R1.01 : Premières notions de qualité

Káthia Marçal de Oliveira – kathia.oliveira@uphf.fr

Qu’est que c’est la qualité pour vous?

Qu’est que c’est la qualité du logiciel pour vous ?

Définition de la qualité

- La qualité est développer, créer et fabriquer des produits plus économiques, plus utiles et satisfaisants pour l'acheteur. (ISHIKAWA, 1986)
- Ensemble des traits et des caractéristiques d'un produit ou d'un service qui leur confèrent l'aptitude à satisfaire des besoins exprimés ou implicites (ISO 8402, 1994).

Produit logiciel

- Différents types :
  - cahier des charges
  - documentation utilisateur
  - spécifications,
  - programme exécutable
  - code source
  - etc.

Assurer le fonctionement

disponible

pas tomber en panne

pas mettre en risqué les personne

être fiable

....

Sureté du Fonctionement : Definition

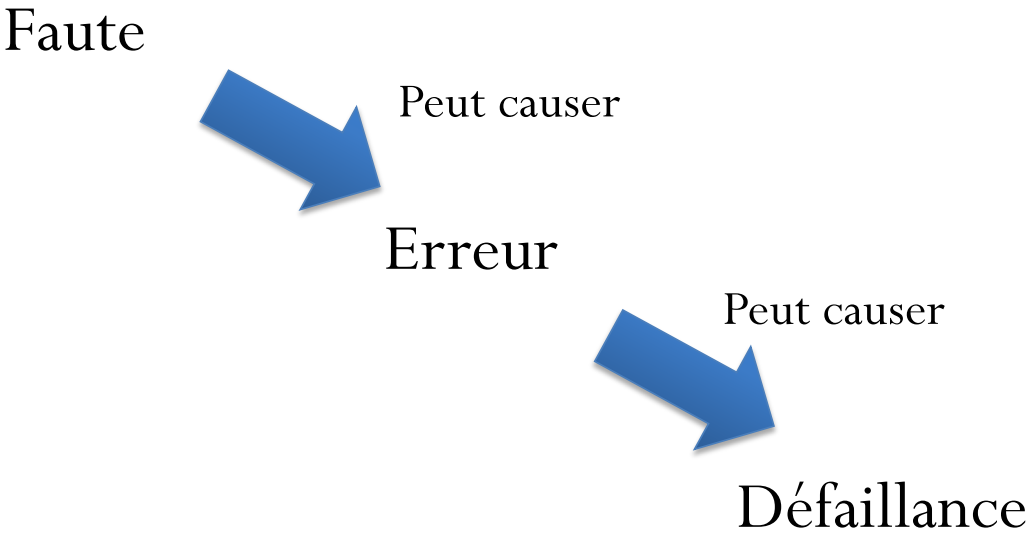
- La sûreté de fonctionnement d'un système est **la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre** (Laprie et al. 1995).

- Vise principalement à réduire le plus possible le nombre de **défaillances d'un système**

Laprie J.-C., J. Arlat, J.-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.-C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac, P. Thévenod, Guide de la sûreté de fonctionnement, Editions Cépaduès, Toulouse, mai 1995, 369 p.

Sureté du Fonctionement : Entraves

Événements qui peuvent affecter la sûreté de fonctionnement du système



Entraves Définition

Faute → Caractéristique statistique du logiciel qui quand elle est activée peut provoquer une erreur.

Erreur → Item d'information ou état incohérent.

Défaillance → Etat du produit dans lequel le service délivré n'est pas conforme au cahier des charges

La cessation de l'aptitude d'une entité à accomplir une fonction requise.

Fautes

- Exemples de fautes de code:
  - On a oublié d'initialiser une référence, une variable
  - On a fait un test dans un « if » sur la mauvaise condition ('>' plutôt que '<')
  - On appelle une méthode avec la mauvaise instance d'une classe.
  - On a utilisé l'assignation (=) plutôt que la comparaison (==)
  - etc…
- Note: si on n'exécute jamais le code où se trouve la faute algorithmique, elle ne se manifestera jamais. Il s'agit d'une faute **latente**.

