



Journées Perl 2008





Vous avez dit « Kalité » ?

- x **Tentative de définition**

- x La « Kalité » est une *approximation* de « Qualité »
- x Nul ne sait ce que c'est vraiment...
- x Mais on *pense* la reconnaître quand on la voit !

- x **C'est avant tout une question de *confiance***

- x Construite grâce à la réussite de tests (mais ce n'est pas tout)
- x Mais absence de bugs (trouvés) n'implique pas Kalité !
- x Éventuellement si la couverture *fonctionnelle* des tests est *décente*

- x **La chasse aux *bugs* reste cependant ouverte !**

- x Un bug est une différence entre l'attendu et l'implémenté
- x C'est aussi une différence entre test, documentation & code
- x Si la documentation est ambiguë, c'est bel et bien un bug !



Avertissement !





Quand & Quoi ¹

- x **Complètement avant**
 - x Littérature
 - x CPAN
 - x Articles, conférences, /Perl Mon(k|geur)s/, etc.
 - x « *Lis. Apprends. Évolue.* » – Klortho le Magnifique
- x **Avant**
 - x *Générer* le squelette du module
 - x Utiliser un système de classes OO comme **Moose** (si applicable)
 - x Écrire les tests (un soupçon d'**XP** dans votre code)
- x **Pendant (*la phase d'écriture de code*)**
 - x Documenter au fur et à mesure (et pourquoi pas, avant ?)
 - x Ajouter des tests si nécessaire



Quand & Quoi ²

- x **Après (*entre codage & livraison*)**
 - x Tester (suite de tests – acceptation *et* non-régression)
 - x Mesurer la couverture de POD
 - x Mesurer la couverture de code des tests
 - x Mesurer la couverture fonctionnelle des tests (Ah ! Ah !)
 - x Générer des synthèses lisibles
 - x Pour avoir un statut en un coup d'œil et une meilleure traçabilité
- x **Bien après (*la livraison*)**
 - x Reformuler tôt, reformuler souvent
 - x La suite de tests (non-régression) *devrait* assurer que rien n'a été cassé
 - x Suite à un rapport de bug...
 - x Ajouter tout d'abord des tests afin de reproduire le problème
 - x Puis seulement éradiquer le(s) bug(s)
 - x Tester de nouveau (suite de tests – non-régression)



L'Enfer, c'est les autres...

- x **... codeurs** 😊

- x « Codez comme si le gars devant reprendre votre code était un psychopathe violent qui sait où vous habitez. » – D. Conway



- x **Préface du SICP**

- x « Les programmes doivent être écrits afin d'être lus par des humains et incidemment exécutés par des machines. »



Les pré-requis ¹

- x **SCM – gestion de code source (~ historique + //)**
 - x Par exemple : cvs, subversion, svk, darcs, git, etc.
 - x Attention ! Requierent une certaine *étiquette* (labels, branches, etc.)
 - x Utiliser une « *machine à différence* » (diff), avec GUI (tkdiff)
- x **RT – gestion de tickets (~ intention)**
 - x Par exemple : cvstrac, trac, RT, bugzilla, etc.
- x **Éditeur avec coloration syntaxique**
 - x Par exemple : NEdit, vi, emacs, etc.
- x **Des règles de codage consistantes**
 - x Voire même les « bonnes » ;)
 - x Cf PBP (livre) + [Perl::Critic](#) (module) + perltidy (outil)



Les pré-requis ²

- x **On peut même utiliser un IDE**
 - x Comme Eclipse + plugin Perl
- x **On ne choisit pas forcément...**
 - x SCM, RT, « *bonnes* » pratiques ou même l'éditeur de texte :(
 - x Fonction de l'OS, des choix « *corporate* », du client, etc.
 - x Faute d'avoir ce que l'on aime, il faut aimer ce que l'on a !
- x **Mais on *choisit* d'utiliser les directives « *use* »**
 - x `use strict; # code`
 - x `use warnings; # test`
 - x C'est même très fortement conseillé !
- x **Sinon : « *Il y en a qui ont essayé...* »**



Les pré-requis ³

x « **Ils ont eu des problèmes !** »





