SI6

MVC

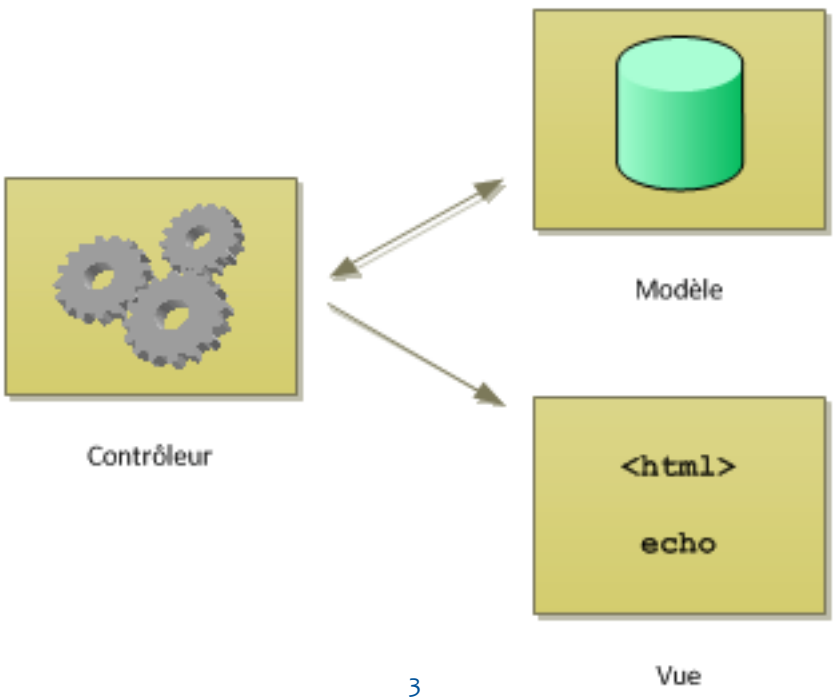
Test unitaires

Contenus

* MVC : modèle, vue, contrôleur, 3 tier
* Tests unitaires : utilité

MVC

* C’est un design pattern :



MVC

* C'est un modèle ou patron qui date des années 1980
* Répond aux besoins des applications interactives (sites internet, IHM, app, …)
* Model : les données
* View : la vue, l'interface
* Controller : le contrôleur, logique de contrôle, synchronisation

MVC

* Modèle :
  * Il gère les données, et peut être en relation avec une BD
  * Il comporte les méthodes pour agir sur les données : insertion, mise à jour, suppression, …
  * Il ne s'occupe pas de la mise en forme
  * Il n'agit pas sur le contrôleur ou la vue
  * Tous les traitements sont effectués dans cette couche

MVC

* Vue :
  * C'est l'interface utilisateur
  * Elle récupère les interactions utilisateurs à transmettre au contrôleur
  * Elle récupère les données à afficher traitées par le modèle

MVC

* Contrôleur :
  * Synchronise la vue et le modèle
  * Reçoit les événements utilisateurs
  * Envoi les consignes de modification des données au modèle
  * Informe la vue que les données ont changé et qu'une mise à jour est nécessaire

MVC

* En programmation cela implique :
  * L'utilisation de classe adéquate
  * L'utilisation de POO
  * Donc d'un langage objet : C#, ObjC
  * Xcode implémente automatiquement du MVC
  * VS n'implémente pas automatique du MVC
  * Les modifications sur un composant n'affecte pas les autres composants
  * Le développement d'une application est plus long

MVC

* Asp.net : framework .net pour site internet
* C++ : Qt
* Java : spring, swing
* ObjC : cocoa
* Perl : catalyst
* PHP : symfony
* Python : Django
* Ruby : ruby on rails

MVC

* Les framework sont à attacher à la couche concernée :
  * Vue : swing, AWT,
  * Modèle : hibernate
  * Métier (contrôleur) : EJB, javabeans, webservices, …

* Attention à ne pas mélanger couche métier et contrôleur (MVC <> 3 tier)

MVC variantes

MVC	modèle, vue, contrôleur	- simple - assigne des responsabilités aux composants	- pas de composant responsable de la logique de présentation - multiplication des contrôleurs ou contrôleur ayant trop de responsabilités
AM-MVC	modèle, vue, contrôleur, modèle d'application	- assigne les responsabilités de présentation - conserve l'état de la vue	- utilise des événements spécifiques pour modifier la vue
MVP Supervising Controller	modèle, vue, présentateur	- permet de modifier la vue sans événement spécifique - conserve l'état de la vue	- couple fortement le présentateur à la vue
MVP Passive View	modèle, vue, présentateur	- permet de modifier la vue sans événement spécifique - conserve l'état de la vue - découple la vue du modèle	- couple fortement le présentateur à la vue
Presentation Model	modèle, vue, modèle de présentation	- découple le présentateur et la vue - conserve l'état de la vue - utilise des événements génériques (PropertyChanged).	- couplage des logiques de présentation et de cas d'utilisation
MVPC	modèle, vue, contrôleur, présentateur	- découple la logique de cas d'utilisation et la logique de présentation	- complexe - couplage présentateur-vue

Avenir du MVC

* MVC : quelques faiblesses (pas de lien vue-modèle)
* MVVM : moins de code que MVC est aussi lisible
* MVP : résout la faiblesse MVC identifiée, remplace le contrôleur par un présentateur (tout un programme !)

