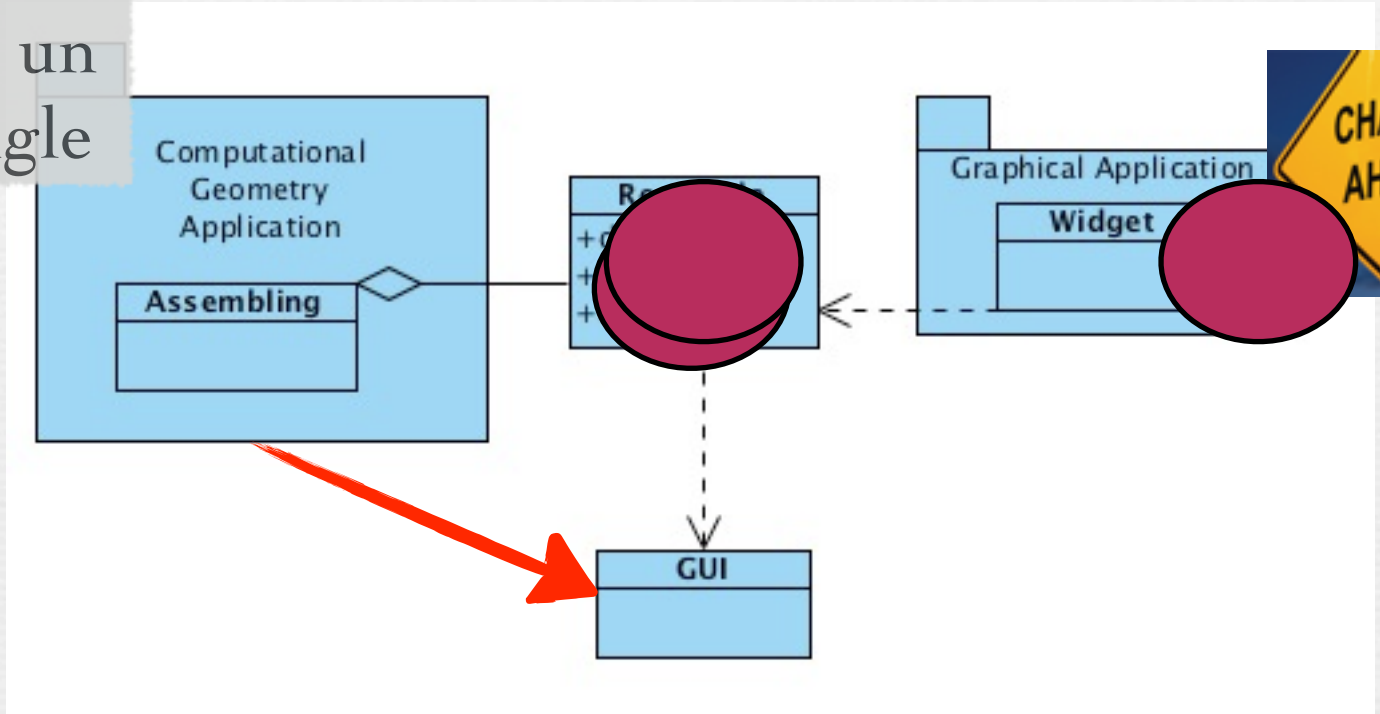
Ne dessine jamais un rectangle




http://www.objectmentor.com/resources/articles/srp.pdf

# Simplicité => Responsabilité Unique

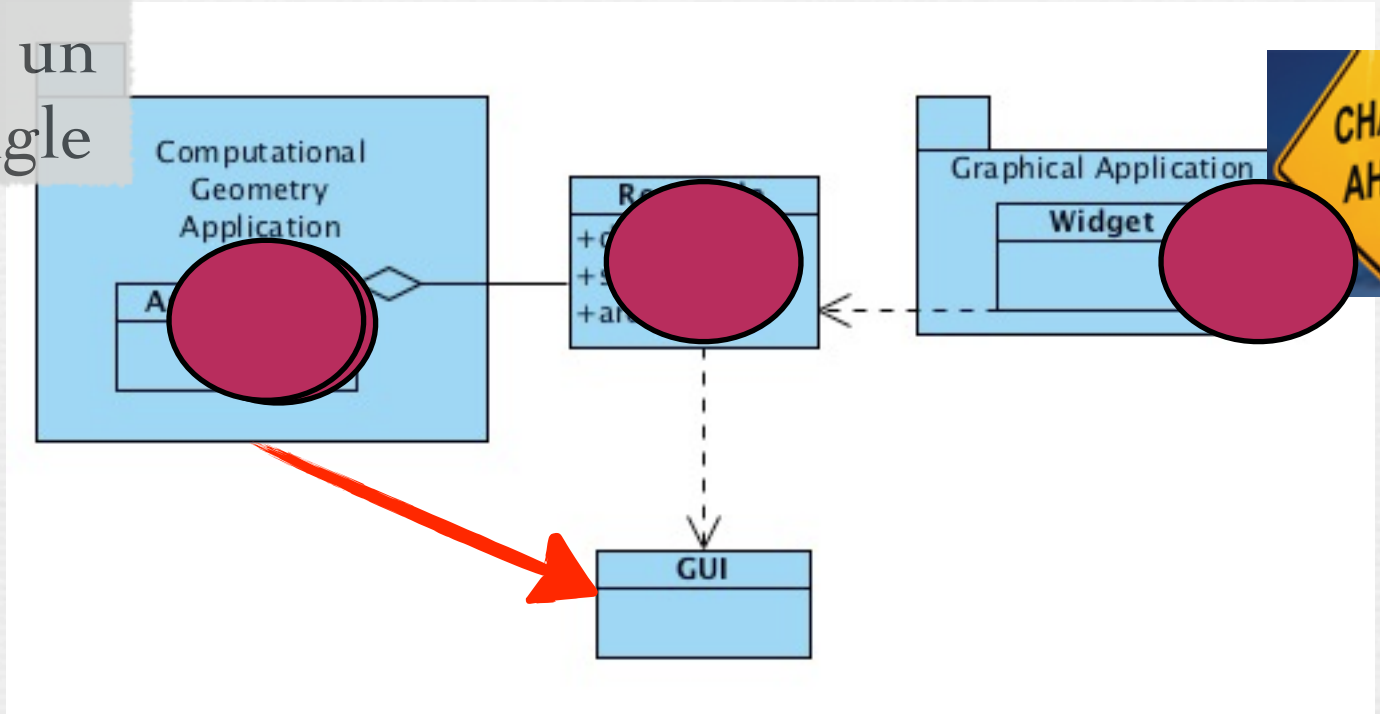
Ne dessine jamais un rectangle




http://www.objectmentor.com/resources/articles/srp.pdf

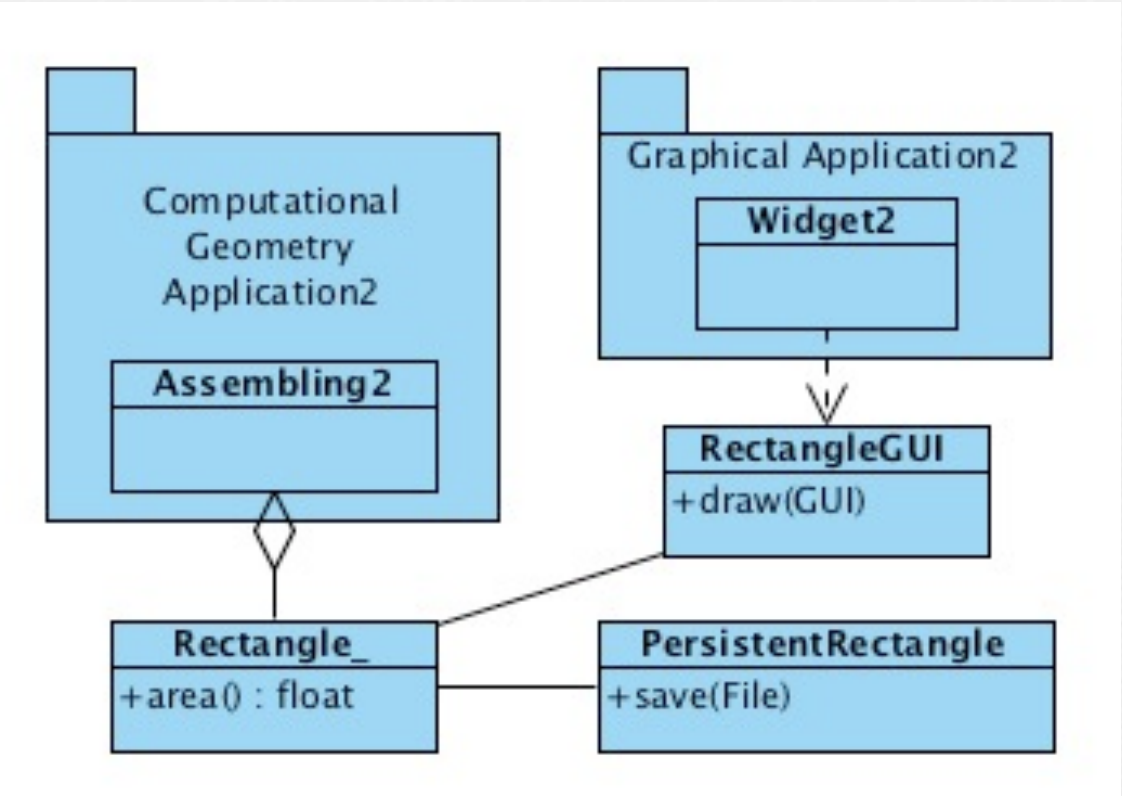