Ne pas réinventer la roue ¹

- x **Éviter de commettre les erreurs des autres**
 - x Difficile de résister au syndrome NIH (« *Not Invented Here* »)
 - x Moins d'orgueil, plus de paresse !
- x **Chercher plutôt à utiliser un module du CPAN**
 - x « *Je code en CPAN, le reste c'est de la syntaxe* » – Audrey Tang
- x **Mais faire au préalable une revue de module**
 - x Utilité dans la pratique
 - x Possibilités de configuration
 - x Développement actif du module (mais peut-être déjà stable)
- x **Mais si vous voulez toujours réinventer la roue...**
 - x Au moins essayez d'en réinventer une meilleure !



Ne pas réinventer la roue ²

- x **Quelques tâches parmi tant d'autres...**
 - x Des fois même pénibles à coder !
- x **Analyser la ligne de commande**
 - x `Getopt::Long` (un grand classique)
 - x `Pod::Usage`
- x **Gérer des configurations**
 - x `Config::Std` (~ M\$ INI)
 - x `YAML`
 - x Et non, pas de XML (« *même pas dans tes rêves les plus fous* ») !
- x **Pèle-mêle... (cf `Phalanx Top 100`)**
 - x `HTML::Parser`, `XML::Twig`, `Spreadsheet::ParseExcel`,
`Parse::RecDescent`, `RegExp::Common`, `List::MoreUtils`, etc.



Ne pas réinventer la roue ³

x **Littérature**

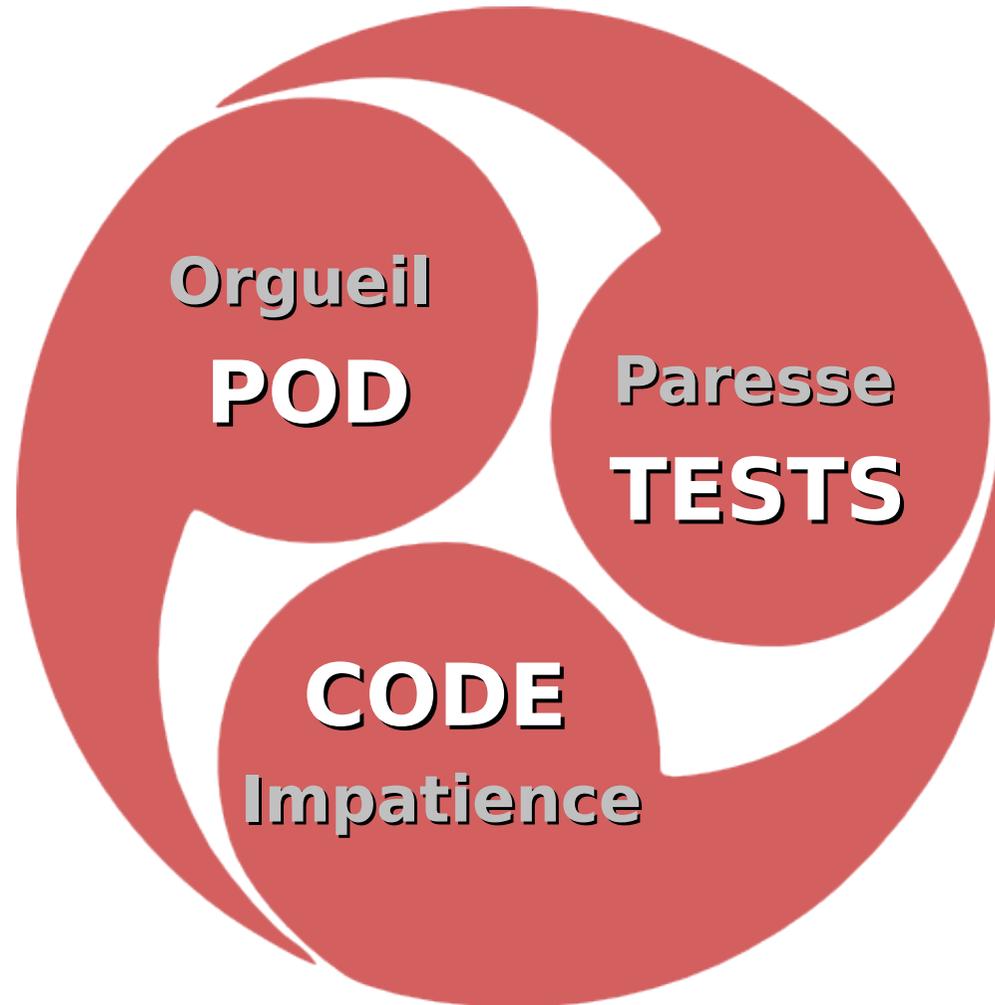
- x Perl en action (Perl cookbook – Christiansen & Torkington)
- x De l'art de programmer en Perl (Perl best practices – Conway)
- x Mastering algorithms with Perl (Orwant, Hietaniemi & MacDonald)
- x Perl testing: a developer's handbook (Ian Langworth & Chromatic)
- x The pragmatic programmer (Hunt & Thomas)
- x Lessons learned in software testing (Kaner, Bach & Pettichord)
- x The art of agile development (Shore & Warden)

