

# Principes, techniques et outils de test

## Outils de test et d'analyse logiciels pour la productivité et la qualité

**Klaus Lambertz,**  
Co-fondateur de Verifysoft



Mulhouse, le 30 novembre 2009





# Agenda

- Verifysoft et ses partenaires
- Coût de l'erreur logiciel
- Les outils de test dans le cycle de développement
- Complexité du code / Analyse statique
- Test unitaire et couverture de test
- Test fonctionnel
  
- Questions / Discussion

# Verifysoft Technology

- Verifysoft Technology GmbH
  - Créée en 2003
  - Offenburg (Allemagne)
  - env. 200 clients en Europe
- Distributeur et développeur des outils de test logiciel
  - support, conseils et séminaires





# Partenaires

- Testwell, Tampere (Finlande) 
- Conformiq, Saratoga (USA) 
- Coverity, San Francisco (USA) 

# Nos clients



Domaines principaux d'application :

Développement des applications „critiques“

- Aéronautique
- Automotive
- Médical
- Nucléaire



# Nos références



et beaucoup d'autres...

# Coût de l'erreur logiciel

- Les pertes liées aux erreurs de programmation augmentent chaque année



Estimation : ???



# Coût de l'erreur logiciel

- **100-150.000.000.000 Euros/an en Europe**  
dû aux erreurs de logiciels

→ Pour y pallier, le meilleur moyen est:  
le „savoir-faire“ des développeurs !!!

(Les Hatton, Kingston University London)



# Coût de l'erreur logicielle

L'erreur logicielle

coûte jusqu'à **14.000.000 Euro**

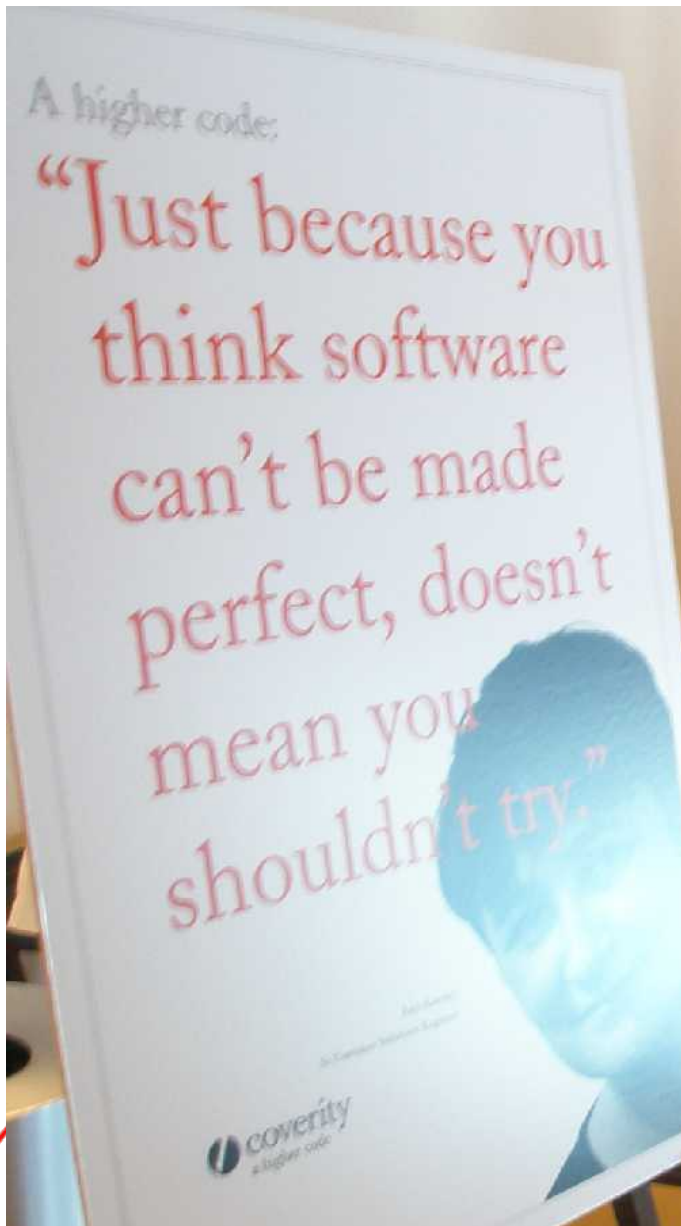
par entreprise et par an

50% des sociétés trouvent dans la première année d'utilisation jusqu'à dix erreurs critiques

IDC-Study „Improving Software Quality to Drive business Agility“,

sondage 2008 sur des sociétés américaines comptant 250-10.000 employés [www.idc.com](http://www.idc.com)

# Coût de l'erreur logiciel



40-60% du budget global est généralement consacré au test et à la correction.

Le but du test est d'arriver à un produit

**„zéro défaut“**



# Le test avec des outils

Le test est nécessaire  
pour obtenir des certifications

IEC 61508

EN 50128 (ferroviaire)

IEC 62304 (médical)

EN 62138 (nucléaire)

DO-178B (aéronautique)



# Le test avec des outils

Le test: indispensable  
mais ... très coûteux  
activité répétitive

→ Automatisation du test  
utilisation d'outils de test !