Intervalle de temps entre le moment de la création de la faute et la première activation conduisant à une erreur.

Exemple

```
int a, b, x, y, k, ...
...
x=1;
if (a == b) {
    ...
    x = a - b; }
if (k == 1) {y = y / x};
```

Faute ➔ Il fallait écrire x = a + b

Lors d'une exécution :

Si a≠b
pas **d'erreur** (malgré la présence de la faute)

Si a=b
si k ≠ 1 ➔ pas de défaillance
(malgré la présence de l'erreur)
si k = 1 ➔ défaillance x=0

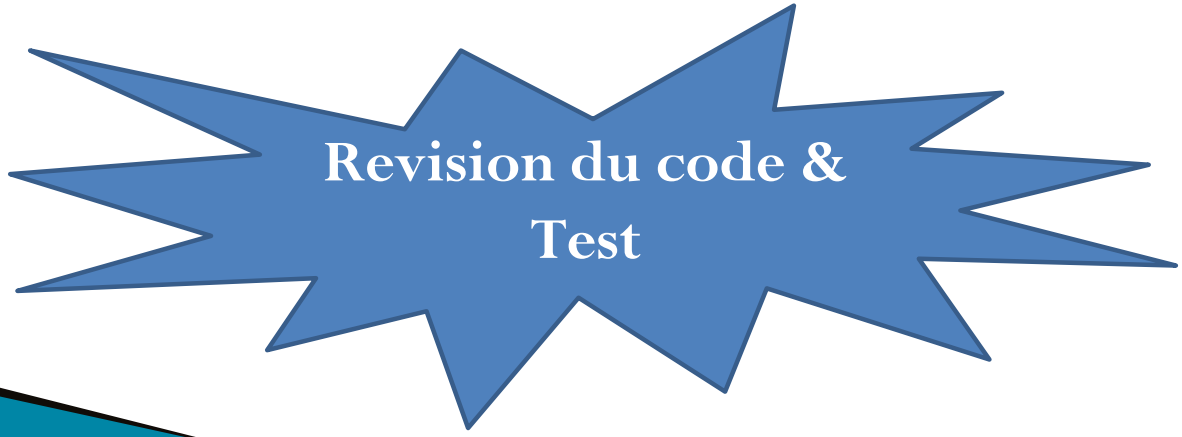
Classification des défaillances

- Suivant la nature de la perturbation :
  - **de valeur** - délivrance d'une valeur erronée.
  - **temporelle** - la réaction n'arrive pas au moment attendu (elle arrive soit trop tôt soit trop tard).
  - **par arrêt** - plus de données, plus de réaction. arrêt sur défaillance.

- Suivant la durée de la défaillance :
  - **Durable** - altère le service rendu pendant un temps représentant une fraction importante de la durée de la mission.
  - **Temporaire** - altère le service rendu momentanément puis fournit a nouveau un service correct.