x **Expériences**

- x Groupes (Mongueurs de Perl, Perl Mongers, Perl Monks)
- x Conférences (Journées Perl, Perl Workshops, YAPC)
- x Articles ([Mongueurs de Perl](#) / [GLMF](#), [perl.com](#))



Le triptyque du programmeur





Au commencement...

- x **Bien construire son propre module**
 - x Un module Perl est une arborescence très précise
 - x Facile d'oublier l'un de ses nombreux fichiers
 - x Difficile de se souvenir de la syntaxe de chacun d'entre-eux
- x **Utiliser un module dédié du CPAN**
 - x Par exemple : [Module::Starter](#) (voire même [Module::Starter::PBP](#))
 - x Crée des « *gabarits* » à compléter
 - x Certains tests vérifieront qu'ils ont bien été modifiés
- x **Cf l'article de Sébastien Aperghis-Tramoni**
 - x « *Créer une distribution pour le CPAN* », [GLMF #69](#)
 - x <http://articles.mongueurs.net/magazines/linuxmag69.html>



Le test pour les nuls ¹

- x **Tester = confronter *intention* & *implémentation***
 - x À l'aide de techniques (tests directifs ou aléatoires contraints)
 - x Et d'un modèle de référence (OK ~ pas de \neq avec ce dernier)
- x **Intention**
 - x Cristallisée dans une spécification, un plan de test, etc.
 - x Quand lesdits documents sont bel et bien disponibles !
 - x Documents *non-formels* car destinés à une lecture humaine
 - x Donc bien évidemment sujets à *interprétation*
- x **Implémentation**
 - x Code (+ documentation)
 - x Décomposé en unités de base (modules = \sum fonctions)



Le test pour les nuls ²

- x **Développement piloté par les tests (TDD)**
 - x Tests unitaires (au standard [xUnit](#) ou non)
 - x Tests d'acceptation (ou de recette – ce pour quoi le *client* a payé)
- x **Suite de tests ≈ spécification exécutable**
 - x C'est déjà un peu plus formel (ou moins informel ;)
 - x « *Les vieux tests ne meurent jamais, ils deviennent des tests de non-régression !* » – [Chromatic](#) & [Michael G Schwern](#)
- x **Mais un test réussi ne révèle pas grand chose !**
 - x Ça doit même devenir frustrant pour le testeur !
- x **Juste histoire d'enfoncer le clou...**
 - x « *Tester un programme démontre la présence de bugs, pas leur absence.* » – [Edsger Dijkstra](#)



Le test pour les nuls ³

- x **Un ingénieur de test doit se poser 2 questions**
 - x « *Est-ce correct ?* »
 - x « *Ai-je fini ?* »
- x **Est-ce correct ?**
 - x *Tous* les tests de la suite sont-ils OK ?
 - x C'est le rôle du protocole TAP et de `Test::More` + `Test::Harness`
 - x Avec les notions de SKIP/TODO, c'est plutôt binaire (i.e., 100%)
- x **Ai-je fini ?**
 - x Mes tests ont-ils exercé toutes mes lignes de code ?
 - x Notion de couverture de code (associée à une métrique)
 - x C'est le domaine du module `Devel::Cover`
 - x Mais... mes lignes de code implémentent-elles la fonctionnalité ?