# Cycle de développement

Analyse des besoins  
et faisabilité

Recettes

Spécifications

Tests de  
validation

Conformiq  
Qtronic

Test fonctionnel

Conception  
architecturale

Tests de  
validation

Conception  
détaillée

Tests  
unitaire

Testwell CTA++

Test unitaire

Testwell CMT++

Couverture de test

Codage

Testwell CMT++  
CMTJava

Mesure  
de complexité

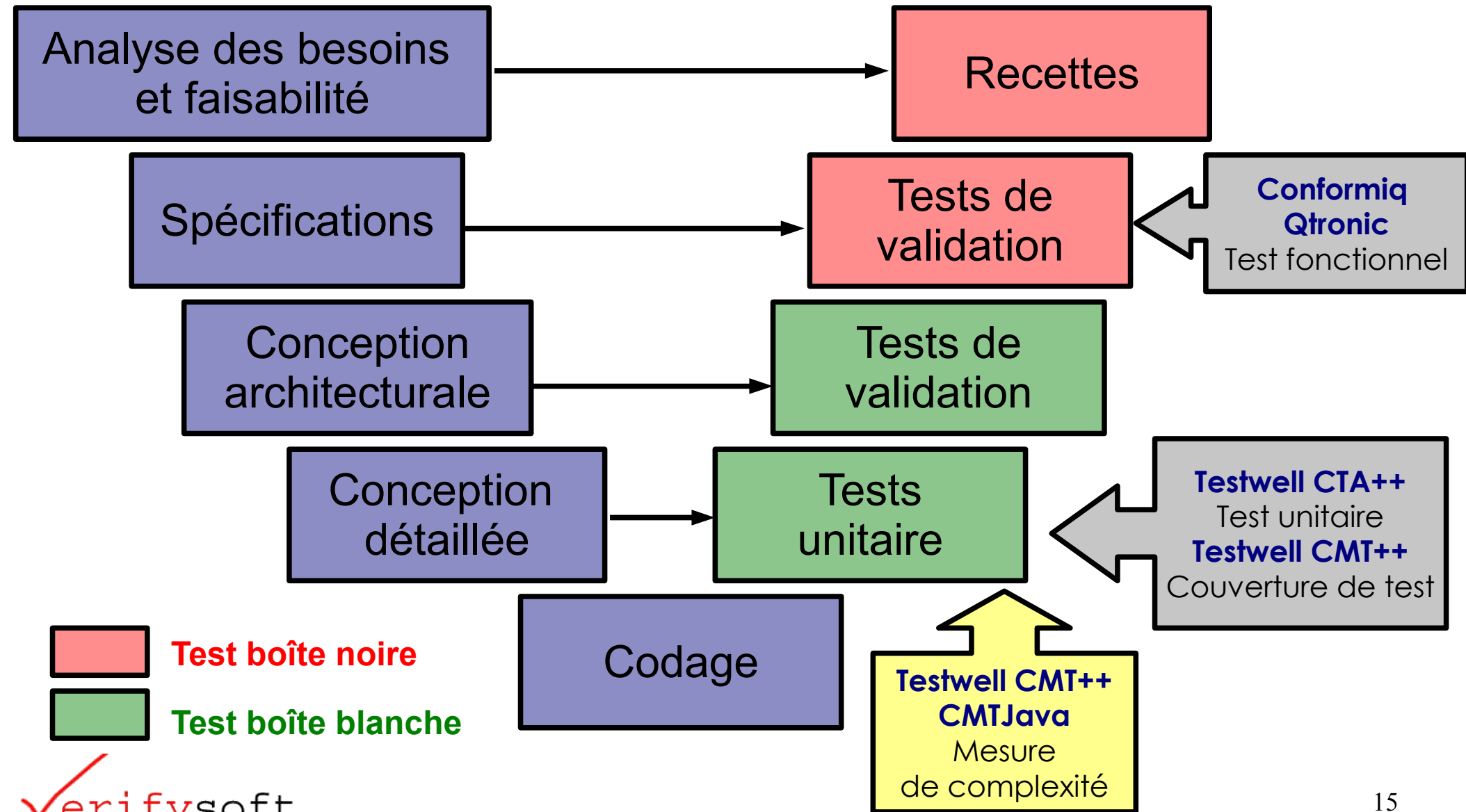


Test boîte noire



Test boîte blanche

# Cycle de développement



# Testwell CMT++ / CMTJava

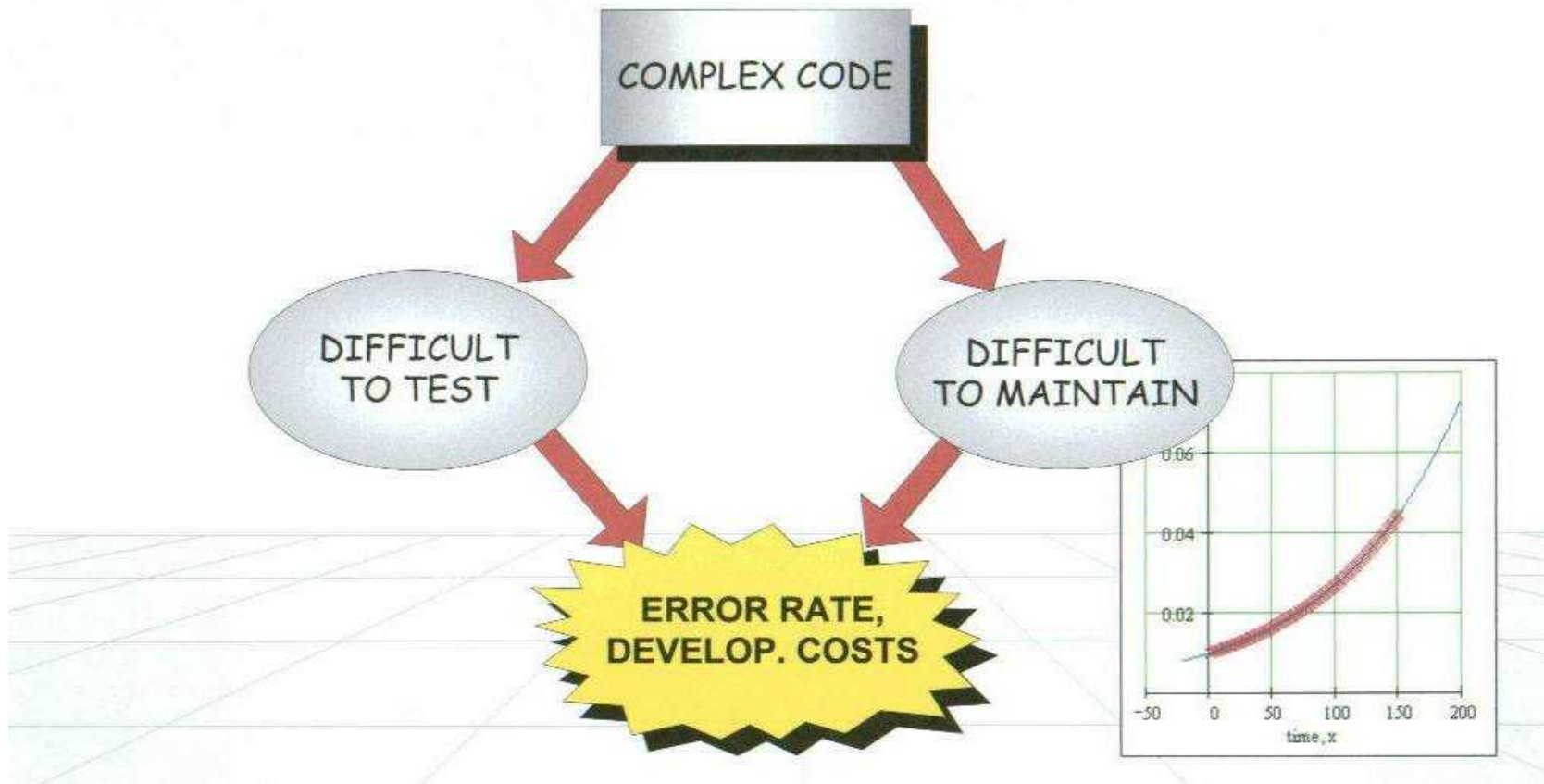
Testwell CMT++  
Testwell CMTJava



Mesure de complexité du code  
pour C/C++ / Java

# Testwell CMT++ / CMTJava

Wish to locate complex code





# Testwell CMT++ / CMTJava

- Pourquoi analyser la complexité du code?
  - La complexité du code est en corrélation avec le taux d'erreur et la robustesse de l'application
- Un code complexe est difficile à tester
  - plus d'erreurs dans l'application finale
- Un code complexe est difficile à maintenir
- La complexité du code est souvent la raison d'erreur
- Testwell CMT++ et CMTJava permettent donc de faire des économies.



# Testwell CMT++ / CMTJava

Testwell CMT++ et CMTJava  
analysent les métriques suivantes:

- \* Métriques de lignes de code (LOC)
- \* Métriques d'Halstead
- \* Nombre cyclomatique de McCabe
- \* Indice de maintenabilité



# Testwell CMT++ / CMTJava

## Métriques des lignes de code (LOC)

- **LOCphy**  
nombre de lignes (number of physical lines)
- **LOCpro**  
nombre de lignes avec du code programme
- **LOCbl**  
nombre de lignes vides
- **LOCcom**  
nombre de lignes avec commentaires



# Testwell CMT++ / CMTJava

## Métriques des lignes de code (LOC)

Valeurs recommandées pour une fonction:  
entre 4 et 40 LOCpro

Une définition de fonction contient au moins un prototype, une ligne de code, et une paire d'accolades, qui font 4 lignes. Une fonction plus grande que 40 lignes de programme implémente probablement beaucoup de fonctions.



# Testwell CMT++ / CMTJava

## Métriques des lignes de code (LOC)

Valeurs recommandées pour un fichier:  
entre 4 et 400 LOCpro

La plus petite entité qui peut raisonnablement occuper un fichier source complet est une fonction, et la longueur minimum d'une fonction est de 4 lignes. Les fichiers plus longs que 400 lignes de programme (10..40 fonctions) sont habituellement trop longs pour être compris en totalité.

Commentaires: 30 à 75 %



# Testwell CMT++ / CMTJava

## Nombre cyclomatique de Mc Cabe v(G)

décrit la complexité d'écoulement  
(Control flow complexity)  
d'un programme

Plus le nombre cyclomatic est grand, plus il y aura de chemins d'exécution dans la fonction, et plus elle sera difficile à comprendre.



# Testwell CMT++ / CMTJava

## Nombre cyclomatique de Mc Cabe v(G)

- Valeur recommandée pour une fonction: <15  
Plus de 15 chemins sont difficiles à identifier et tester.
- Valeur recommandée pour un fichier: <100

# Testwell CMT++ / CMTJava

## Halstead-Metrics

- B nombre d'erreurs estimé
- D niveau de difficulté, prédisposition d'erreurs
- E effort pour implémenter
- L niveau du programme (représente le niveau du programme)
- N longueur du logiciel
- **N1 nombre d'opérateurs**
- **N2 nombre d'opérandes**
- n taille de vocabulaire ou nombre d'opérateurs uniques et d'opérandes
- n1 nombre d'opérateurs uniques
- n2 nombre d'opérandes uniques
- T temps nécessaire pour l'implémentation (temps nécessaire pour comprendre)
- V volume: taille de l'implementation d'un algorithme

# Testwell CMT++ / CMTJava

## Index de maintenabilité (MI)

Indique quand il est moins coûteux et risqué de re-écrire le code au lieu de garder des parties complexes du code

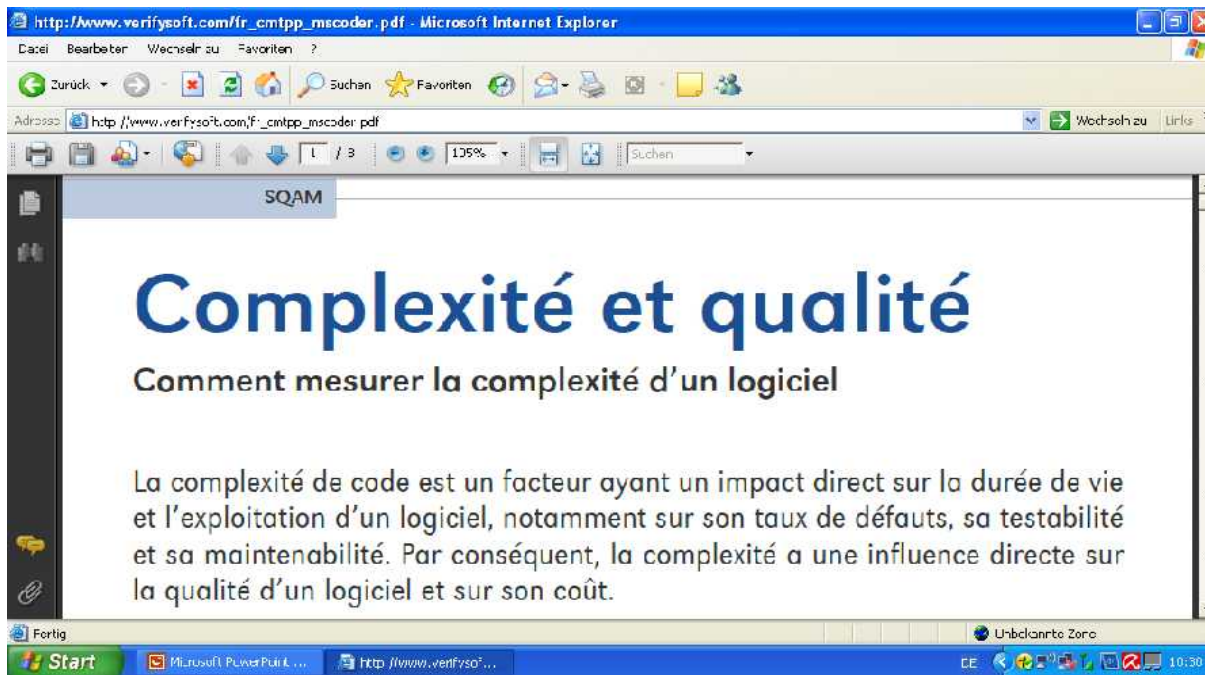
- 85 et plus bonne maintenabilité
  - 65-85 maintenabilité modérée
  - < 65 difficile à maintenir
- avec des parties de code vraiment mauvaises (grosses, non commentées, non structurées), la valeur MI peut même être négative

# Testwell CMT++ / CMTJava

Pour plus d'information



[www.verifysoft.com/fr\\_cmtx.html](http://www.verifysoft.com/fr_cmtx.html)



[www.verifysoft.com/  
fr\\_cmtpp\\_mscoder.pdf](http://www.verifysoft.com/fr_cmtpp_mscoder.pdf)



Testwell CTC++, CMT++ and CMTJava are products of Testwell Oy, Tampere (Finland)