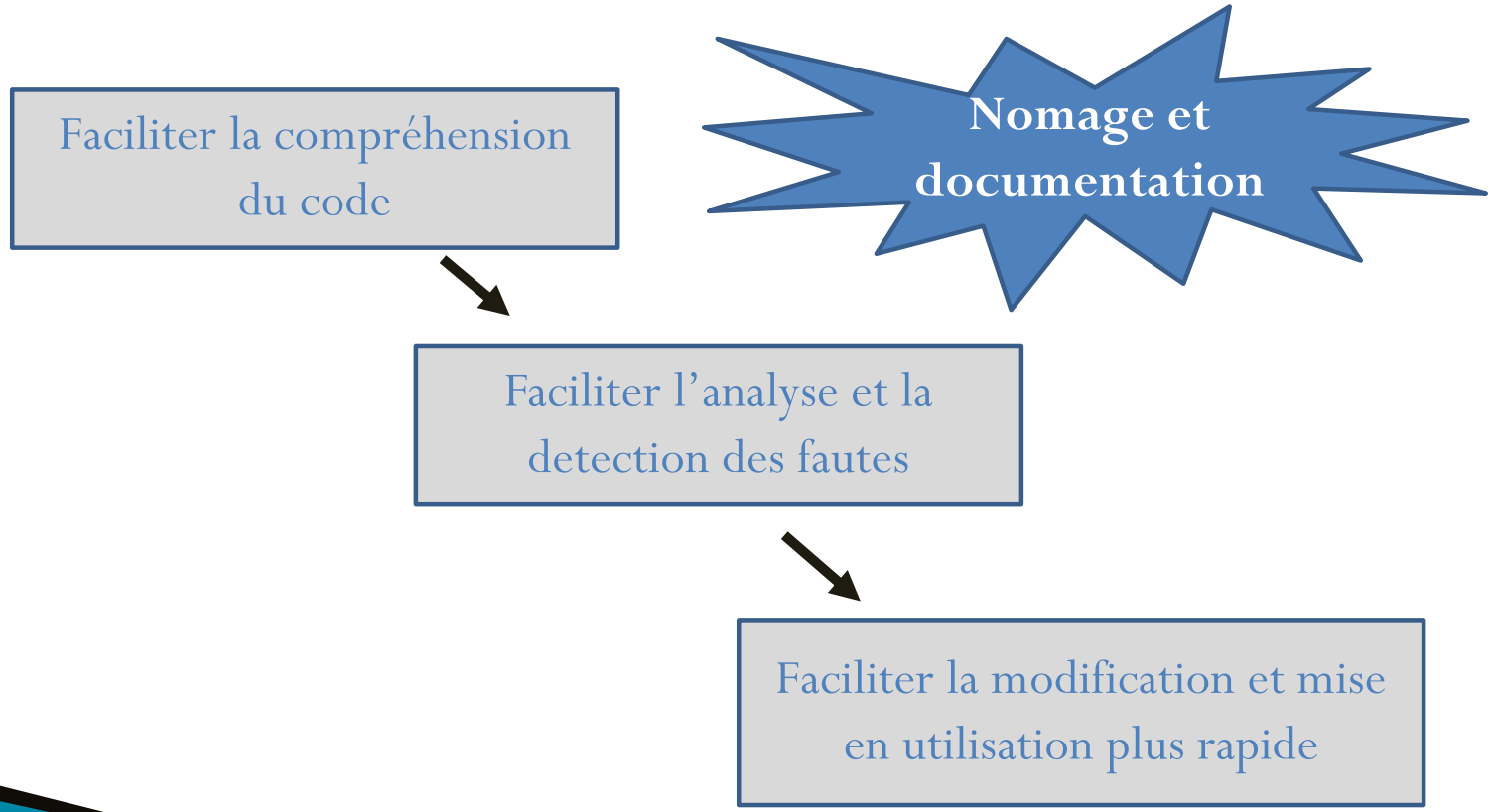
Quoi faire ?

- Evaluer prévisionnellement le comportement du système par rapport à l'occurrence des fautes
  - Examiner les défaillances des composants d'un système et leurs conséquences sur la sûreté de fonctionnement
- Etats observables du système
  - Service correct : accomplit la (les) fonctions du système
  - Service incorrect : non (accomplit la (les) fonctions du système)



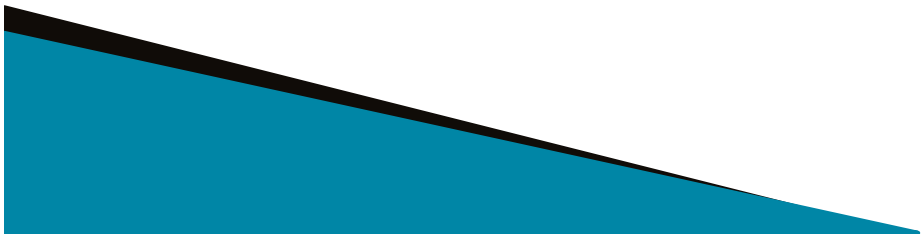
Quoi faire ?