Le test pour les nuls ⁴

- x **Couverture de code \neq couverture fonctionnelle**
 - x C'est en effet très tentant de confondre les deux
- x **Couverture de code**
 - x On peut couvrir le code à 100% mais...
 - x Il peut manquer en fait celui réalisant la fonctionnalité demandée !
- x **Couverture fonctionnelle**
 - x Meilleure définition du « *ai-je fini ?* » en ces termes
 - x Liée aux combinaisons possibles en entrée d'une fonction (cf CRT)
- x **M'enfin ! Comment mesurer la CF en Perl ?**
 - x C'est possible avec des HDVL comme [SystemVerilog](#)
 - x C'est dans les TODO du module [Test::LectroTest](#)



Le test pour les nuls ⁵

x Couverture de code \neq couverture fonctionnelle

x Le trivial contre-exemple suivant

x `=head2 foo`

Retourne 'foo' à 'bar' et 'gabuzomeu' à 'baz'. Retourne undef si entrée inconnue.

`=cut`

```
sub foo {  
  my $s = shift;  
  
  return 'gabuzomeu' if $s eq 'baz';  
  
  undef;  
}
```

`use Test::More tests => 2;`

```
is ( foo( 'baz' ), 'gabuzomeu', "retourne 'gabuzomeu' à baz" );  
is ( foo( 'foo' ), undef, "retourne undef si entrée inconnue" );
```

x Atteint 100% de CC... mais n'implémente pas `foo ('bar') = 'foo' !`

x

File	stmt	bran	cond	sub	pod	time	total
t_foo.t	100.0	100.0	n/a	100.0	n/a	100.0	100.0
Total	100.0	100.0	n/a	100.0	n/a	100.0	100.0



Tests unitaires

```
use warnings;  
  
use Test::More;  
  
plan(tests=>N);  
  
use_ok('Foo');  
  
# ...
```

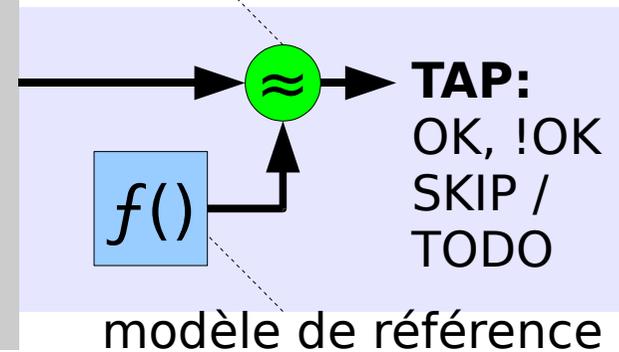
```
is_deeply(  
    bar ($baz),  
    refbar ($baz),  
    'baz au bar'  
); stimulus  
  
# ...
```

t/13-bar.t

```
use strict;  
  
package Foo;  
use Carp::Assert::More;  
  
# ...  
  
=head2 bar  
    bar ( baz )  
  
Rince baz.  
=cut  
  
sub bar {  
    my $baz = shift;  
    assert_nonblank $baz;  
    # ...  
}  
  
# ...
```

lib/Foo.pm

Test::More
is(), is_deeply(), ...





Programme de test type ¹

- x **Le plus simple possible**
 - x « *Quis custodiet ipsos custodes ?* »
 - x Sinon il devient nécessaire de tester / factoriser le code de test
- x **En fait, souvent une boucle**
 - x Itérations sur des cas d'utilisations (« *use cases* »)
 - x Comparaison sortie effective / résultat attendu
 - x La fonction de différence varie en fonction du type de données
- x **Pour ce qui est de l'attendu**
 - x Structure complexe sérialisée (Data::Dumper::Simple)
 - x Format dédié (XML, JSON, etc.) mais attention à la différence !
 - x Résultat d'une fonction de référence (ré-écriture complète)
 - x Structure complexe codée en dur (liste/hash/Tie::IxHash)