# Testwell CMT++ / CMTJava

CMT++ Summary Report

file:///C:/Users/ed/cmt/CMTHTML/index.html

Links anpassen Weitere Lesezeichen

```

*****
*          CMT++, Complexity Measures Tool for C/C++, Version 4.2          *
*                                                                           *
*          COMPLEXITY MEASURES REPORT                                     *
*                                                                           *
*          Copyright (c) 1993-2007 Testwell Oy                          *
*****

License notice: This is a limited period evaluation copy license.

The input CMT++ report was produced at Mon Jan 12 17:11:08 2009
cmt options: -o C:\Users\ed\cmt\report.tmp -f C:\Users\ed\cmt\files.txt
Html'ized by cmt2html v2.2 at Mon Jan 12 17:11:08 2009
cmt2html options: -i C:\Users\ed\cmt\report.tmp

This is SUMMARY view. Go to DETAILED view. See instructions.

Alarms-# Measured object v(G) LOCphy LOCpro c% V B MI
-----
[Progress Bar] calibrate.c 20 172 107 - 3195 1.24 110
[Progress Bar] do_mounts.c 69 406 321 - 11535- 3.18- 105
[Progress Bar] do_mounts_md.c 47 280 213 - 8280- 2.39- 88
[Progress Bar] do_mounts_rd.c 46 429 313 - 11742- 3.05- 107
[Progress Bar] msgutil.c 15 127 98 - 2751 1.02 95
[Progress Bar] OVERALL (24 %)

OVERALL SUMMARY:

Measure 5 Files 40 Functions
Alarmed % Limits Alarmed % Limits
-----
Cyclomatic number v(G) 0 0 1-100 4 10 1-15
Program lines LOCpro 0 0 4-400 12 30 4-40
Comment % 5 100 30-75 19 47 30-75
Volume V 3 60 100-8000 9 22 20-1000
Estimated number of bugs B 3 60 0-2 0 0 n/a
Maintainability index MI 0 0 65- 1 0 65-
    
```

report - Editor

```

Date: 12/12/2009 17:18:18
Format: HTML
View: Default
-----
*          CMT++, Complexity Measures Tool for C/C++, Version 4.2          *
*                                                                           *
*          COMPLEXITY MEASURES REPORT                                     *
*                                                                           *
*          Copyright (c) 1993-2007 Testwell Oy                          *
*****

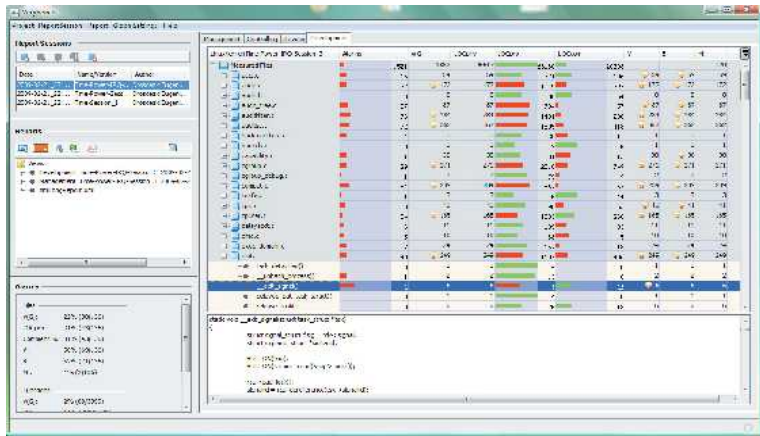
License notice: This is a limited period evaluation copy license.

This report was produced at Mon Jan 12 17:18:18 2009
Options: -o C:\Users\ed\cmt\report.txt -f C:\Users\ed\cmt\files.txt

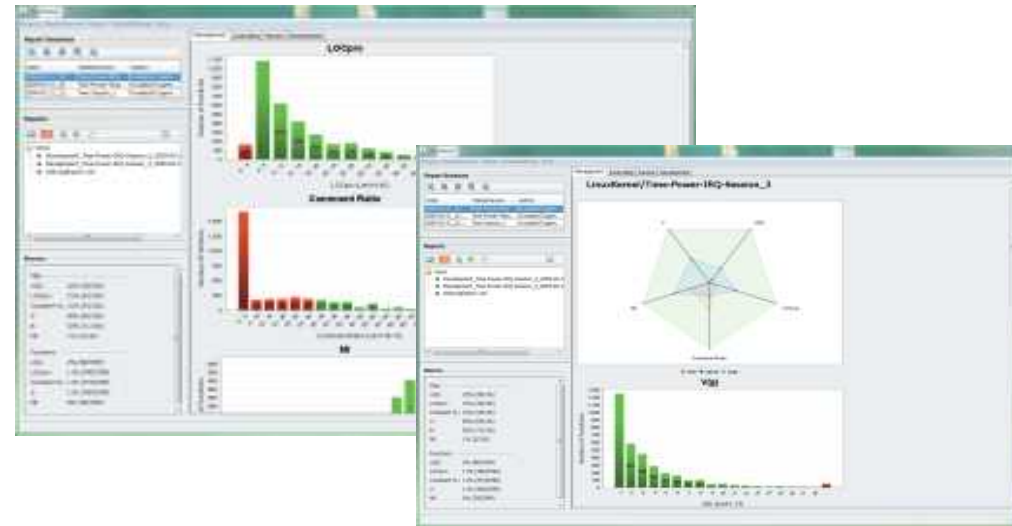
File: C:\Users\ed\Desktop\TestFiles\linux-2.6.26.8\linux-2.6.26.8\init\calibrate.c
-----
Line Measured object v(G) LOCphy LOCpro c% V B MI
-----
13 1pj_setup() 1 5 5 92 0.07 121
31 calibrate_delay_direct() 9 70 42 1123 0.30 101
104 calibrate_delay_direct() 1 1 1 38 0.01 152
114 calibrate_delay() 10 59 47 1148 0.35 86
-----
172 calibrate.c 20 172 107 - 3195 1.24 110
-----

File: C:\Users\ed\Desktop\TestFiles\linux-2.6.26.8\linux-2.6.26.8\init\do_mounts.c
-----
Line Measured object v(G) LOCphy LOCpro c% V B MI
-----
31 load_ramdisk() 1 5 5 104 0.07 171
38 readonly() 2 7 7 90 0.02 116
45 readwrite() 2 7 7 90 0.02 115
57 name_to_dev() 27 86 55 1837 0.57 90
114 root_dev_setup() 1 5 5 88 0.02 121
132 rootwait_setup() 2 7 7 88 0.02 116
163 mount_attr_setup() 1 5 5 61 0.01 129
170 ls_names_setup() 1 5 5 61 0.01 123
177 root_delay_setup() 1 5 5 92 0.02 121
187 get_fs_names() 7 26 24 115 0.15 83
214 do_mount_root() 3 14 13 520 0.12 95
229 mount_block_root() 9 24 44 1127 0.30 91
281 mount_nfs_root() 3 10 9 166 0.04 106
296 change floppy() 3 27 27 889 0.22 82
323 mount_root() 9 28 26 447 0.07 98
354 prepare_namespac() 14 52 18 955 0.19 91
-----
406 do_mounts.c 69 406 321 - 11535- 3.18- 105
    
```