- Aider pour la mantunebailité



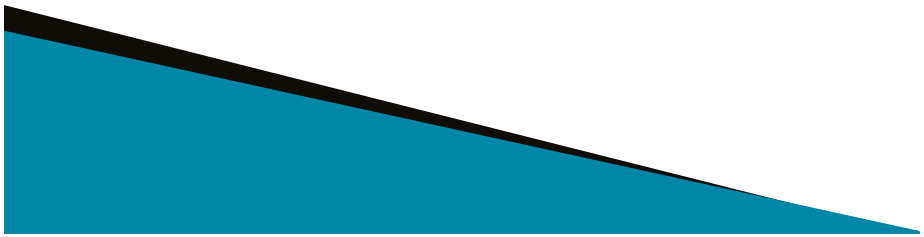
Convention de Nomage

- Ensemble de règles destinées à choisir les **identifiants** (noms des éléments) dans le code source et la documentation
- Avantages :
  - homogénéisent de code
  - rendre le code source plus facile à lire et à comprendre avec moins d'efforts
  - permettre aux développeurs de trouver plus facilement les noms utilisés dans un projet (aussi bien pour les nommer que pour les appeler) ainsi que de collaborer plus facilement quand un code est écrit à plusieurs



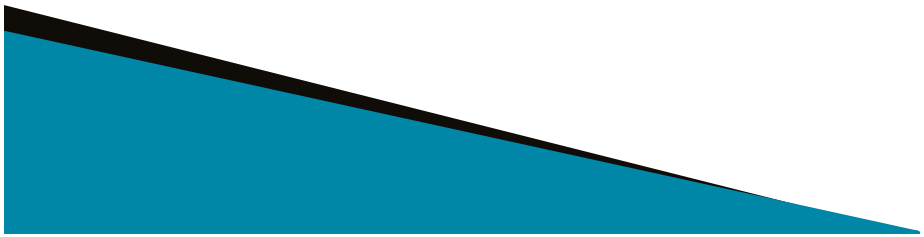
Convention de Nomage

- Grands principes :
  - choisir des **noms signifiants**, qui indiquent ce qu'est la chose nommée.
  - Eviter les noms trop courts et trop longs.
  - Eviter les abréviations (acronymes)
  - Définir un moyen permettant de repérer immédiatement si un identifieur est un type, une constante, un paramètre, une fonction, un pointeur, etc



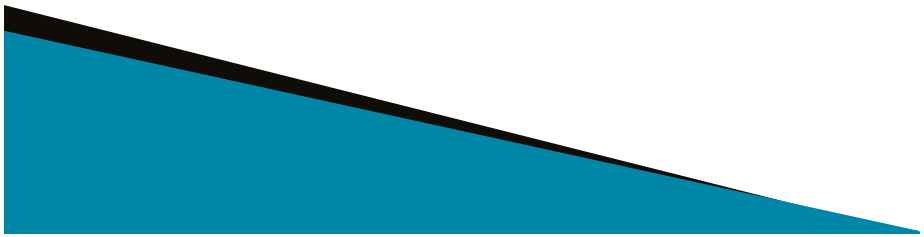
Les differents conventions de Nomage

- Le Camel case
- Le Pascal case
- Le Kebab case ou Spinal case
- L'Underscore case ou Snake case