Programme de test type ²

* `use warnings; # fortement conseillé`

`use SpecifyParser;`

Module
CPAN

`use Test::More;`

Dumps
+ Plan

```
my @pl_files = glob "*/*.pl";          # sérialisés avec Data::Dumper::Simple
plan ( tests => scalar @pl_files ); # plan de test
```

```
my $parser = SpecifyParser->new;
my $checks;
```

Boucle
+ Stimuli

```
for my $pl_file ( @pl_files ) {
    ( my $v_file = $pl_file ) =~ s/\.pl$/\.v/;

    do $pl_file; # désérialise $checks
```

Compare

```
is_deeply ( [ $parser->parse_file ( $v_file ) ], $checks, $pl_file );
}
```



Le protocole TAP ¹

- x « **Test Anything Protocol** »
 - x Séparation des tests de l'interpréteur de résultats
 - x Développé par des perlistes mais en fait indépendant du langage
- x **La fonctionnalité à tester est une boîte noire**
 - x Le programme de test doit juste fournir un flux TAP
 - x A l'aide d'une boîte à outils (un module du CPAN bien sûr)
 - x Par exemple le module `Test::More` (`plan()`, `use_ok()`, `is()`, etc.)
- x **Le nombre de tests est annoncé à l'avance**
 - x Notion de « *plan* » de test (léger AMHA, vs IEEE Std 829 + lourd)
 - x Le test est déclaré faux si M tests reportés \neq N tests annoncés
 - x Car un test peut par exemple tout planter avant la fin des tests



Le protocole TAP ²

- x **Le plan de test**
 - x 1..N (todo X Y)?
- x **Les tests**
 - x ok X - *description*
 - x not ok X - *description*
 - x ok Y # SKIP *raison*
 - x not ok Y # TODO *raison*
- x **SKIP**
 - x Éluder le test à cause d'un facteur externe (module ∃!, OS, etc.)
- x **TODO**
 - x Fonctionnalité pas encore implémentée (peut néanmoins être OK)



Le protocole TAP ³

- x **Cf articles GLMF #88 & #89**
 - x « *Les tests en Perl - Présentation et modules standards* » & « *Tests et Perl - Bonnes pratiques* »
– Sébastien Aperghis-Tramoni & Philippe Blayo
- x **Quelques interpréteurs TAP**
 - x [Test::Harness](#)
 - x [Test::TAP::Model](#) (MI bâti sur le flux TAP → [Test::TAP::HTMLMatrix](#))
- x **Pour en savoir plus...**
 - x La spécification est en fait le module [TAP](#)
 - x Article sur Wikipedia : [Test_Anything_Protocol](#)
 - x Présentation de Philippe aux Journées Perl (juste après !)
 - x Site web : <http://testanything.org/>
 - x La présentation [Test::Tutorial](#) (chromatic & Michael G Schwern)



Test::Harness

x **make test**

```
x % make test
PERL_DL_NONLAZY=1 /usr/local/bin/perl "-MExtUtils::Command::MM" \
  "-e" "test_harness(0, 'blib/lib', 'blib/arch')" t/*.t
```

```
t/00-load.....ok 2/6#
t/00-load.....ok
```

```
Testing SOCK v1.0.2, \
Perl 5.008007, /usr/local/bin/perl
```

← Traçabilité

Tests
fonctionnels

```
t/01-rip_fmxml.....ok
t/02-rip_fmxml_again.....ok
t/03-rip_register_bit_fields.....ok
t/04-parse_fmxml_datasheet.....ok
t/05-rip_fmxml_table.....ok
t/06-evaluate.....ok
t/07-scavenge_full_description...ok
t/08-spirit_version.....ok
t/09-frontend_tools.....ok
```

Tests
POD

```
t/boilerplate.....ok
t/pod-coverage.....ok
t/pod.....ok
```

Synthèse

```
All tests successful.
Files=13, Tests=141, 40 wallclock secs (20.52 cusr + 1.12 csys = 21.64 CPU)
```