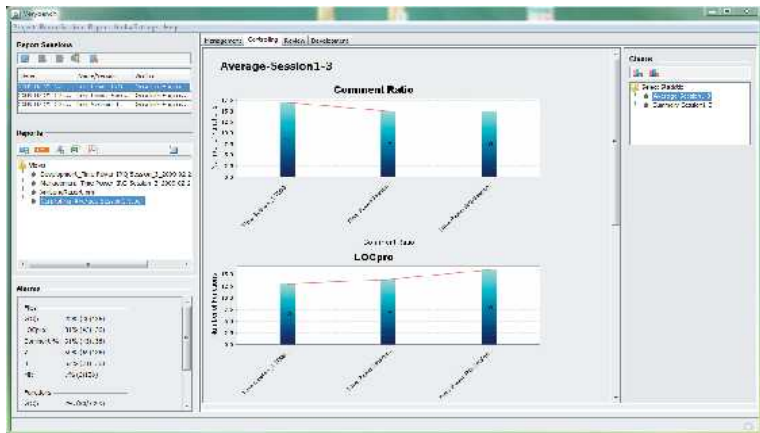
# Testwell CMT++ / CMTJava



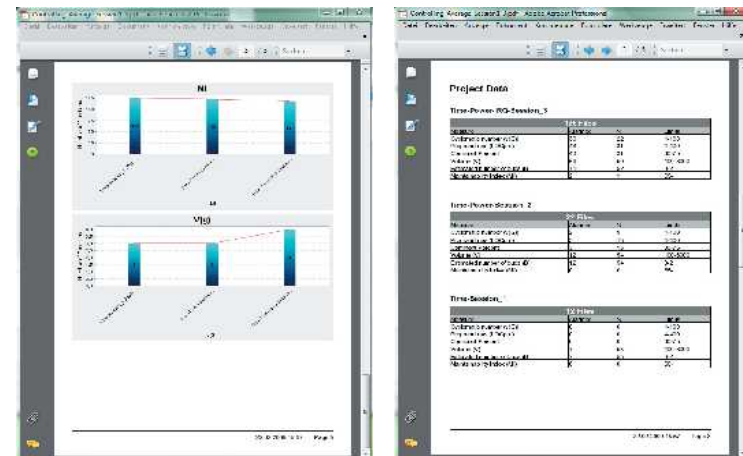
Point de vue „Développeur“



Point de vue „Manager“

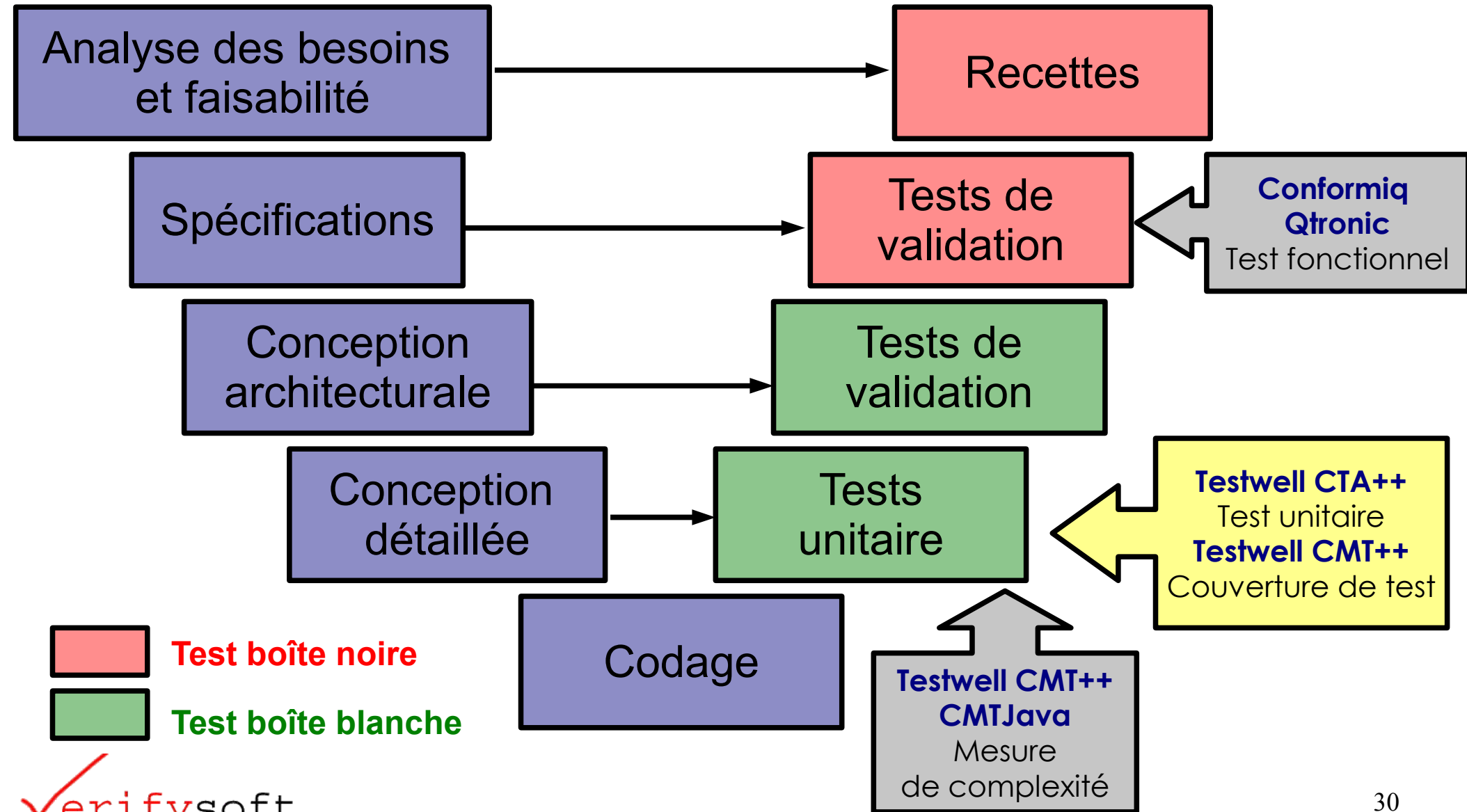


Point de vue „Audit“



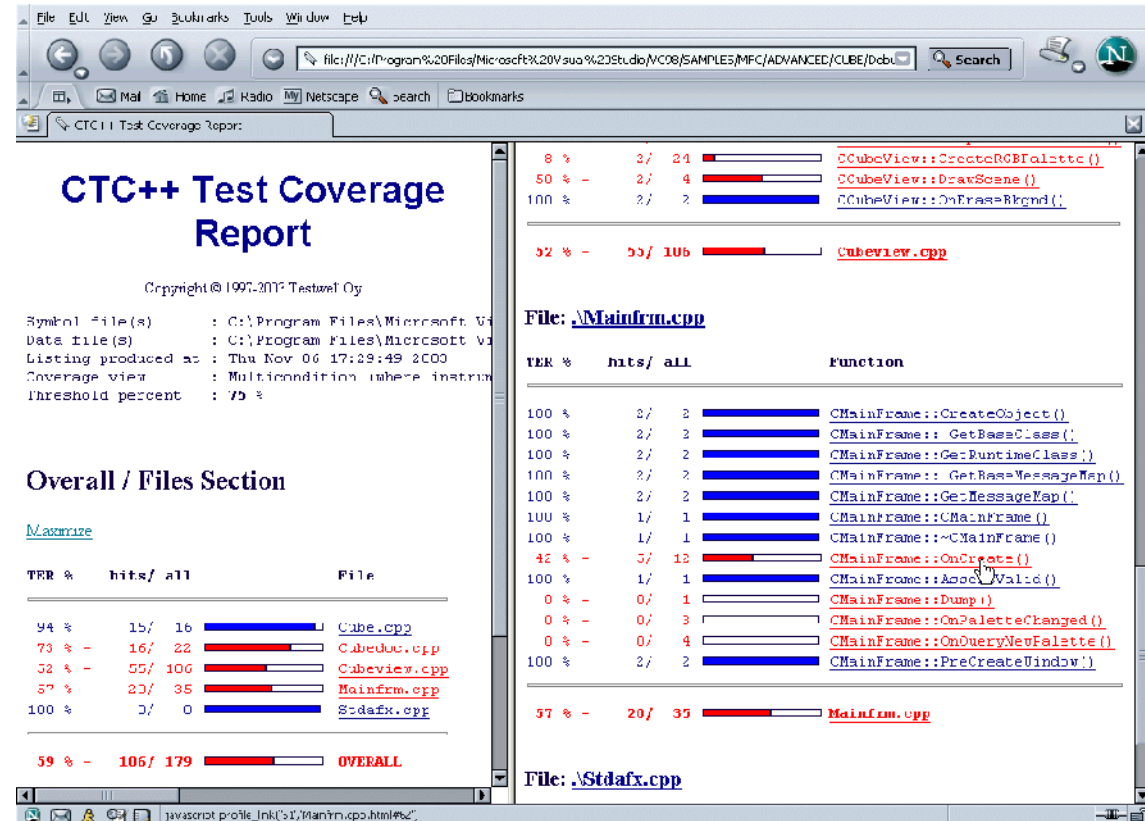
Tous ses rapports sont disponibles en pdf

# Cycle de développement



# Testwell CTC++ Couverture de test

## Testwell CTC++ Couverture de test pour C and C++



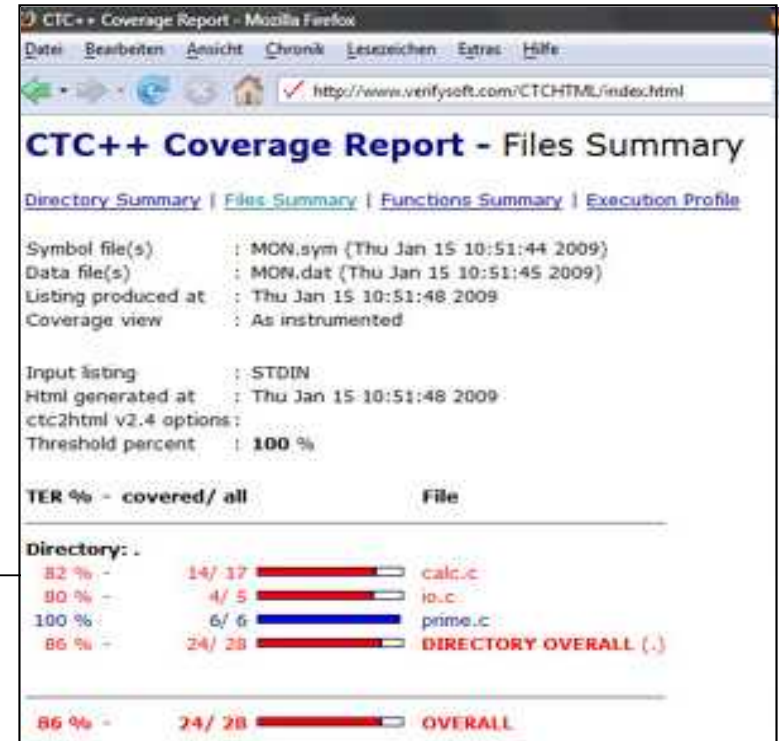
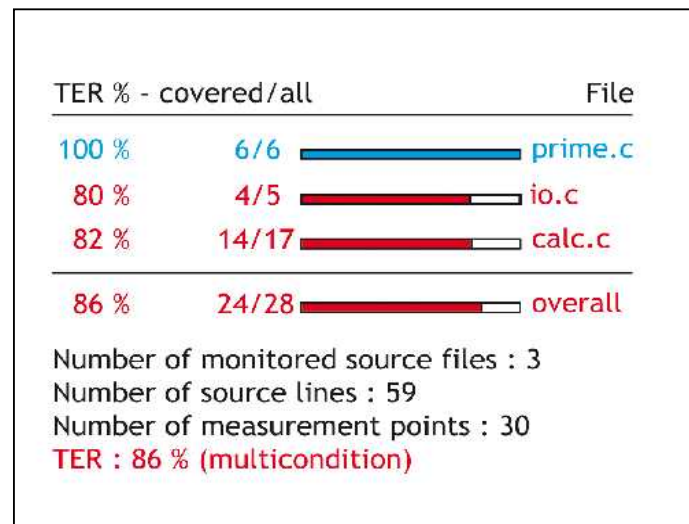
## CTC add-on pour Java et C#

# Testwell CTC++ Couverture de test

## Couverture de test / code

montre les parties du code

- testées/non testées
- exécutées/pas exécutées



# Testwell CTC++ Couverture de test

## Pourquoi mesurer la couverture de code?

- Preuve que 100% (ou “x”% selon exigences) ont été testé
- Vous pourrez écrire de meilleurs tests/cas de tests (plus adaptés)
- Vous savez quand vous pouvez arrêter de tester  
(critère de fin de tests)
- Vous évitez de passer du temps à écrire des cas de tests redondants
- Vous pouvez prouver à vos clients que les codes ont été testés  
conformément à leurs attentes
- En tant que clients, vous pouvez être certains que les codes délivrés par votre sous  
traitant sont conformes à vos attentes
- Nécessaire pour obtenir une certification (par ex. DO178-B)