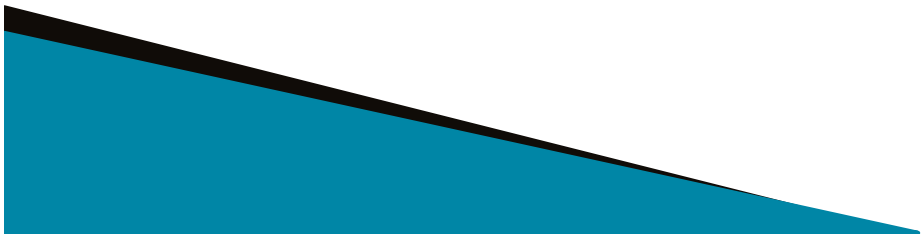
Convention de Nomage : Le Camel case

- Les mots sont liés sans espace. Chaque mot commence par une majuscule à l'exception du premier.
  - **Ex.** : maVariable, nomParent
- Quand on utilise des acronymes (URL, CSS, HTML, etc...), seule la première lettre de ces derniers sont à mettre en majuscule. ex: monUrl
- la convention la plus connue et la plus utilisée.
- Utilisée en JavaScript, Java, C++, C# .



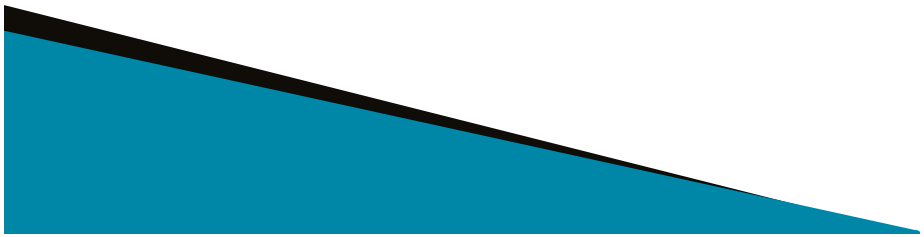
Convention de Nomage : Le Pascal case

- les mots sont liés sans espace. Chaque mot commence par une Majuscule.
  - **Ex.** : MaVariable, NomParent
- Acronymes : MonUrl
- Utilisée en Pascal, encore pour nommer les class en **PHP**.



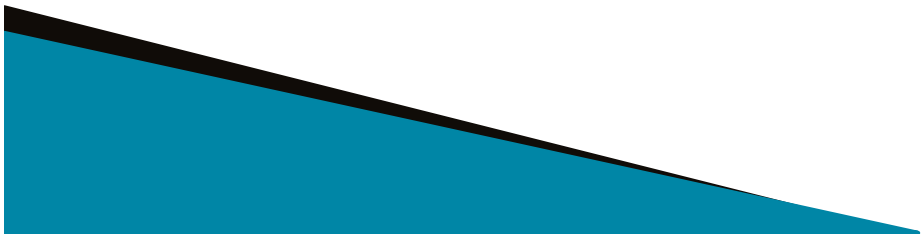
Convention de Nomage : Le Kebab case ou Spinal case

- les mots sont en minuscule et sont liés par des tirets (-).
  - **Exemples** : ma-variable, nom-parent
- Utilisée pour écrire les URL ainsi que pour nommer les images, PDF et autres fichiers à destination du web.
- Aussi pour nommer les **class** et **ID** dans le code HTML e CSS.



Convention de Nomage : L'Underscore case ou Snake case

- les mots sont en minuscules et sont liés par des underscores (tiret bas : _).
  - **Exemples** : ma_variable, nom_parent
- Utilisée en **PHP**, en **Ruby** ou encore en **Phyton**.
- En PHP pour le nomdes constantes tous les mots sont majuscule, toujours séparés par des underscores (Exemple : MA_CONSTANTE)