# Simplicité => Responsabilité Unique

Ne dessine jamais un rectangle

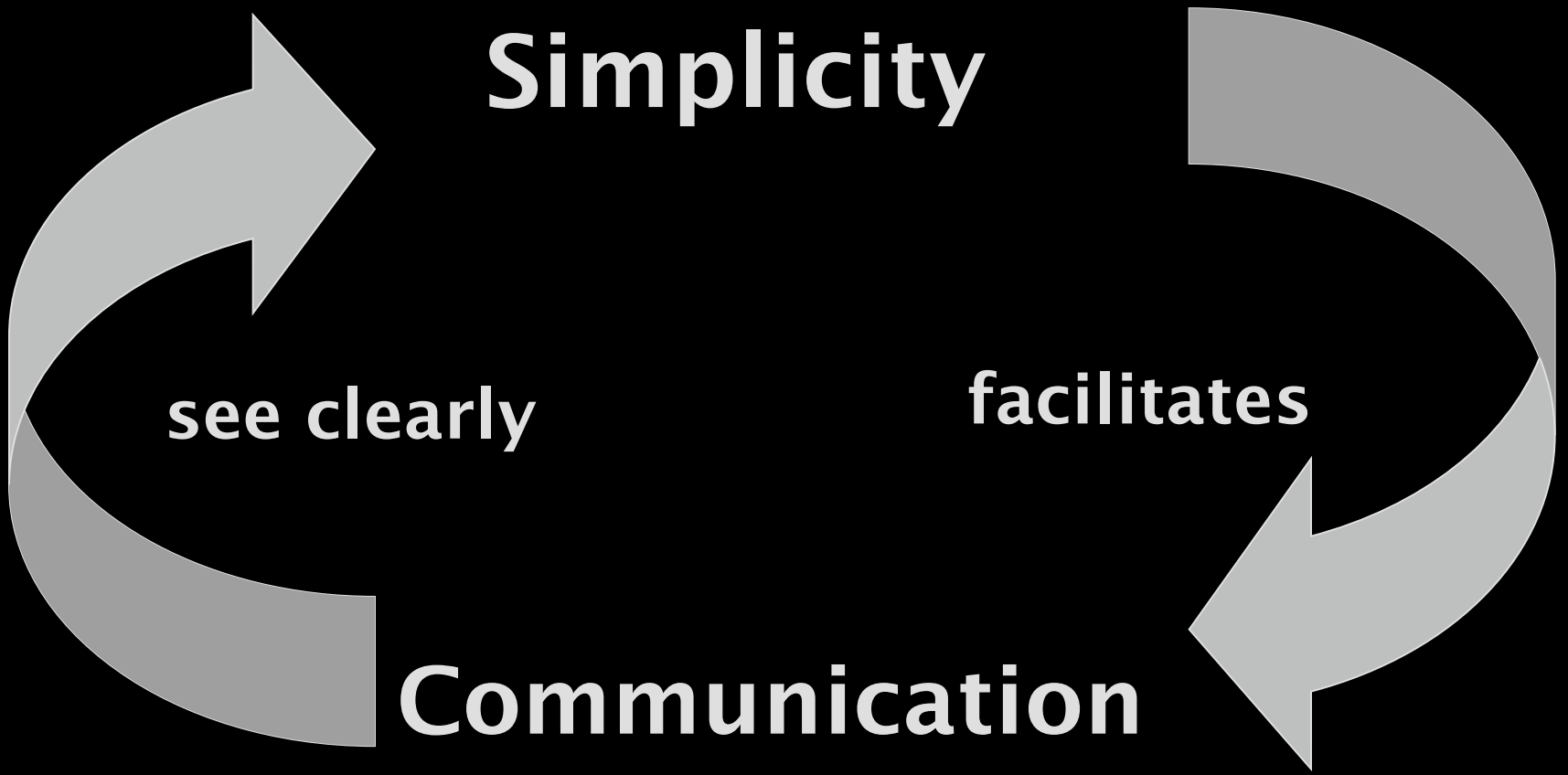



http://www.objectmentor.com/resources/articles/srp.pdf

# Responsabilité Unique



# Simplicity supports Communication



# «Ayez le Courage»

- Corrigez!
- Jetez!
- Essayez!

# «Ayez le Courage»

- Corrigez!
- Jetez!
- Essayez!