# Testwell CTC++ Couverture de test

## Analyse de Test pour tous les niveaux (- C3):

- Couverture de fonction
- Couverture de décision
- Couverture de condition/de branche  
(Condition/Branch Coverage)
- Couverture de condition/décision modifiée  
(Modified Condition/Decision Coverage, MC/DC)
- Couverture de condition multiple  
(Multicondition Coverage, MCC)

# Testwell CTC++ Couverture de test



Testwell CTC++ vérifie tous les niveaux du code et peut être utilisé pour des certifications

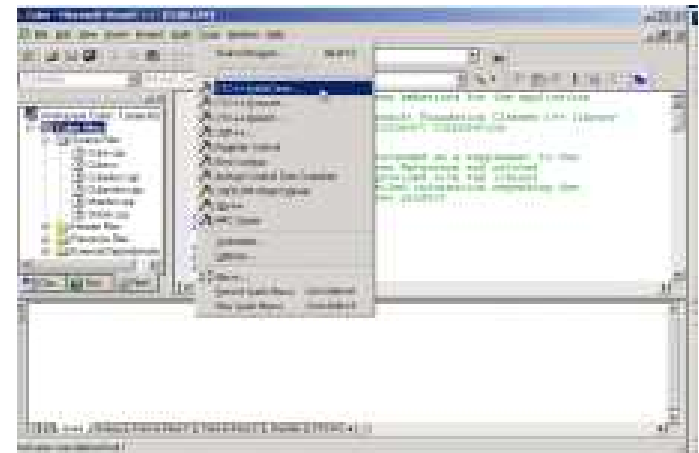
Aéronautique, Automotive, Médical, ...  
par ex. DO178-B (tous les niveaux)



# Testwell CTC++ Couverture de test

Testwell CTC++ est simple à l'emploi:

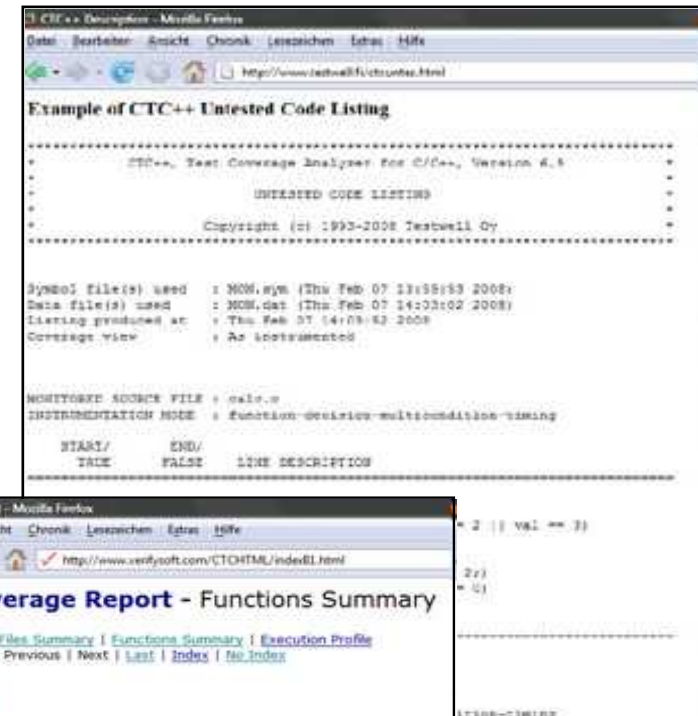
- Aucune modification de code nécessaire
- Support existant makefiles
- GUI integration in important IDEs
  - Microsoft Visual Studio
  - WindRiver Tornado
  - Borland C++ 5.02
  - Metrowerks CodeWarrior
  - Eclipse



# Testwell CTC++ Couverture de test

Sorties: text, XML ou HTML:

- montre les codes non testés  
(mise en évidence)
- montre combien de fois la partie  
du code à été testée
- différents rapports de couverture
  - summary-levels for files
  - functions and whole application
  - Execution Profile Listing



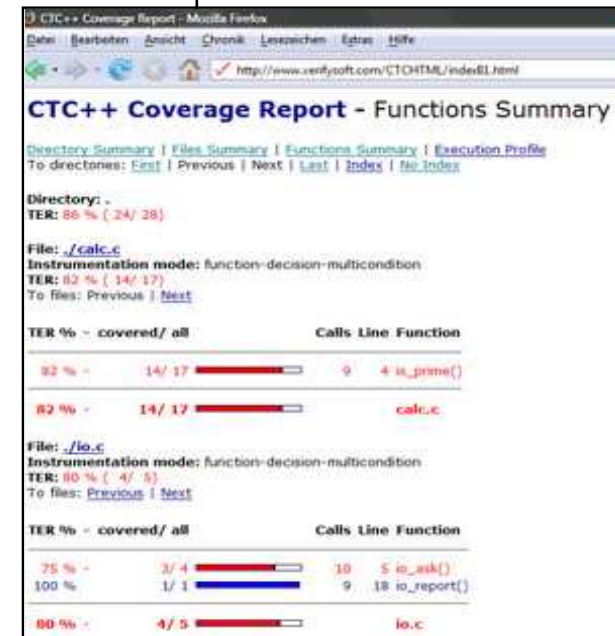
Example of CTC++ Untested Code Listing

```
CTC++, Test Coverage Analyzer for C/C++, Version 4.5
UNTESTED CODE LISTING
Copyright (c) 1993-2008 Testwell Oy

SYMBOL file(s) used : MON.sym (Thu Feb 07 13:55:53 2008)
Data file(s) used : MON.dat (Thu Feb 07 14:03:02 2008)
Listing produced at : Thu Feb 07 14:03:42 2008
Coverage view : As instrumented

NORETURN SOURCE FILE : calc.c
INSTRUMENTATION MODE : function-decision-multicondition-timing

START/ END/
TIME FALSE LINE DESCRIPTION
```



# Testwell CTC++ Couverture de test

## Testwell CTC++

### idéal pour les systèmes embarqués

- très faible coût d'instrumentation
- fonctionne avec toutes les cibles / microcontrôleurs
  - “host target add-on” est fourni en code source
    - peut être facilement adapté à de nouvelles cibles
- fonctionne avec les plus petites cibles
- fonctionne avec tous les compilateurs



# Testwell CTC++ Couverture de test

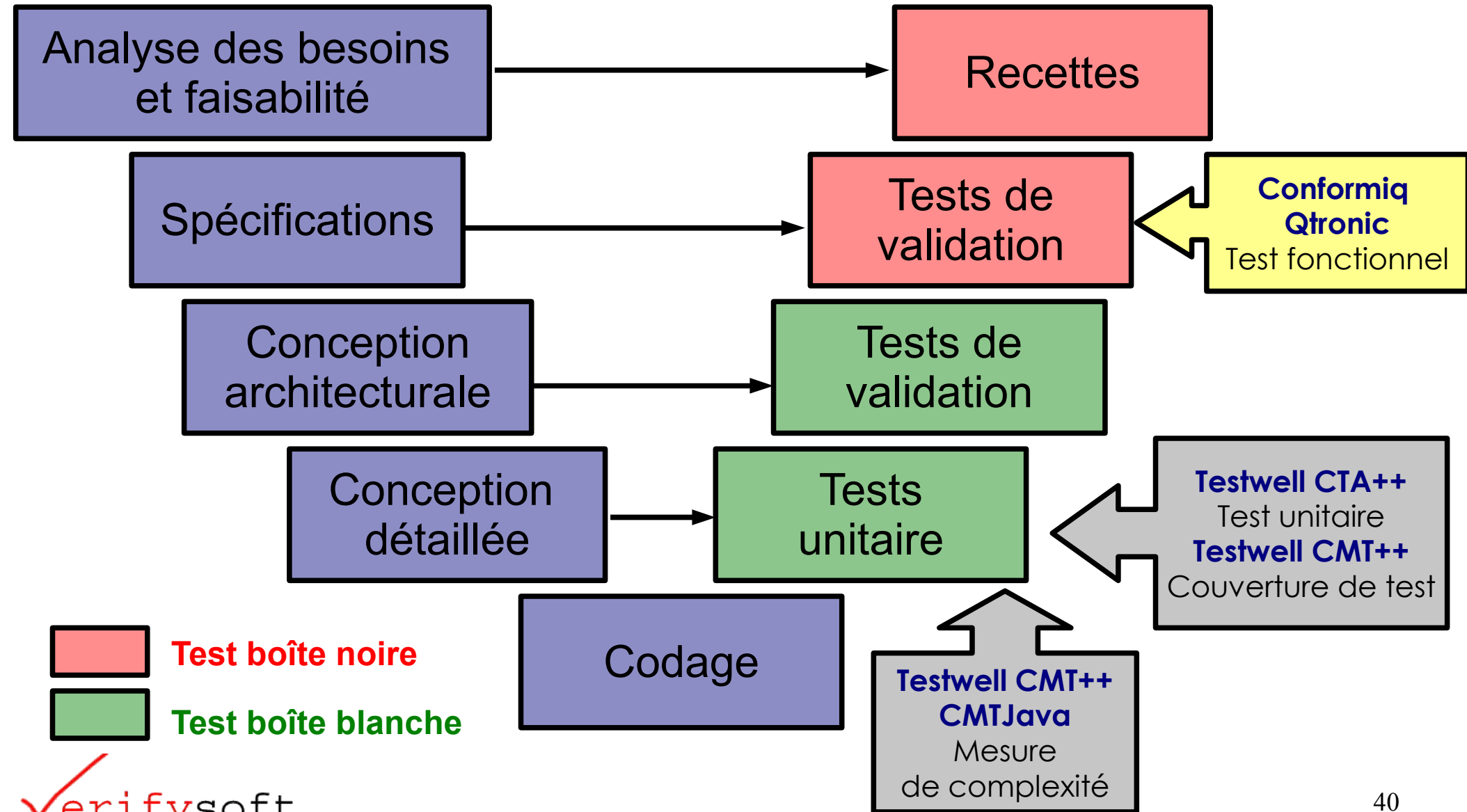
## CTC++ Add-on for Java et C#

extension Testwell CTC++ pour Java et C#

→ Vous avez seulement besoin d'un outil de couverture pour C, C++, Java, C#, ...