Matrice TAP ¹

- x **Offre une vue synthétique de la suite de tests**
 - x Très appréciable car le nombre de tests ne peut qu'augmenter

- x **À l'aide d'un interpréteur dédié**

- x Le module `Test::TAP::Model::Visual`
- x L'interpréteur analyse le flux TAP et construit un MI TTM
- x Le MI TTM est transformé en HTML (`Test::TAP::HTMLMatrix`)

- x **Très simplement**

```
x use Test::TAP::Model::Visual;
   use Test::TAP::HTMLMatrix;

   $ttm = Test::TAP::Model::Visual->new_with_tests( <t/*.t> );
   $v    = Test::TAP::HTMLMatrix->new( $ttm );

   open FH, "> matrice.html";
   print FH $v->html;
```



Matrice TAP ²

x **Notre make test de tout à l'heure**

Test file		Test cases	%
t/00-load.t	OK		100.00%
t/01-rip_fmxml.t	OK		100.00%
t/02-rip_fmxml_again.t	OK		100.00%
t/03-rip_register_bit_fields.t	OK		100.00%
t/04-parse_fmxml_datasheet.t	OK		100.00%
t/05-rip_fmxml_table.t	OK		100.00%
t/06-evaluate.t	OK		100.00%
t/07-scavenge_full_description.t	OK		100.00%
t/08-spirit_version.t	OK		100.00%
t/09-frontend_tools.t	OK		100.00%
t/boilerplate.t	OK		100.00%
t/pod-coverage.t	OK		100.00%
t/pod.t	OK		100.00%
TOTAL	13 files	141 test cases: 141 ok, 0 failed, 0 todo, 0 skipped and 0 unexpectedly succeeded	100.00%



Matrice TAP³

x Autre exemple : transformation de données

Test file		Test cases																%	
t/ANA33lib/at59000_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/ARM926EJS88lib/at58000_ind_tlf4.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/CSCLib/at58850_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/ETM9mpLib/acu59k_ind_tlf4.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/IO12lib/at59000_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/IO33lib/at58800_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/IO33lib/at59000_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/LLSCLib/acu59k_ind_edb.t	FAILED	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	88.89%
t/SC14lib/at58800_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/SCLib/at58800_ind_edb.t	FAILED	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	88.89%
t/SCLib/at59100_ind_xml.t	FAILED	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	61.11%
t/SCLib/at59101_ind_xml.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/asc948/at58800_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/asca50/at58800_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/asca91/acu59k_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/ascano/at59100_ind_tlf4.t	FAILED	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	88.89%
t/ascaro/at58800_ind_edb.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/ascasj/at59100_ind_xml.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/ascb2l/at58850_ind_edb.t	FAILED	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	88.89%
t/ascb92/at58850_ind_edb.t	FAILED	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	88.89%
t/asciso/at59100_ind_xml.t	FAILED	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	94.44%
t/pingsLib/at58850_ind_tlf4.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
t/pongsLib/at58850_ind_tlf4.t	OK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	100.00%
TOTAL	23 files	414 test cases: 396 ok, 18 failed, 0 todo, 75 skipped and 0 unexpectedly succeeded																95.65%	



Couverture de code ¹

- x **Charger le module `Devel::Cover` lors du test**
 - x `% cover -delete`
 - x `% HARNESS_PERL_SWITCHES=-MDevel::Cover make test`
 - x `% cover -report html`

Coverage Summary

Database: /data/141/users/xavier/dev/SOCK/SOCK/cover_db

file	stmt	bran	cond	sub	pod	time	total
<u>blib/lib/SOCK.pm</u>	100.0	n/a	n/a	100.0	n/a	0.0	100.0
<u>blib/lib/SOCK/BOT.pm</u>	87.0	81.9	68.4	90.6	100.0	31.3	87.0
<u>blib/lib/SOCK/DOM.pm</u>	100.0	70.8	100.0	100.0	100.0	42.4	96.0
<u>blib/lib/SOCK/DOM/Component.pm</u>	94.8	81.8	64.7	100.0	100.0	6.1	91.8
<u>blib/lib/SOCK/MJD.pm</u>	100.0	n/a	n/a	100.0	100.0	0.1	100.0
<u>blib/lib/SOCK/NRT.pm</u>	100.0	100.0	n/a	100.0	100.0	20.1	100.0
Total	93.2	81.0	68.4	96.1	100.0	100.0	91.3