  - *Soyez prêt à faire des changements de conception quand ils ont utiles.*
  - *Aidez vous d'une suite de tests automatiques pour vérifier que vous ne «cassez» rien.*

# XP *Practices*

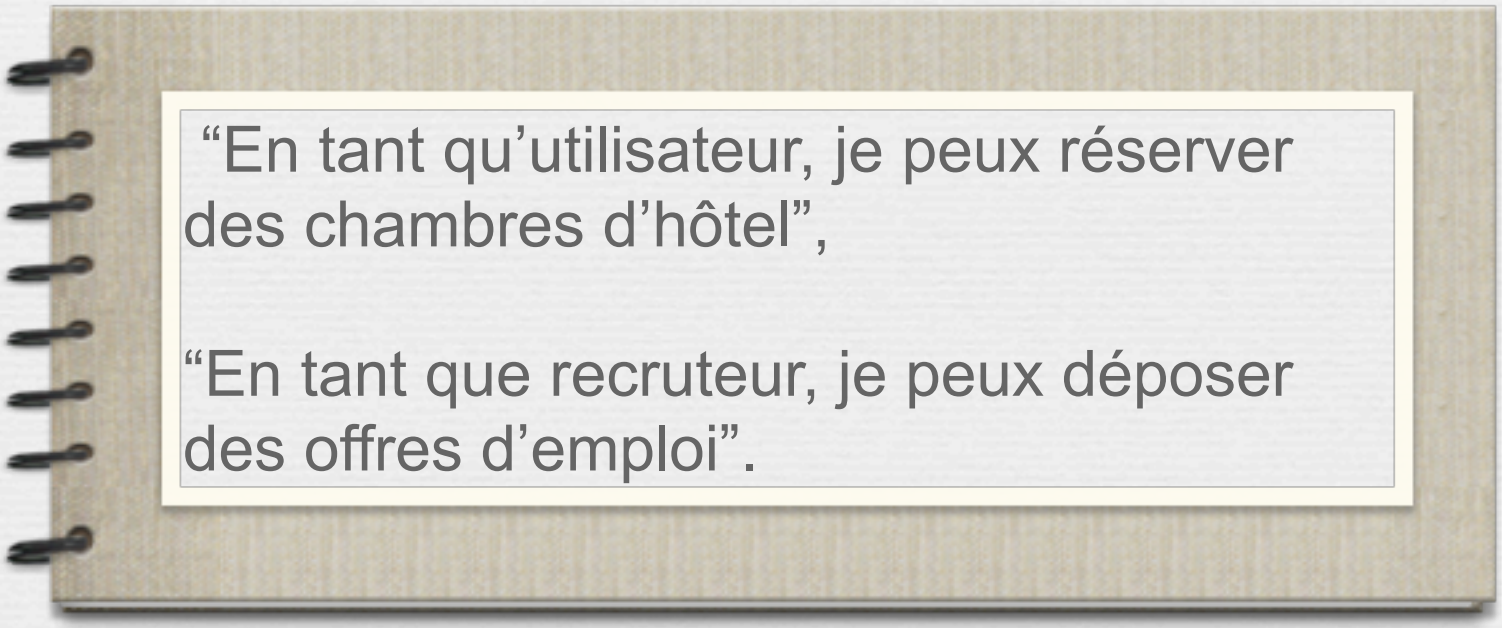
- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Planning*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# User stories / (Récits ou histoires d'utilisateur)