Régles de codage et convention de nommage

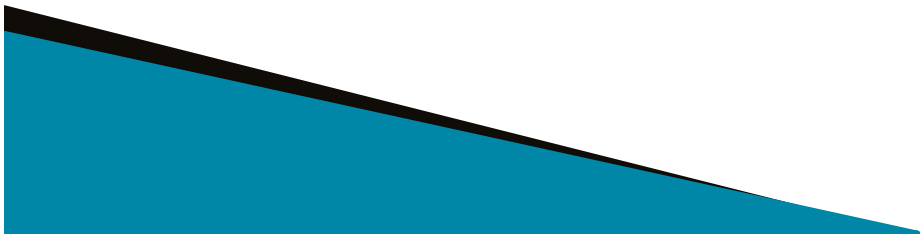
- **Les variables et attributs**
  Un nom de variable (ou d'attribut) est un **nom principal** (surtout pas un verbe) suffisamment éloquent, éventuellement completé
- **Les fonctions et méthodes**
  - Un nom de fonction est construit à l'aide d'un **verbe** (surtout pas un nom), et éventuellement d'éléments supplémentaires comme : une quantité, un complément d'objet ou un adjectif représentatif d'un état)
  - Le verbe peut être au présent de l'indicatif ou à l'infinitif.
- **Les classes**
  Un nom de classe est un **nom principal** (surtout pas un verbe) suffisamment éloquent (il représente un concept),

Documentation du Code

- Grands principes :
  - Commenter le code **pendant** l'écriture du code (après, c'est trop tard).
  - **Maintenez la documentation en même temps que le code**
  - Commenter tout ce qui difficile à comprendre, pas ce qui est clair sans commentaire
  - Un code non commenté ou trop commenté ce n'est pas bon
  - « Versionnez » la documentation avec le code si vous utilisez un gestionnaire de code
  - il est recommandé de décrire les commentaires de manière **succincte**, sans exagération.
  - Commentez ce qui est relevant

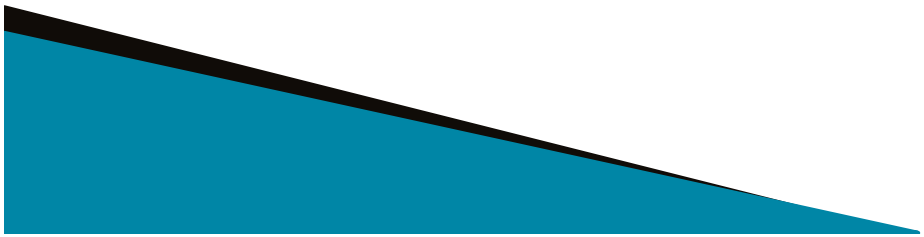
Où commenter ?

- Ajouter un commentaire avant chaque fonction précisant ce qu'elle fait et pourquoi elle doit être appelé
- Ecrire un commentaire pour les variables importantes
- Commenter les loops (boucles) importants. Mettre l'objectif de la boucle, et quel est le résultat attendu
- Commenter l'enchainement de ifs (avant)



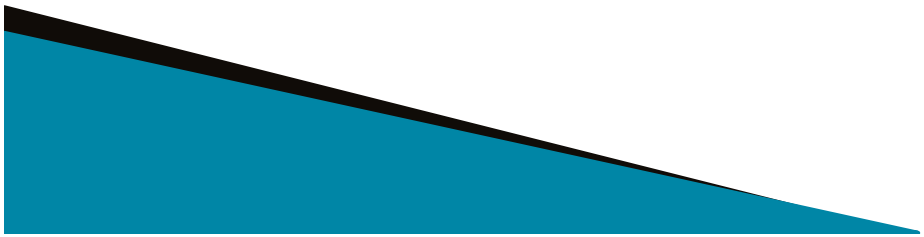
Autres….