* Microsoft ne semble plus croire à ce concept MVC et travaille sur MVVM (model view viewModel)

MVC et 3 tier

* Attention à ne pas mélanger les concepts de génie logiciel
* MVC :
  * design pattern (motif de conception), certain le considèrent comme un architectural pattern (motif d'architecture)
* 3 tier :
  * Sépare également une application en 3 parties distinctes

MVC et 3 tier

* 3 tier :
  * User interface : partie présentation de l'application
  * Business logic : couche métier qui s'occupe du traitement de l'information
  * Data Access : partie accès et stockage des données

MVC et 3 tier

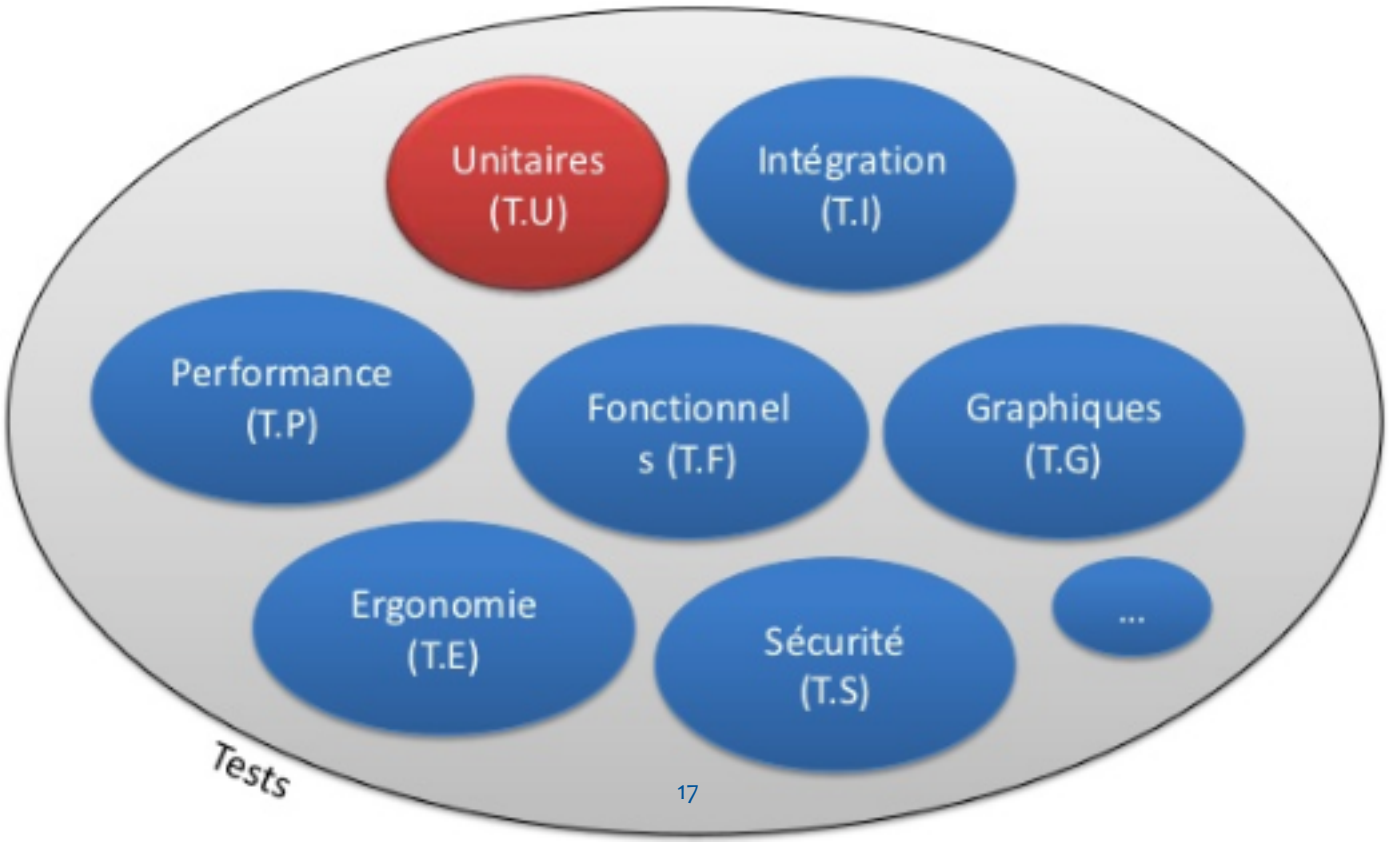
* Pour que MVC = 3 tier
  * Il faut ajouter à MVC une couche d'abstraction d'accès aux données (DAO : Data Access Object)
* Pour que 3 tier = MVC
  * Il faut ajouter à 3 tier une couche de contrôle entre User interface et Business Logic
* Donc, dans les deux cas il faut 4 couches !!!
  * MVC => MVCDAO !
  * 3 tier => 4 tier !