# Cycle de développement





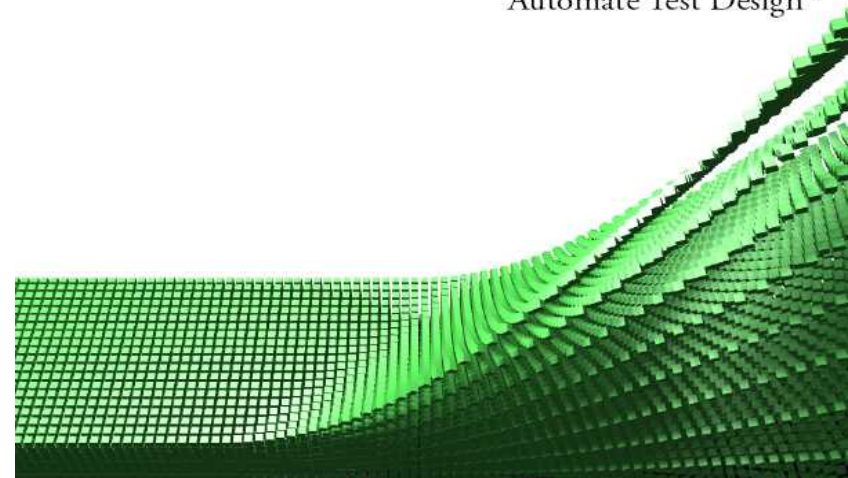
# Conformiq Qtronic

## Générateur automatique de cas de test

pour les test fonctionnels

(tests de boîte noire)

CONFORMIQ  
QTRONIC™  
Automate Test Design™



# Conformiq Qtronic

Les cas de tests manuels prennent du temps ...  
et entraînent des risques:

💣 tests incorrects

💣 tests oubliés

🕒 tests redondants

🕒 la maintenance pour les scripts prend du temps





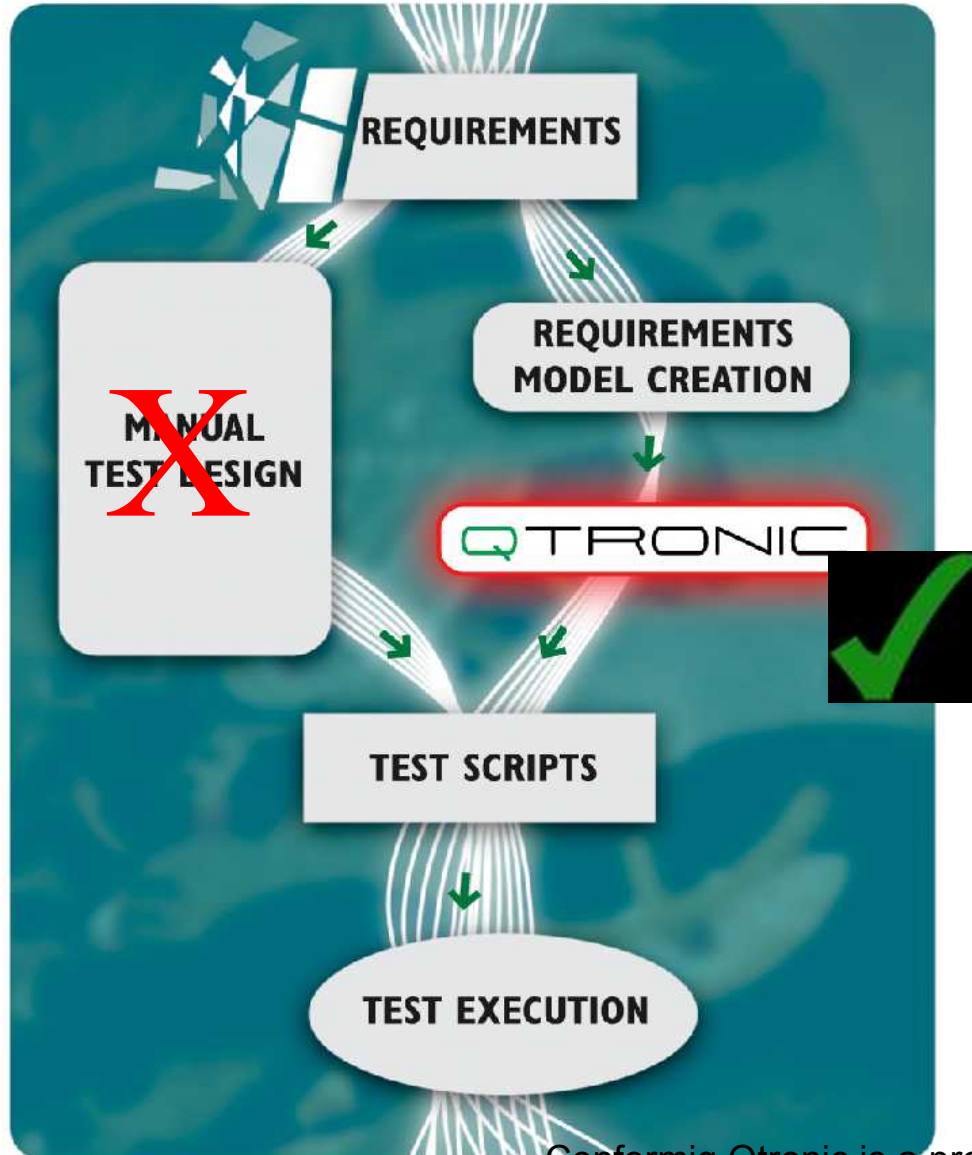
# Conformiq Qtronic

Notre solution: **Automated Test Design**

→ model driven testing, model based testing,  
specification based testing,  
specification driven testing,  
...



# Manuel vs. Automatique



Génération automatique  
de cas de test basée sur  
des modèles

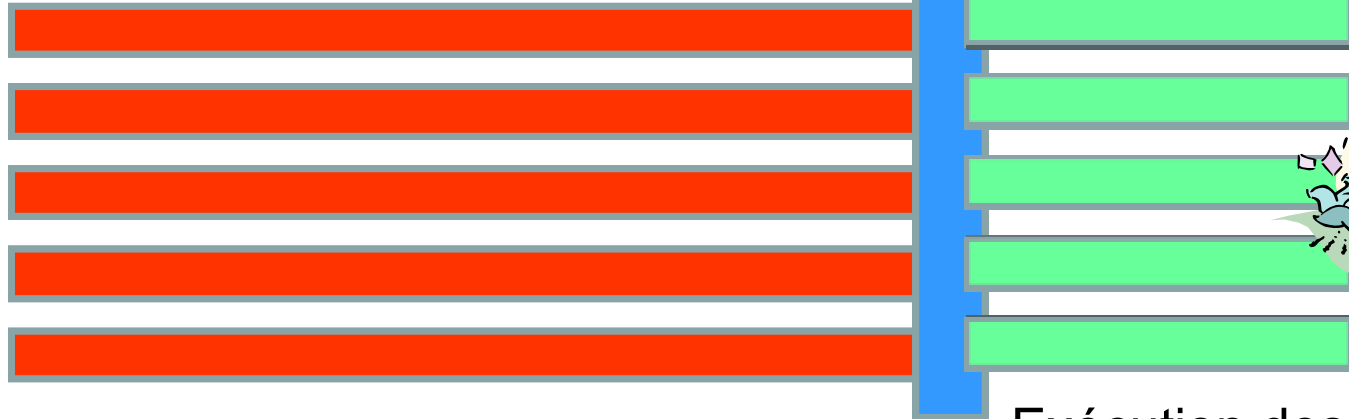
au lieu d'écrire les cas de  
tests manuellement