- Une ou deux phrases résument ce que veut l'utilisateur
- Décrit comment le système est sensé travailler




Lorsqu'une expression du besoin existe en UML, elle peut être utilisée

# Planning

- Planning : consensus entre le client et le développeur.
  - priorités : compromis entre la valeur ajoutée des user stories et la complexité de développement.
  - Développement uniquement sur les buts principaux
  - le traitement des objectifs secondaires est différé.
- **Technique** : Planning Game.

http://www.rad.fr/xpcv.htm

# Planning Game
## Le jeu de la planification

- Choix d'un rythme de livraison (2-3 mois)
- Choix du rythme des itérations (1 à 3 semaines)
- **Étapes :**

**1. Le client décrit ses besoins: user stories**

Un projet peut comporter une centaine de user stories.

Il estime la priorité des user stories.

**2. Les développeurs estiment le coût d'implémentation**

En équipe avec la collaboration du client. On note en "semaines idéales". Une histoire trop grande (>3 points) doit être partagée. Si on ne sait pas estimer, on explore le sujet en faisant un essai ("spike").

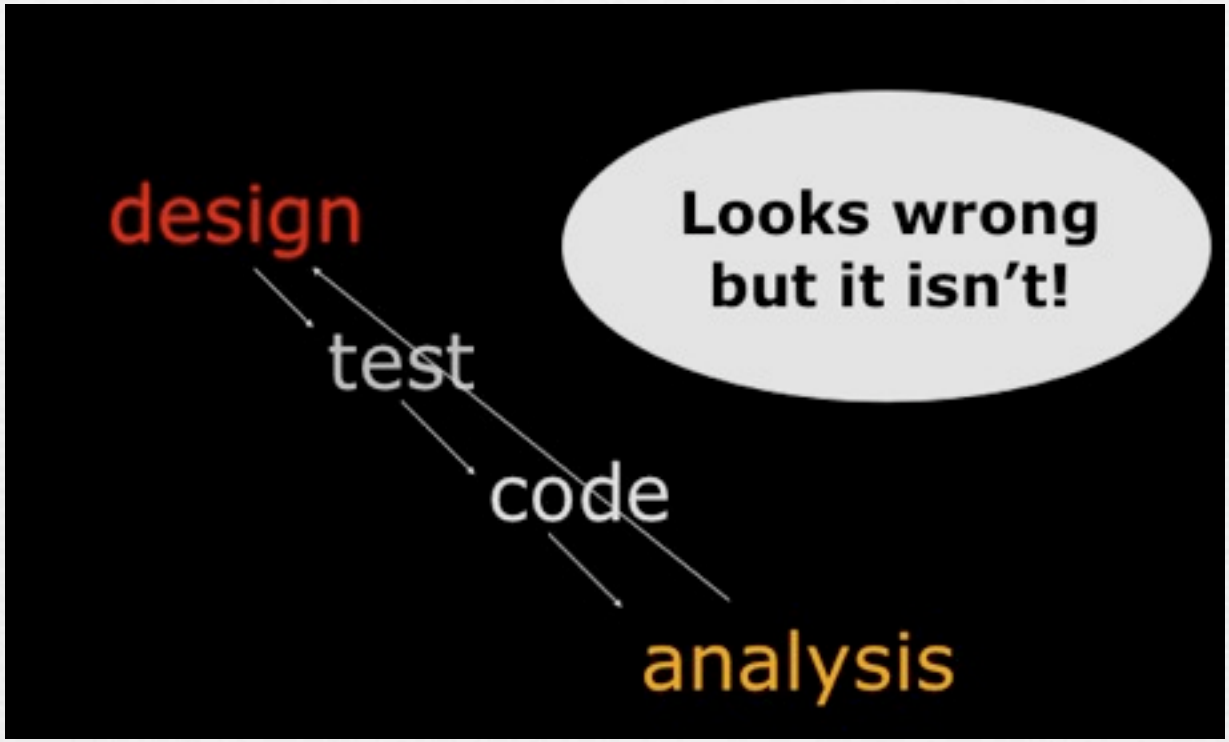
# Des cycles courts

- Production rapide de versions limitées de l'application.
- Itérations structurées en étapes : *spécifications courtes, développement, tests, retour du client*
- Réduction du risque d'incompréhension
- Modifications fréquentes mais concentrées dans un cycle très court.
- **Technique** : livraisons fréquentes. Elles permettent un feedback immédiat, tout en offrant des fonctionnalités validées pouvant être utilisées. Fréquence de livraison hebdomadaire.