Couverture de code ²

- x **Statements**
 - x Toutes les instructions ont-elles été exécutées ?
- x **Branches**
 - x Vérifie les alternatives des branchements conditionnels (if, ?:)
- x **Conditions**
 - x Vérifie les différentes possibilités dans les expressions logiques
- x **Subroutines**
 - x Toutes les fonctions ont-elles été appelées ?
- x **POD**
 - x Utilisation du module [POD::Coverage](#)



Documentation

- x **Du code testé et couvert, c'est déjà pas mal**
 - x Du code documenté, c'est mieux !
- x **Documentation écrite en POD (« *Plain Old Doc* »)**
 - x Il faut en vérifier la [syntaxe](#) à l'aide du module [Test::POD](#)
 - x C'est le rôle du test `t/pod.t` (créé par [Module::Starter](#))
- x **Couverture de POD**
 - x Tâche assumée par le module [Test::POD::Coverage](#)
 - x Vérifie que toute fonction possède une documentation associée
 - x En pratique, vérifie
 - x `=item foo ... =cut`
 - x `=head foo ... =cut`
 - x C'est le rôle du test `t/pod-coverage.t` (créé par [Module::Starter](#))



Kwalitee

- x **Pour une définition plus exhaustive : CPANTS**
 - x « *CPAN Testing Service* » – <http://cpants.perl.org/kwalitee.html>
 - x Définit des indicateurs de Kalité (« *ALPHA – Hic sunt dracones!* »)
- x **Obtenir les indicateurs : le module `Test::Kwalitee`**
 - x Ajouter le test `t/kwalitee.t` à la suite de tests :
 - x

```
eval { require Test::Kwalitee };  
exit if $@;  
Test::Kwalitee->import;
```
 - x
 - ok 1 - extractable
 - ok 2 - has_readme
 - ok 3 - has_manifest
 - ok 4 - has_meta_yml
 - ok 5 - has_buildtool
 - ok 6 - has_changelog
 - ok 7 - no_symlinks
 - ok 8 - has_tests
 - ok 9 - proper_libs
 - ok 10 - no_pod_errors
 - ok 11 - use_strict
 - ok 12 - has_test_pod
 - ok 13 - has_test_pod_coverage



Assertions

- x **Décrivent les hypothèses de travail d'une fonction**

- x Ses limites, ce qu'elle ne sait pas faire « *du tout du tout* » !
- x Il vaut mieux planter le programme dès que possible
- x Plutôt que de le laisser s'embarquer vers l'imprévisible
- x Un plantage à cause d'une assertion est plus facile à résoudre
- x « *Les programmes morts ne racontent pas de craques !* »
– Hunt & Thomas in The Pragmatic Programmer

- x **Assertions du module `Carp::Assert::More`**

- x Simples.....`assert_ + (is, isnt, like, defined, nonblank)`
- x Numériques.....`assert_ + (integer, nonzero, positive, ...)`
- x Références..`assert_ + (isa, nonempty, nonref, hashref, listref)`
- x Array/hash.....`assert_ + (in, exists, lacks)`



Test::LectroTest ¹

- x **Les tests traditionnels sont dits « dirigés »**
 - x Séquences de stimuli et de comparaisons à des valeurs attendues
 - x Mais on ne pense pas forcément à tout (# de combinaisons)
- x **Une alternative : des tests aléatoires contraints**
 - x Ou en Anglais, CRT : « *Constrained Random Testing* »
 - x On laisse la machine faire le sale boulot, (pseudo-)aléatoirement
- x **La recette avec Test::LectroTest**
 - x Associer un type à chacun des paramètres (des *types* en Perl ?)
 - x Ajouter des contraintes aux paramètres (~ sous-ensembles)
 - x Faire N itérations, mesurer la CF, bidouiller les contraintes, goto 0
- x **Pas encore utilisé en production** ☹
 - x Son alter-ego dans le monde matériel l'est (codé en [SystemVerilog](#))