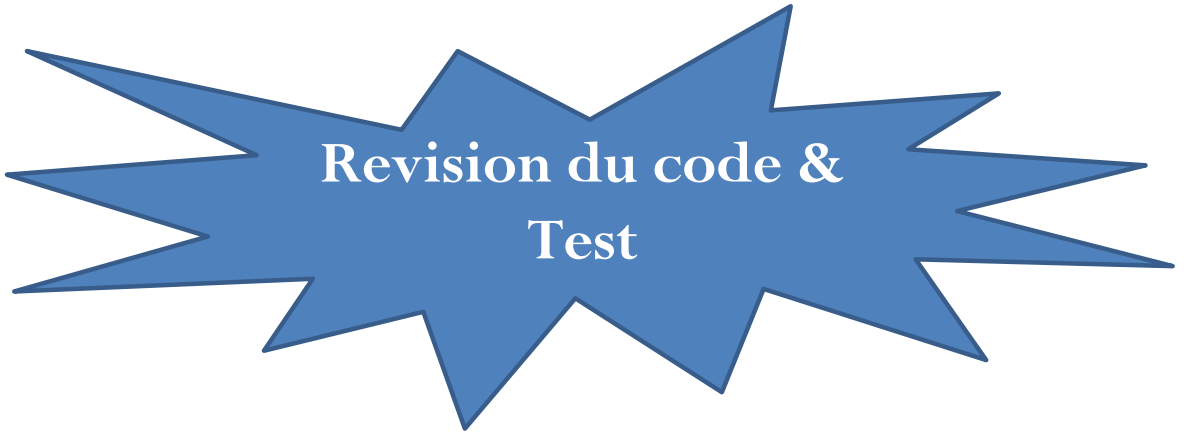
- Documenter les alertes ou les avertissements indiquant qu'il y a un problème à venir dans le code et qu'une solution de contournement a été trouvée
- Si vous pensez que c'est nécessaire, commentez la partie du code qui n'a pas fonctionné et laissez-la en héritage, pour enregistrer que cette approche n'a pas fonctionné.
- Ajoutez des commentaires qui informent les autres développeurs lorsque vous faites quelque chose qui semble étrange.



Style de codage pour aider comprehension

- Limiter la taille des functions - Ecrire des fonctions courtes et simples
- Limiter le nombre de caractères par ligne (choix courrant 80 ou 100 max)
- Indenter le code, notamment pour faire ressortir les blocs de code (entre { }), donc en particulier : le corps des fonctions, les structures de contrôles (conditionnelles, boucles , …)




Revision du code &
Test

Révision du Code

- **Examen systématique du code source par une tierce personne**.

- Son objectif est de :
  - déceler les bugs avant la mise en production,
  - d'améliorer **la qualité**
  - d'améliorer la sécurité du code.

Pendant la revue on vérifie le nommage et la documentation

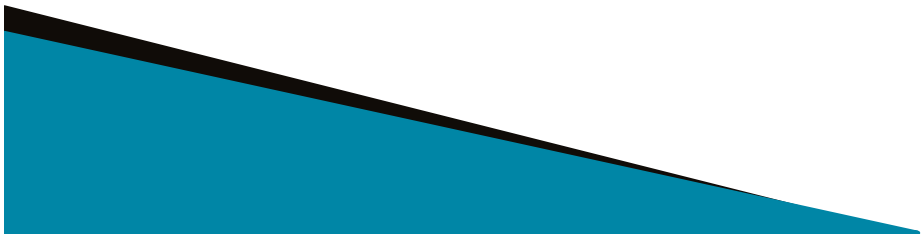
Avantages de revue du code

- Oblige le développeur à relire ce qu'il vient de coder, et donc à **prendre du recul**
- Permet de **limiter fortement le risque de bugs** mis en production
- Sensibilise toute l'équipe au **respect des bonnes pratiques** et à la qualité du code
- Favorise un **code plus lisible et maintenable**
- Encourage la tenue à jour de la **documentation**
- Permet la **montée en compétences** de l'ensemble de l'équipe
- **Améliore l'implication** et la compréhension de chaque développeur dans le projet.