# Manuel vs. Automatique

Test Plan

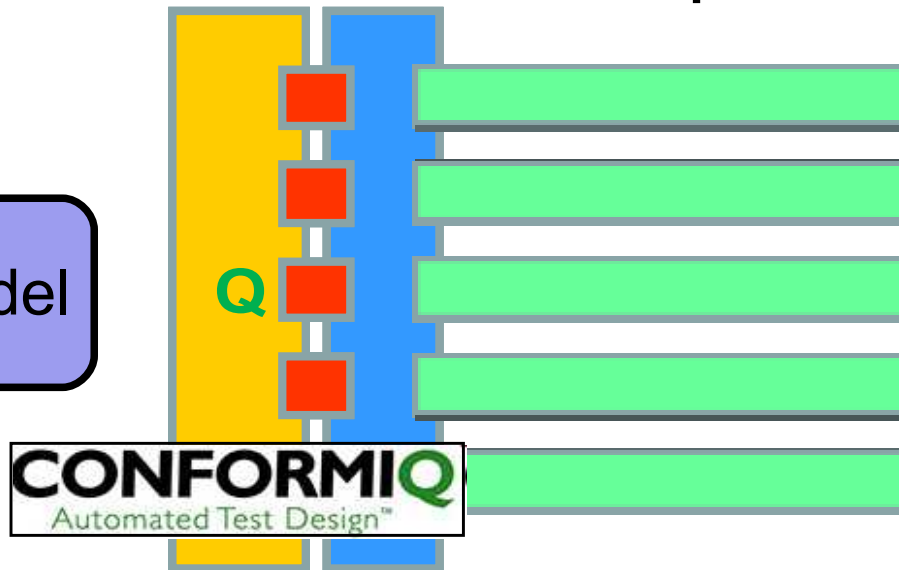
Cas de tests écrits manuellement



Exécution des scripts de tests

System Model

Génération automatique

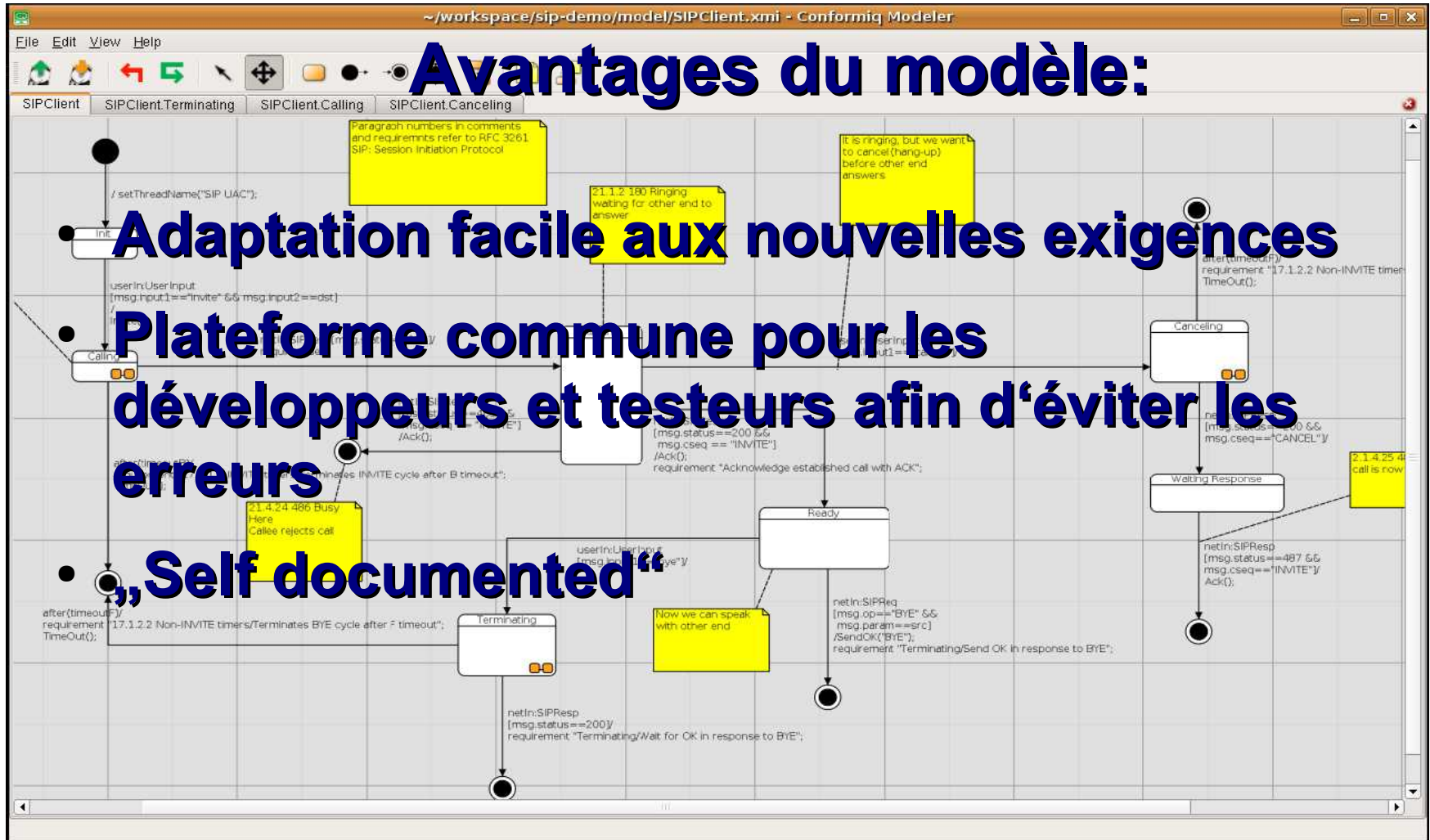


Exécution des scripts de tests

# Model Driven Testing

## Avantages du modèle:

- **Adaptation facile aux nouvelles exigences**
- **Plateforme commune pour les développeurs et testeurs afin d'éviter les erreurs**
- **„Self documented“**





# Modelage

- textuellement en Java (avec des éléments C#)
  - „Qtronic Modelling Language“ (QML)
- graphiquement: UML State Charts (optional)
- Le modèle peut être réalisé:
  - Text editor („Java“)
  - Qtronic Modeller (UML State Charts)
  - Third Party Modeling (UML) Tools

# Génération des tests

Processus:

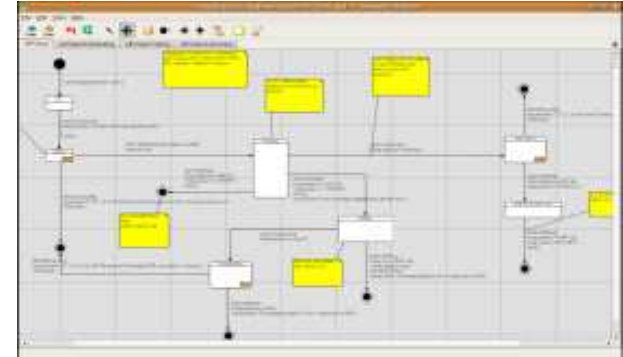
Création du modèle

Import du modèle

Choix des critères de couverture de test et des formats pour les scripts de test

Génération automatique de tests par Conformiq Qtronic

Exécution des tests avec votre environnement





# Génération des tests

Formats pour la génération de scripts de test:

Python

TCL

TTCN-3

C, C++

Visual Basic

Java

Junit

Perl

Excel

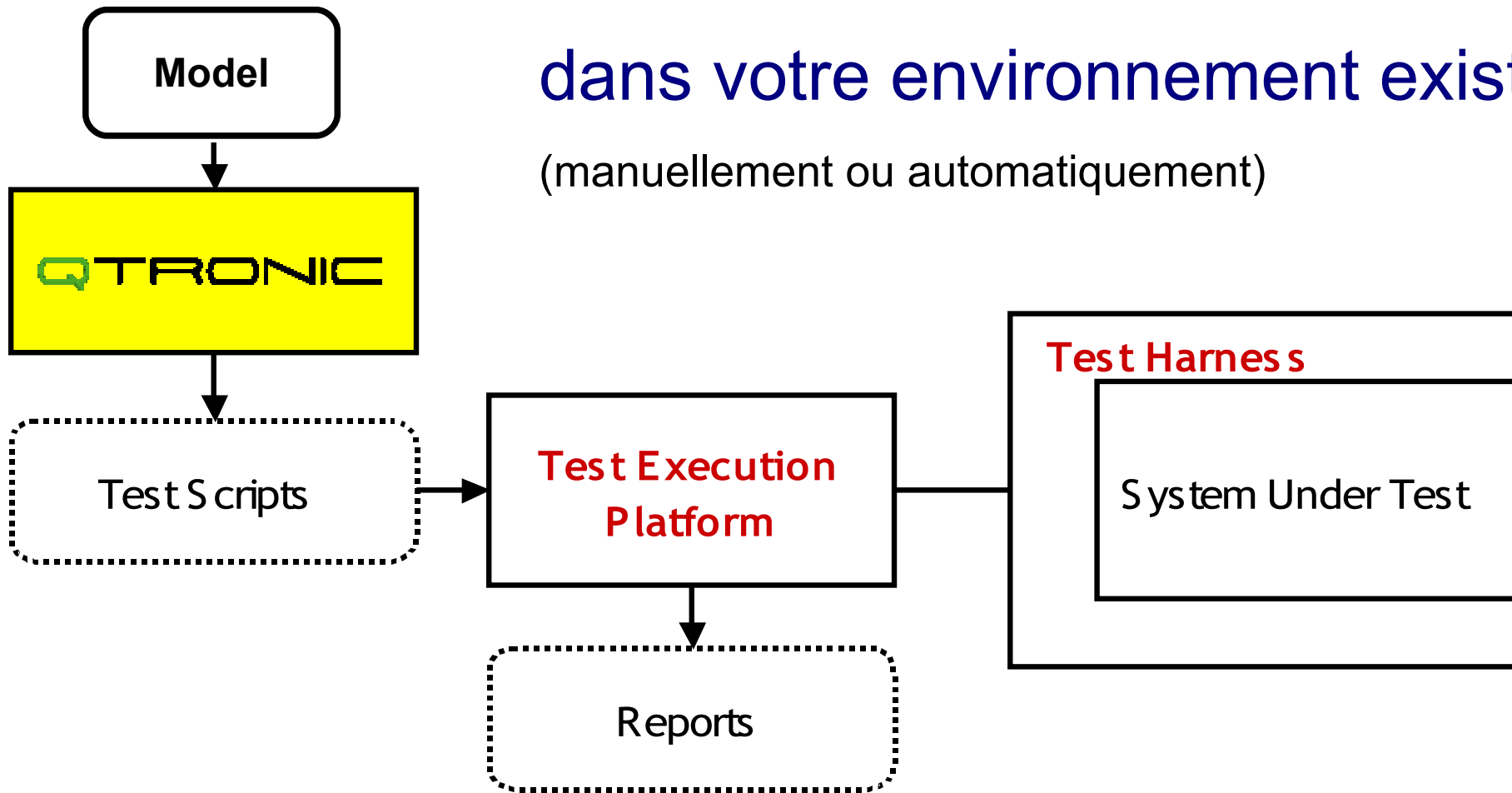
HTML

Word

Shell Scripts

# Exécution des cas de test

Exécution des cas de test  
dans votre environnement existant  
(manuellement ou automatiquement)



# Génération de cas de test

The screenshot displays the Qtronic Eclipse SDK interface for testing requirements. The main window is titled "Qtronic - Requirement Goals for SIPClient - Eclipse SDK".