Test::LectroTest²

x Le code suivant

```
x use Test::LectroTest::Compat; # ::Compat permet d'interfacer Test::More
use Test::More tests => 1;
```

```
sub ref_foo {
  { bar => 'foo', baz => 'gabuzomeu' }->{shift()}; ← Modèle de référence
}
```

```
my $property = Property {
```

```
  ##[ s <- Elements( 'foo', 'bar', 'baz' ) ]## ← Contraintes sur les entrées
```

```
  is( foo( $s ), ref_foo( $s ), 'foo / ref_foo' ) ← Comparaison à la référence
```

```
}, name => 'toutes les entrées possibles de foo' ;
```

```
holds( $property, trials => 100 ); # vérifie que la propriété "tient" sur 100 entrées aléatoires
```

x Prouve que foo ('bar') ≠ 'foo'

```
x # Failed test 'property 'toutes les entrées possibles de foo' falsified in 4 attempts'
# in lt_foo.t at line 30.
# got: undef
# expected: 'foo'
# Counterexample:
# $s = "bar";
```

x CF ≈ statistiques sur les ≠ valeurs des entrées



Reformuler

- x **Reformuler tôt, reformuler souvent**
 - x Pour combattre sans relâche l'entropie toujours grandissante
 - x À cause de : résolution de bugs, nouvelles fonctions ou « *ruses* »
 - x La suite de tests *devrait* assurer le respect de l'intégrité (cf C.F.)
- x **Seulement sur une branche de développement !**
 - x On ne touche pas à une branche de production, sauf bug avéré
- x **Ni rajouter de fonctions, ni résoudre des bugs !**
 - x Seulement rendre le code plus concis/lisible/testable... élégant !
 - x « *La simplicité est le pré-requis de la lisibilité* » – [EWD498](#)
 - x Progresser par petits incréments (plus facile a tracer/défaire)
- x **Pour en savoir plus...**

- x [Présentation](#) de Michael G Schwern : “Tales Of Refactoring!”



En résumé...

x **A priori**

- x Lire, apprendre et poser des questions (puis « évoluer » ☺)
- x Utiliser des outils éprouvés (SCM, RT, éditeurs, etc.)
- x Avoir de « bonnes » pratiques (i.e., consistantes)
- x Ne pas réinventer la roue
- x Écrire les tests (voire même la documentation) en premier

x **A posteriori**

- x Utiliser le protocole de test TAP et ses interpréteurs associés
- x Analyser les taux de couverture de code (\neq CF !) et de POD
- x Insérer des assertions dans le code
- x Voire même laisser la machine faire le sale boulot à votre place !
- x Générer des rapports de test synthétiques et lisibles



Ingénierie sociale

- x **Comme dans beaucoup d'activités humaines**
 - x Il y a la technique **et** il y a l'engagement
- x **La technique**
 - x On vient d'en parler pendant 40 (longues, non ?) minutes
 - x Et c'était loin d'être exhaustif !
- x **L'engagement**
 - x Sans la *motivation*, point de Kalité !
 - x C'est un *chemin* (à suivre) et non pas une *destination* (à atteindre)
- x **Une dernière citation pour conclure**
 - x « À cette époque (1909), l'ingénieur en chef était aussi presque toujours le pilote d'essai. Cela eu comme conséquence heureuse d'éliminer très rapidement les mauvais ingénieurs dans l'aviation. »
– Igor Sikorsky



Questions ?





Sur la toile...

- x **Les hoplites de la Phalange veulent en découdre...**

- x Kwalitee : <http://qa.perl.org/phalanx/kwalitee.html>

- x **Modules CPAN**

- x `Module::Starter`, `Module::Starter::PBP`

- x `Carp::Assert`, `Carp::Assert::More`

- x `Devel::Cover`

- x `Test::More`, `Test::Harness`

- x `Test::POD`, `Test::POD-Coverage`

- x `Test::TAP::Model`, `Test::TAP::HTMLMatrix`

- x `Test::LectroTest`

- x **Talks**

- x `Test::Tutorial`, `Devel::Cover` & `Test::LectroTest`