Test

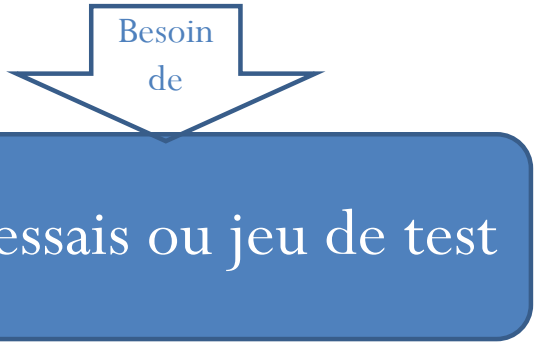
- « Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus »-*IEEE Standard Glossary of Software Engineering Terminology*

- « C'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts »-G. Myers (*The Art of Software testing*)



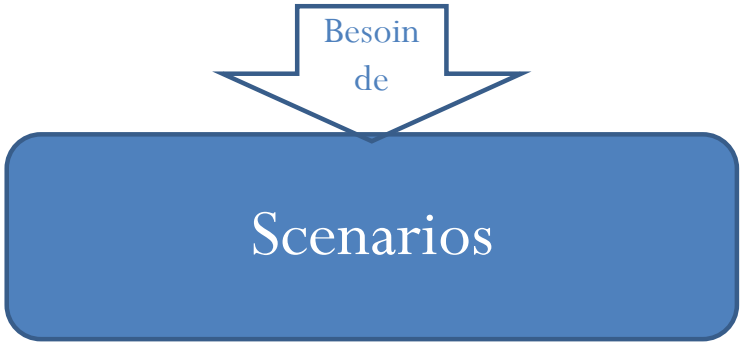
Objectifs du test

- S’assurer de la correspondance entre un programme et sa spécification
- Révéler les défauts du logiciel
- Garantir que l’exécution du programme ne donne pas un résultat inattendu
- Avoir confiance dans le fonctionnement adéquat du système
- Connaître le point limite du système avant son blocage
- Appréhender les risques liés à la livraison d’un système aux utilisateurs



Jeu d’essais ou jeu de test

- Est une description d’une suite d’action et de résultats attendus, ayant pour objectif la **validation d’une fonctionnalité de l’application**.



Qu’est-ce qu’un scénario ?

- Un scénario de test regroupe l’ensemble des étapes à réaliser pour tester une fonction.
- **Un scénario = un cas de test** avec des conditions initiales, un déroulement et des résultats attendus.
- Il ne valide qu’une seule fonctionnalité/exigence.
- Au moins un test est réalisé pour chacune des fonctions. Plusieurs tests peuvent être menés, on parle alors de campagne ou de plan de tests.

Exemple 1 : Connexion sur l'ENT

Scénarios :

- Un utilisateur saisi login et mot de passe correcte
- Un utilisateur saisi login correcte et mot de passe incorrecte
- Un utilisateur saisi login incorrecte et mot de passe correcte
- Un utilisateur saisi son login et mot de passe incorrectes
- Un utilisateur laisse le login vide
- Un utilisateur laisse le mot de passe vide

Exemple 2 : la mise en ligne d'un "Code de réduction" sur le des achat.

Scénarios :

1. Un utilisateur non connecté arrive sur la page panier avec un article dans celui-ci et il rentre le code de réduction.
2. Un utilisateur identifié connecté de catégorie 1* arrive sur la page panier avec un article et rentre le code de réduction. * La catégorie 1 n'offre aucun avantage.
3. Un utilisateur identifié connecté de catégorie 2* arrive sur la page panier avec un article et rentre le code de réduction. * La catégorie 2 accorde automatiquement 10% de réduction.

Le cas de test

- A partir de chaque scénario, nous définissons des cas de test
- il suffit de placer une donnée en entrée d'une application et de vérifier sa valeur en sortie
- Le test d'une application peut se faire avec :
  - des tests qui fonctionnent et donnent un résultat positif.
  - des tests qui ne fonctionnent pas et donnent un résultat négatif.
- La détermination de l'état de **réussite / échec** dépend de la façon dont le résultat attendu et le résultat réel se comparent.
  - Résultat de test = résultat attendu ➔ réussite
  - Résultat de test != résultat attendu ➔ échec