http://www.rad.fr/xpcv.htm

# XP Cycle (in the small)




Martin (2003): 'programmers work in very short cycles, adding a failing test, then making it work.'

# XP *Practices : Designing*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Designing*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Designing*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Designing*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# Un design simple

Le design est :

- simple
- focalisé sur les besoins actuels dans l'ordre de leurs priorités.

**Technique** :

- Planification initiale basée sur la valeur ajoutée des fonctionnalités attendues.

- Livraisons en fonctionnalités réduites.

- Respect de règles de mises en oeuvre

http://www.rad.fr/xpcv.htm

# Refactoring du code

- Le code est remanié continuellement et progressivement.

- **Objectifs :** Gérer la détérioration du code (**code rot)**

**Software rot**, also known as **code rot** or **software erosion** or **software decay** or software entropy, is a type of bit rot. It describes the perceived slow deterioration of software over time that will eventually lead to it becoming faulty, unusable, or otherwise in need of maintenance. This is not a physical phenomenon: the software does not actually decay, but rather suffers from a lack of being updated with respect to the changing environment in which it resides. Wikipedia

# Refactoring du code

- Le code est remanié continuellement et progressivement.

**Technique** : Refactoring par amélioration continue de la qualité du code *sans en modifier le comportement.*

Le résultat du " nettoyage " s'effectue régulièrement et se valide lors de séances de travail collectif impliquant toute l'équipe.



http://www.rad.fr/xpcv.htm

# Refactoring du code (ex)

- Fusionner/Supprimer les codes dupliqués
  - exemple : gestion des commentaires (feedback, comments, trackbacks) => (regardez les contrôleurs)
  - «Dans le cas d'un nouvel enregistrement, nous avions @objet = Objet.new, et dans le cas d'une mise à jour, @objet = Objet.find(params[:id]). Nous avons rassemblé tout le code en double dans une même méthode de traitement appelée depuis les méthodes de création et d'édition, avec l'objet instancié en fonction du besoin.»
- Supprimez le code mort (code jamais appelé dans l'application).
- Élargissez la couverture de tests
- Supprimez les fonctionnalités inutiles

http://t37.net/massacre-a-ide-ou-les-joies-du-refactoring-par-le-vide.html

# XP *Practices : Coding*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Coding*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Coding*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Coding*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

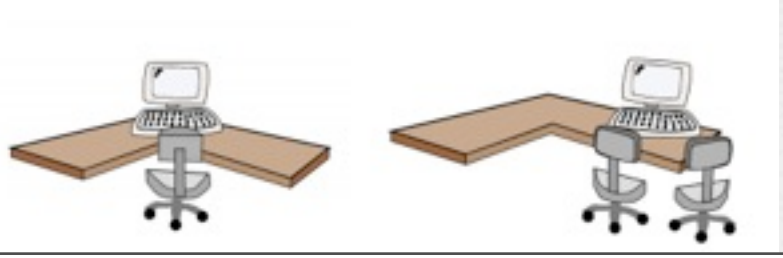
# Prog. en binôme

La programmation en binôme (pair programming) est une pratique où 2 programmeurs

- travaillent côte à côte
- utilisent le même ordinateur (ou la même feuille de papier)
- réfléchissent sur la même spécification, le même algorithme, le même code ou le même test

*Cela fonctionnera? Existe-t-il une meilleure solution? Qu'est-ce qui peut être cassé? Comment pourrions-nous faire plus simplement?*

Echanges de connaissances, moins de bugs, surtout pour les codes les plus difficiles



# Prog. en binôme

La programmation en binôme (pair programming) est une pratique où 2 programmeurs

- travaillent côte à côte
- utilisent le même ordinateur (ou la même feuille de papier)
- réfléchissent sur la même spécification, le même algorithme, le même code ou le même test

*Cela fonctionnera? Existe-t-il une meilleure solution? Qu'est-ce qui peut être cassé? Comment pourrions-nous faire plus simplement?*

Echanges de connaissances, moins de bugs, surtout pour les codes les plus difficiles

perte de temps au pire de 15% , mais 15% de bugs en moins

# Standards de codage

- Pour faciliter l'appropriation collective de l'applicatif, la réutilisation et la communication, les programmeurs codent dans un style et des règles identiques (normes de nommage pour les variables, méthodes, objets, classes, fichiers, etc.).
- **Technique** : Standards de codage, frameworks, design patterns, convention de nommage, etc.

http://www.rad.fr/xpcv.htm

# XP *Practices : Testing*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Testing*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Testing*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Testing*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# La place des tests

- Un logiciel XP est testé et validé en permanence.
- Avant d'implémenter une fonctionnalité, un test est écrit.
- Technique : Test-Driven Development

http://www.rad.fr/xpcv.htm

# Intégration continue

- Assemblage journalier des codes développés.
- Chaque nouvelle implémentation s'appuie sur un applicatif stabilisé.

http://www.rad.fr/xpcv.htm

# Acceptance Tests (Tests fonctionnels)

**Objectifs :** vérifier de manière automatique chacune des fonctionnalités demandées par le client.

➡Le client définit ces tests et participe éventuellement à leur implémentation, assisté pour cela d'un certain nombre de testeurs.