Tests unitaires

* Test =
* ensemble de cas à tester
* Accompagné d'une procédure d'exécution
* Lié à un objectif
* = livrer un produit le plus fiable possible

Tests unitaires

* On peut faire des tests de plusieurs natures :

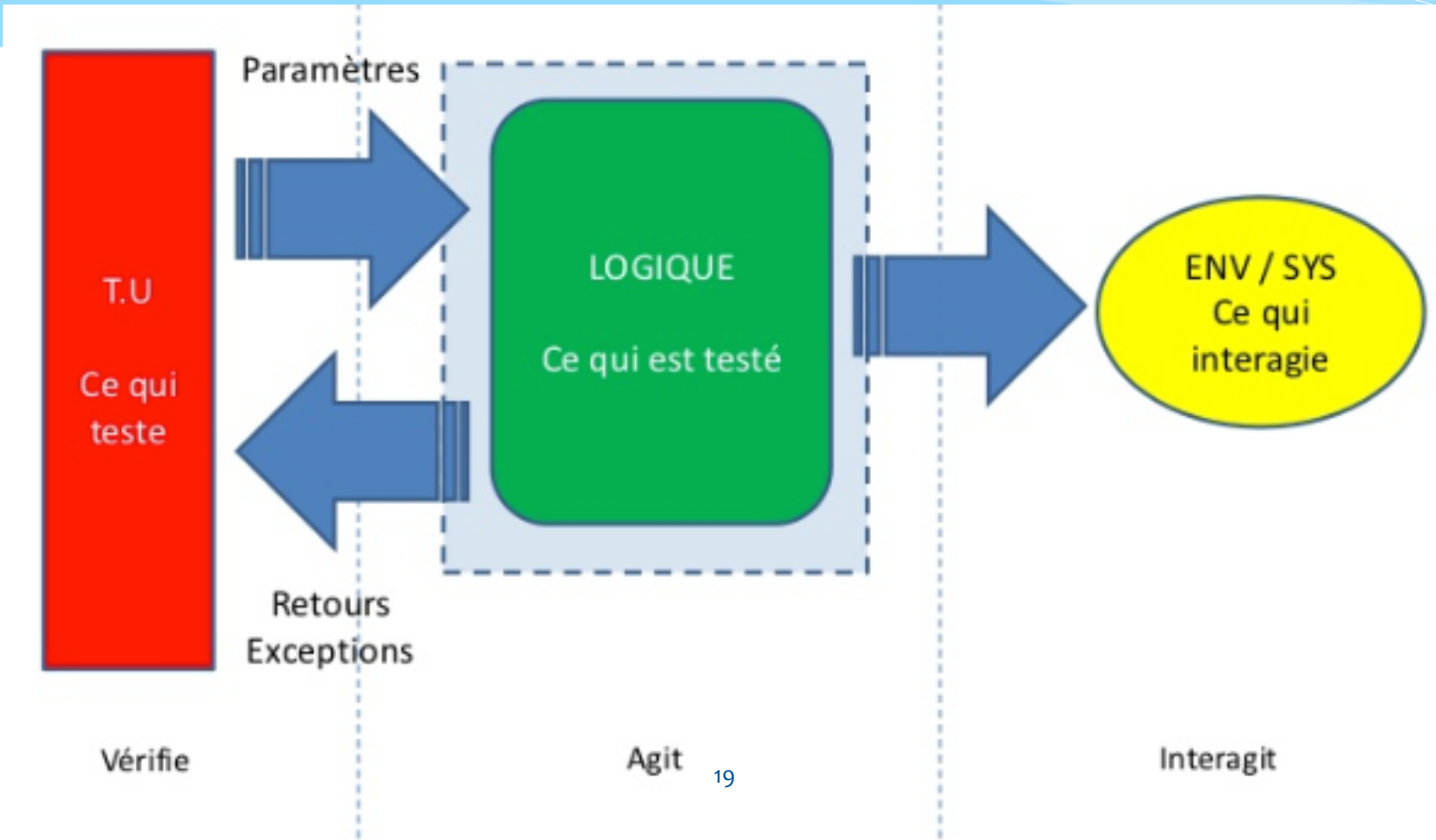


Tests unitaires

* Test unitaire =
* Procédé
* Pour s'assurer d'un fonctionnement correct d'une partie déterminée d'un logiciel
* =module, unité
* Voir à adapter le code à la pratique du test
* Donc voir à écrire des parties courtes ?