***Exemple*** :

- jeux de données : « pour telle entrée, le logiciel doit produire tel résultat ».
- scripts décrivant des séquences d'interactions de l'utilisateur avec l'interface graphique du produit.

Dans un contexte « pur XP » : seules spécifications.

il n'y a pas de *document* de spécifications à proprement parler. En pratique, cependant, des documents peuvent être exigés par l'organisation qui encadre le projet.

# XP *Practices : Listening/Team*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Listening/Team*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Listening/Team*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design

- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# XP *Practices : Listening/Team*

- Test-driven development
- User stories
- The planning game
- Whole team
- Short cycles
- Metaphor
- Simple design
- Refactor mercilessly
- Collective ownership
- Pair programming
- Continuous integration
- Sustainable pace
- Coding standards
- Acceptance tests
- (Emergent design)

# Une équipe «complète» / Whole Team

- «Within XP, the "customer" is not the one who pays the bill, but the one who really uses the system.»

- Présence permanente du client disponible pour des questions.

# Appropriation collective du code

- Responsabilité collective
  - ✓ Chacun est supposée avoir connaissance de l'intégralité du code.
  - ✓ La qualité de l'ensemble du code est de la responsabilité de l'ensemble des programmeurs.
  - ➡ Améliore la qualité effective du code, la réutilisation, la compréhension des interfaces
  - ➡ Supprime les principaux problèmes de turnover.

**Technique** : Les développeurs réorganisent fréquemment les binômes, ce qui permet d'améliorer la connaissance collective de l'application et la communication au sein de l'équipe



http://www.rad.fr/xpcv.htm

# Possession Collective du Code

- Qui que ce soit qui trouve un bug, le corrige.
  - Les tests unitaires garantissent les fonctionnalités
- Propriété collective du code et Refactoring dirigent :
  - D'autres améliorent, sous-classent, étendent ou utilisent le code produit.

Martin (2003): “Any pair of programmers can improve any code at any time.”

# Rythme soutenable

- Les semaines n'ont que 40 heures et les ressources fatiguées font plus d'erreurs toujours plus coûteuses à corriger a posteriori.
- **Technique** : Planning individuel n'impliquant pas d'heures supplémentaires

http://www.rad.fr/xpcv.htm

# Début typique d'une journée

- À l'arrivée:
  - coup d'œil sur les tests de régression de la nuit
- L'équipe entière se réunit debout en cercle (<¼ heure):
  - Chaque coéquipier à tour de rôle décrit:
    - son travail de la veille
    - ce qu'il prévoit faire aujourd'hui
    - et les problèmes ouverts…
  - C'est tout ! Pas de discussion dans le cercle
- Des discussions entre certains coéquipiers sont ensuite informelles
- Les binômes vont développer

# Let's Take an Example

I set a Coursework some years ago:

www.happydate.com

http://www.cs.nott.ac.uk/~dge/G52SEM/

# Une agence de rencontres

Story 1

**Une personne s'inscrit à l'agence en fournissant des renseignements personnels, informations au sujet du genre de personne recherché, un alias pour cacher leur véritable identité, et des coordonnées confidentielles.**

# Une agence de rencontres

Story 2

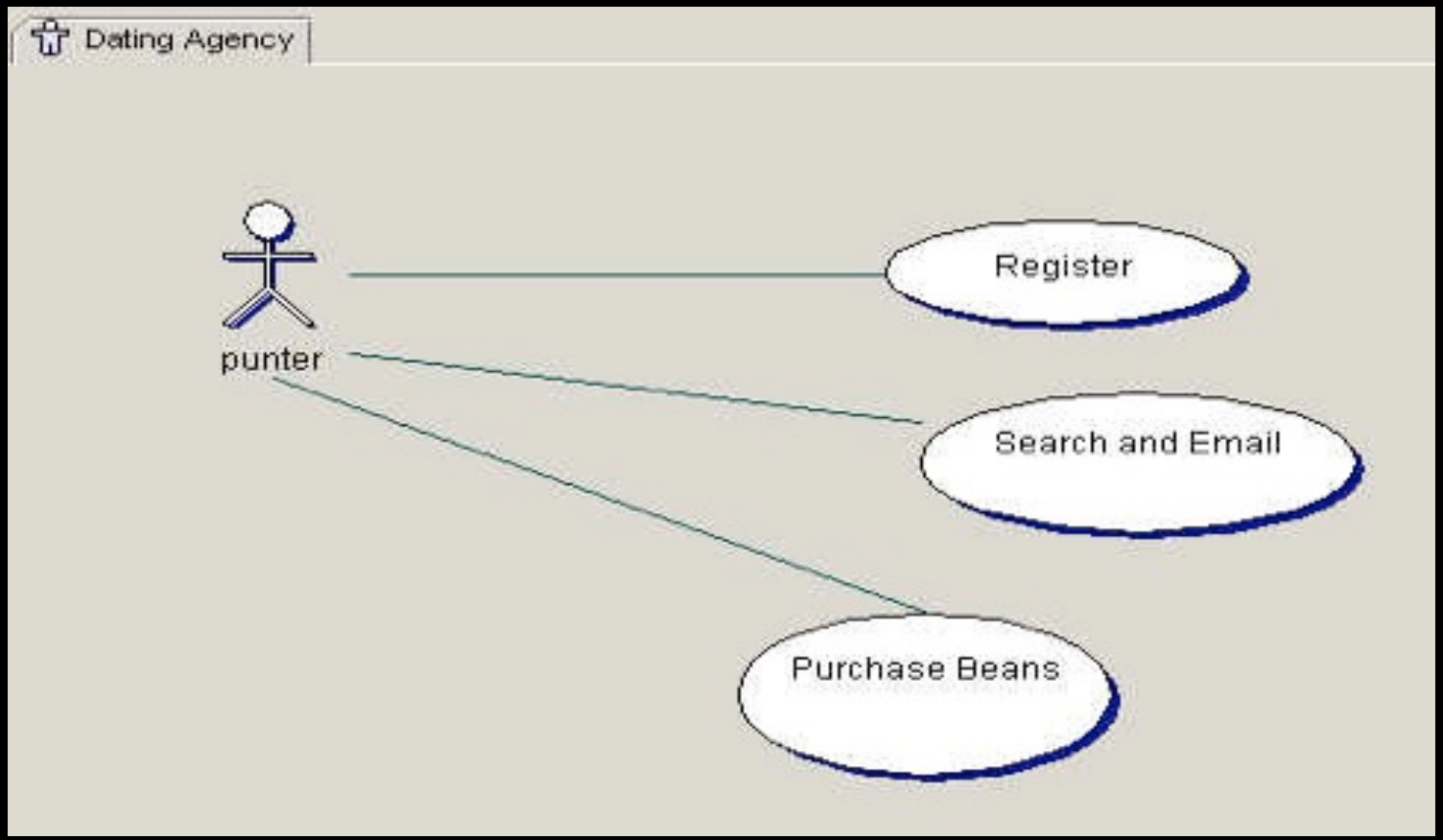
**Une sélection de «clients correspondant» est affichée, et un e-mail peut être envoyé à l'un d'entre eux. Des frais seront encaissés pour ce service.**

# Une agence de rencontres

Story 3

**Afin d'être capable d'envoyer par courriel, une personne doit avoir«des points» pour envoyer. Ceux-ci peuvent être achetés en ligne.**

# Use Cases


Dating Agency
punter
Register
Search and Email
Purchase Beans

# Quelle est la chose la plus simple et la plus utile ?

- Abandonner l'encaissement?
  - Oui – pour obtenir une clientèle initiale
- Quel est le plus simple appariement qui pourrait être acceptable?
  - Sexes différents
  - Différence d'âge inférieure à 12 ans
  - Distance inférieure à 80 km

# Enregistrer un membre

## Enregistrer un membre

- Get Email address
- Get Sexe
- Get Age
- Get Postcode
- Get Nom, Prénom
- Get Pseudo

## Test

- is format valid? Does it exist?
- 0 >= age <= 112
- Is postcode valid?
- Is Pseudo unique ?

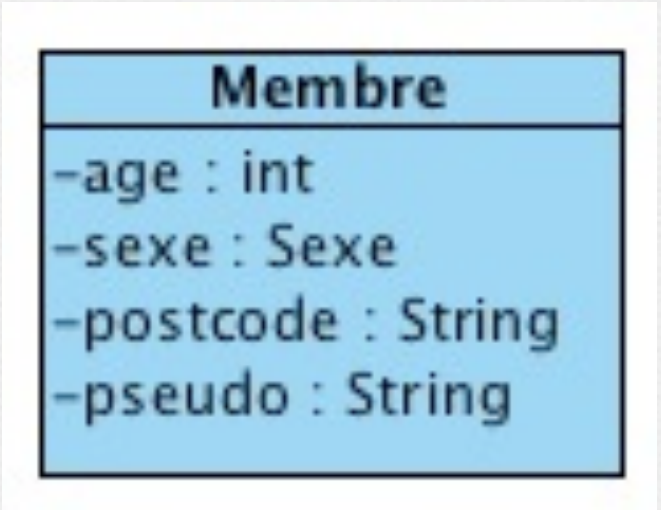
# Enregistrer un membre

## Enregistrer un membre

- Get Email address
- Get Sexe
- Get Age
- Get Postcode
- Get Nom, Prénom
- Get Pseudo