Tests unitaires

* Qu'est ce qu'un test unitaire :



Tests unitaires

* N'est pas un test de bout en bout
* Ne doit exécuter que l'unité à tester
* Doit être déterministe (vrai ou faux)
* Ne doit pas dépendre de l'environnement
* Ne peut pas être exhaustif
* Peut permettre de trouver des cas non pensés
* Peut permettre de refactorer le code au bon moment (quand ça marche)

Tests unitaires

* Pourquoi écrire des tests unitaires :

  * Pour ne plus avoir peur de modifier du code
  * Pour garantir la non régression
  * Pour comprendre du code que l'on a pas écrit
  * Pour pouvoir mettre du code à la poubelle sans avoir peur de perdre quelque chose
  * Le test unitaire est a rapprocher du versioning
  * Il faut tester du code avant d'en faire une version suivante

Tests unitaires

* Un framework de programmation permet de pratiquer les tests unitaires
* Sinon il faut ajouter un outil à votre environnement : xUnit
* Cunit : C
* CppUnit : C++
* Junit : Java
* OCUnit : ObjC
* Nunit : C# (.net)

Tests unitaires

* Les tests s'intègrent plus généralement dans
* XP : eXtreme Programming
* Méthode Agile : SCRUM
* Test Driven Development :
  * écrire le test avant le code
  * modifier le code que si le test trouve une erreur
  * améliorer son code en continue
* Intégration continue : jenkins

Tests unitaires

* Intégration continue :

Tests unitaires

* Un test unitaire consiste à écrire un peu de code pour tester du code
* C'est ce que vous faites déjà quand vous voulez tester votre application pour mettre en valeur certaines erreurs.
* Cela demande des efforts
* Des efforts à faire dans le calme
* Plutôt que de devoir recommencer dans l'urgence d'un projet qui ne fonctionne pas, dans l'attente d'un livrable, face à un client en colère

exemple

```
BEGIN_TEST(testSomme)
{
int a =2;
int b =2;
WIN_ASSERT_EQUAL(4, somme(a,b)); //vérifie que la fonction somme retourne le bon résultat
}
END_TEST
```

Si la console affiche 0 failed c'est que la fonction somme fait bien son travail.

Test Unitaires

* Quel est l'apport des tests unitaires ?

* la qualité de l'application : une production qui va mieux (moins de bugs, et moins de régressions, donc moins de support et de *bug fixing* !)
* la qualité du code (une diminution de la dette technique et du coût de maintenance)
* un meilleur design de code induit par sa testabilité, une granularité plus juste (taille des blocs), une meilleure gestion des dépendances entre composants
* avoir des tests unitaires en filet de sécurité, c'est la possibilité de faire sereinement du refactoring de code qui nous permet plus d'agilité
* une documentation (un exemple d'utilisation), qui améliore la productivité des développeurs

Tests unitaires

* Quel est le coût des tests unitaires ?
* coût supplémentaire d'écriture des tests
* coût de maintenance des tests existants
* analyser les tests en échec (problèmes des "qui s'en occupe")
* correction des tests suite aux évolutions
* maintenance de l'environnement nécessaire aux tests (forge logicielle, base, espace disque, ressources extérieures…)
* un *time-to-market* plus long : en effet, peut-on se permettre de livrer en urgence un patch de production lorsque ses tests sont rouges ? Et doit-on attendre (parfois plusieurs heures) la fin de l'exécution de tous les tests avant de pouvoir le livrer ?

Tests unitaires

* Xcode intègre OCUnit de base
* La commande : cmd + U
* Case à cocher au lancement du projet
* Certains personnes développent avec iOS sans jamais faire de tests
* Attention : dissocier le test unitaire de la recherche de bug avec des outils comme « instruments ».

Tests unitaires

* C'est un levier pour atteindre une architecture limpide
* Mais il faut en être persuadé
* On peut dégager de la documentation à partir d'un test unitaire (compréhension du code)
* Les tests unitaires partent du principe de travailler en équipe
* Une équipe évolue, change, cela peut être sécurisant pour quelqu'un qui arrive de pratiquer les tests, pour être sur de ne pas casser ce qui a été fait.