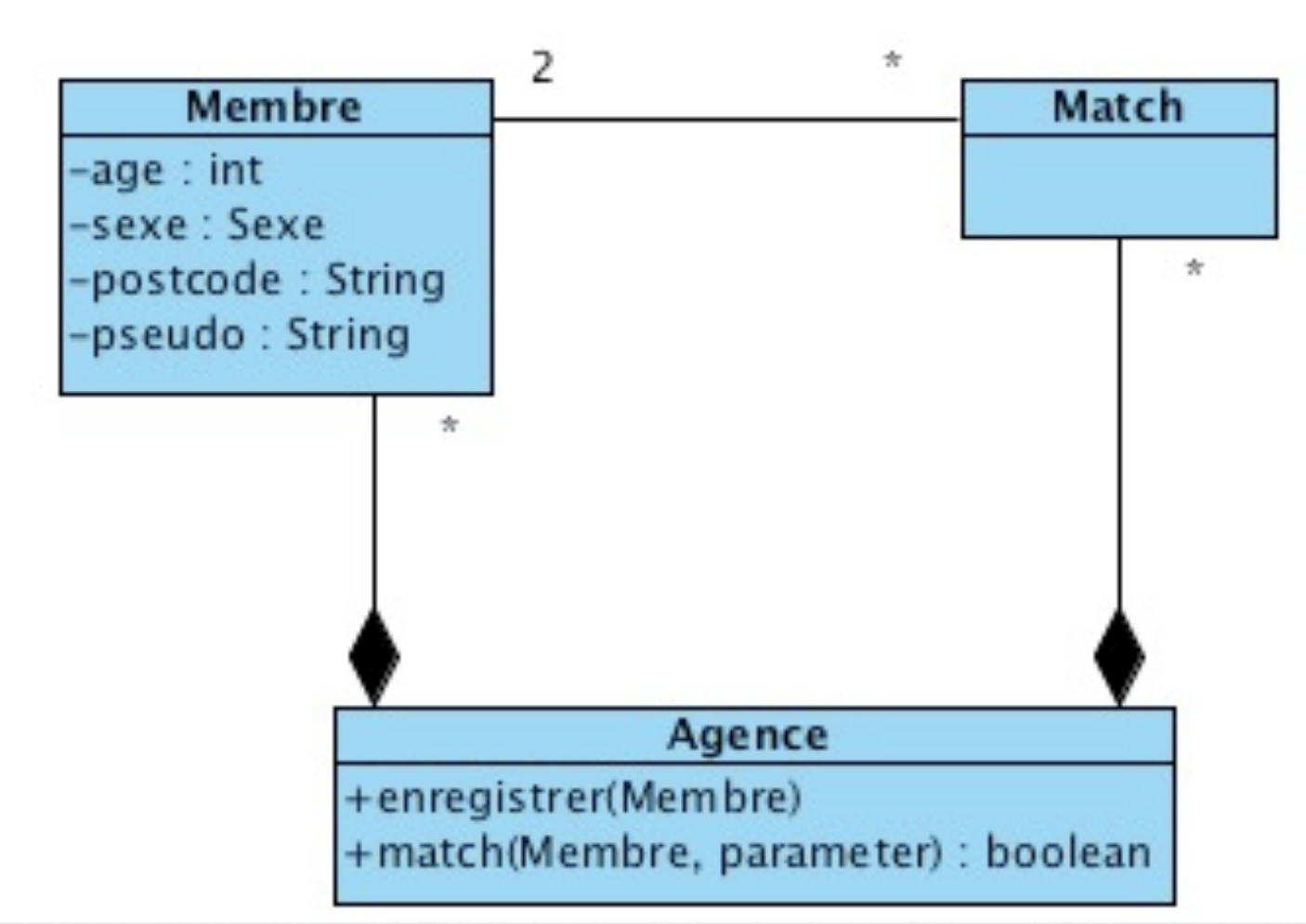
## Test

- is format valid? Does it exist?
- 0 >= age <= 112
- Is postcode valid?
- Is Pseudo unique ?



# Quels sont les objets?

- Agence
- Client
- Match



# Qu'est-ce qui manque ?

- Distance(Postcode1, Postcode2)
  - Combien cela coûte ?
- Web page and screen design
  - minimalist?
- Quand/comment partir?
  - timeout?
- Comment les données sont-elles gérées?
  - Proposer un fichier texte puis une base de données

# What's Hard about XP?

- Always doing the simplest thing
  - We like to look clever
- Admitting you don't know
  - We don't like to look stupid
- Collaborating
  - It takes time and effort
- Breaking down emotional walls about "ownership"
  - We are possessive and don't like to share things
- Creating a great software project and team means going against the grain of human psychology – be explicit about that

# Bidouilleur ou Agiliste

| Bidouilleur | Agiliste |
| --- | --- |
| Evite de planifier et ne contrôle pas son temps | Planification dynamique et point du reste à faire |
| Tests brouillon | Test Driven Development |
| Communique uniquement en situation de blocage | Communication intensive et partage de la connaissance |
| N'informe pas ses supérieurs des difficultés rencontrées | Décisions démocratiques et mode coopératif intensif |

# XP - Constats

- Très bon à s'adapter aux changements
- A des pratiques d'ingénierie très forte
- Améliore considérablement la qualité
- Élimine beaucoup de déchets provenant du processus
- Focus important sur KISS et YAGNI
- L'automatisation est la clé
- Ramène le pouvoir entre les mains du développeur

# Extrême ?

- Comme il est justement souligné sur le site de Design-Up " Au final, on peut se demander si la méthode porte bien son nom. Le terme extrême évoquant des prises de risques insensées, alors qu'en pratique XP vise une suppression systématique du risque. " Dans XP, c'est en fait le bon sens qui est poussé à son extrême !

http://www.rad.fr/xpcv.htm