**Requirements Goals Editor: SIPClient**

Testing Goals	Boundary Values	Requirement Coverage
Requirements	✓ 100	✓ 100
17.1.2.2 Non-INVITE timers	✓ 100	✓ 100
Resends CANCEL after E timeout	✓ ✓	✓ ✓
Terminates CANCEL cycle after F timeout	✓ ✓	✓ ✓
Resends BYE after E timeout	✓ ✓	✓ ✓
Terminates BYE cycle after F timeout	✓ ✓	✓ ✓
Terminating	✓ 100	✓ 100
Wait for OK in response to BYE	✓ ✓	✓ ✓
Send OK in response to BYE	✓ ✓	✓ ✓
17.1.1.2 INVITE timers	✓ 100	✓ 100

**Traceability: Requirement Coverage**

Testing Goals	1	2	3	4	5	6
Requirements						
17.1.2.2 Non-INVITE timers						
Resends CANCEL after E timeout						X
Terminates CANCEL cycle after F timeout						X
Resends BYE after E timeout				X		
Terminates BYE cycle after F timeout				X		
Terminating						
Wait for OK in response to BYE			X			
Send OK in response to BYE		X				
17.1.1.2 INVITE timers						

**Test Case 4**

Sequence diagram showing interactions between Tester and SIP UAC:

```
sequenceDiagram
    participant Tester
    participant SIP UAC
    Tester->>SIP UAC: UserInput -> userIn
    SIP UAC-->>Tester: SIPReq <- netOut
    Tester->>SIP UAC: SIPResp -> netIn
    SIP UAC-->>Tester: SIPResp -> netIn
    Tester->>SIP UAC: SIPReq -> netOut
    SIP UAC-->>Tester: SIPReq -> netOut
    Tester->>SIP UAC: UserInput -> userIn
    SIP UAC-->>Tester: SIPReq -> netOut
    Tester->>SIP UAC: SIPReq -> netOut
```

**Test Cases: SIPClient**

#	Name
1	Test Case 1
2	Test Case 2
3	Test Case 3
4	Test Case 4
5	Test Case 5
6	Test Case 6
7	Test Case 7

**Steps: Test Case 4**

Execution Trace Console:

- Boundary Values: Currently covered 79% (63/79) of target checkpoints.
- Boundary Values: Currently covered 87% (69/79) of target checkpoints.
- Boundary Values: Running incremental test generation.
- Boundary Values: Finally covered 87% (69/79) of target checkpoints.
- Boundary Values: Generated 7 test cases.
- DC 2: Running test asset analysis.
- DC 2: Currently covered 35% (21/61) of target checkpoints.
- DC 2: Currently covered 48% (29/61) of target checkpoints.
- DC 2: Currently covered 50% (31/61) of target checkpoints.
- DC 2: Currently covered 72% (44/61) of target checkpoints.
- DC 2: Currently covered 83% (51/61) of target checkpoints.
- DC 2: Currently covered 90% (55/61) of target checkpoints.
- DC 2: Currently covered 100% (61/61) of target checkpoints.
- DC 2: Running incremental test generation.
- DC 2: 100% coverage reached, stopping.
- DC 2: Finally covered 100% (61/61) of target checkpoints.
- DC 2: Generated 7 test cases.

# Génération de cas de test

## Statut de la génération du test montré par Eclipse

- Boundary Value: Currently covered 82% (65/79) of target checkpoints.
- Boundary Value: Currently covered 87% (69/79) of target checkpoints.
- Boundary Value: Running incremental test generation.
- Boundary Value: Finally covered 87% (69/79) of target checkpoints.
- Boundary Value: Generated 7 test cases.
- Requirement Coverage: Running test asset analysis.
- Requirement Coverage: Currently covered 35% (21/61) of target checkpoints.
- Requirement Coverage: Currently covered 38% (23/61) of target checkpoints.
- Requirement Coverage: Currently covered 59% (36/61) of target checkpoints.
- Requirement Coverage: Currently covered 68% (42/61) of target checkpoints.
- Requirement Coverage: Currently covered 83% (51/61) of target checkpoints.
- Requirement Coverage: Currently covered 95% (58/61) of target checkpoints.

### Test Cases: SIPClient >

Search:

#	Name
1	Test Case 1
2	Test Case 2
3	Test Case 3
4	Test Case 4
5	Test Case 5
6	Cancel Timeouted
7	Test Case 7

Liste des cas de test

- DC 2: Currently covered 48% (29/61) of target checkpoints.
- DC 2: Currently covered 50% (31/61) of target checkpoints.
- DC 2: Currently covered 72% (44/61) of target checkpoints.
- DC 2: Currently covered 83% (51/61) of target checkpoints.
- DC 2: Currently covered 90% (55/61) of target checkpoints.
- DC 2: Currently covered 100% (61/61) of target checkpoints.
- DC 2: Running incremental test generation.
- DC 2: 100% coverage reached, stopping.
- DC 2: Finally covered 100% (61/61) of target checkpoints.
- DC 2: Generated 7 test cases.

# Génération de cas de test

Qtronic - Requirement  
File Edit Navigate Pr

Traceability: Requirement Coverage Console

SIPClient

Messages

- Boundary Value: Currently covered 82% (65)
- Boundary Value: Currently covered 87% (69)
- Boundary Value: Running incremental test
- Boundary Value: Finally covered 87% (69/79)

**Statut de la génération de test montré par**

Test Cases: SIPClient >

Search:

#	Name
1	Test Case 1
2	Test Case 2
3	Test Case 3
4	Test Case 4
5	Test Case 5
6	Cancel Timeouted
7	Test Case 7

generated 7 test cases.

coverage: Running test asset

coverage: Currently covered 3

coverage: Currently covered 3

coverage: Currently covered 3

coverage: Currently covered 3

coverage: Currently covered 8

coverage: Currently covered 9

|||

SIPResp -> netIn

SIPResp -> netIn

SIPReq <- netOut

UserInput -> userIn

SIPReq <- netOut

SIPReq <- netOut

**Liste des cas de test**

Traceability Matrix

Testing Goals

	1	2	3	4	5	6
Requirements						
17.1.2.2 Non-INVITE timers						
Resends CANCEL after E timeout						X
Terminates CANCEL cycle after F timeout						X
Resends BYE after E timeout				X		
Terminates BYE cycle after F timeout				X		
Terminating						
Wait for OK in response to BYE			X			
Send OK in response to BYE		X				
17.1.1.2 INVITE timers						
Terminates INVITE cycle after B timeout						
Resends INVITE after A timeout						
Acknowledge established call with ACK		X	X	X		
State Chart						
2-Transitions						
Transitions						
Implicit Consumption						
States						
SIPClient-Init				X	X	X
SIPClient-Calling-initial		X	X	X	X	X
SIPClient-Calling-Wait		X	X	X	X	X
SIPClient-Calling-junction-2				X	X	X
SIPClient-Ready		X	X	X	X	X

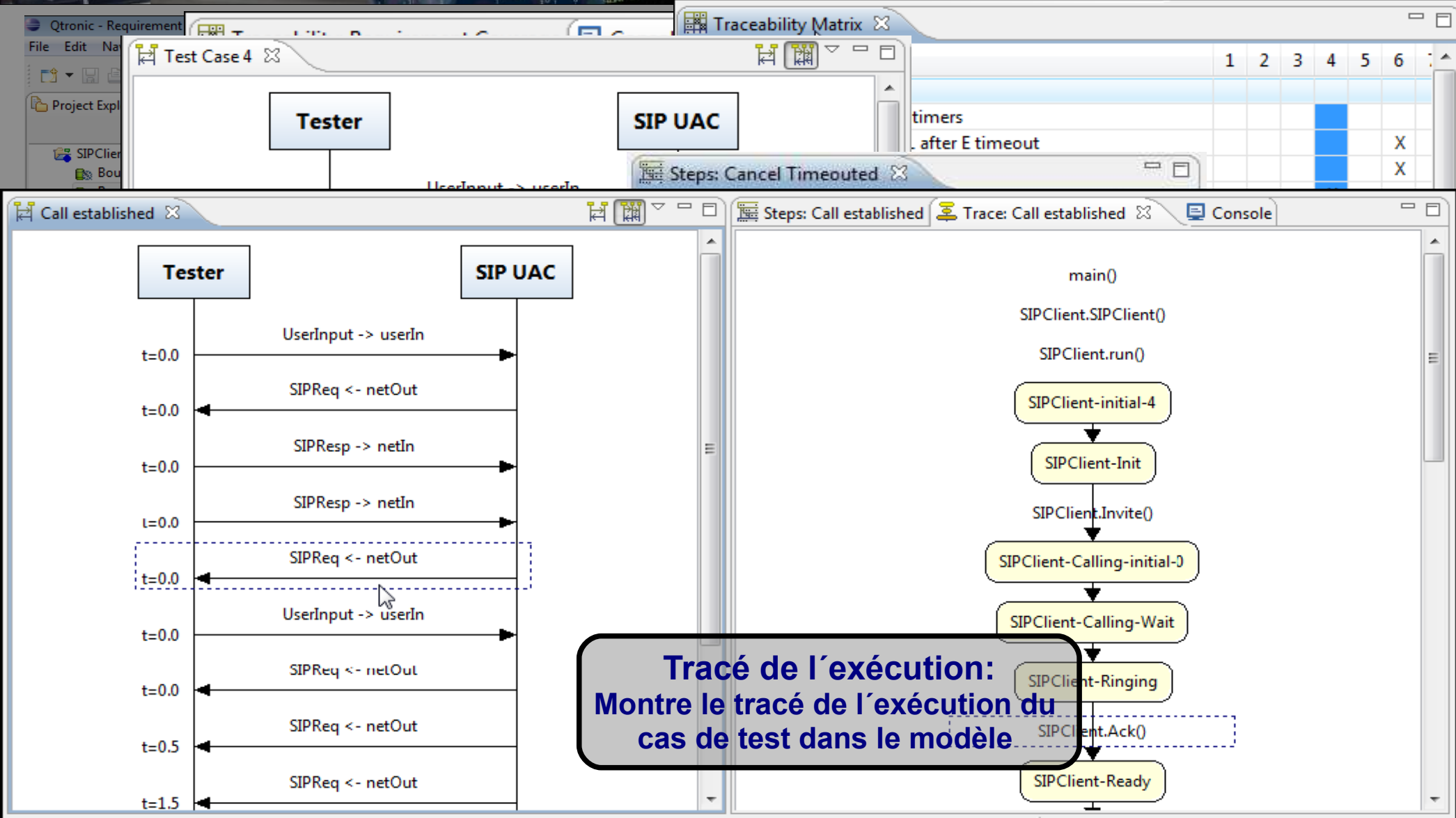
**Tableau de Tracabilité :  
Montre pour chaque cas de test ce que ça couvre**

# Génération de cas de test

The screenshot displays the Qtronic software interface. On the left, a sequence diagram titled "Test Case 4" shows interactions between a "Tester" and "SIP UAC". The diagram includes messages such as "UserInput -> userIn", "SIPReq <- netOut", and "SIPResp -> netIn". A dashed blue box highlights a "SIPReq <- netOut" message at t=0.0. A callout box points to this message with the text "Message graphique: séquence pour un cas de test". Below the diagram, another callout box says "cas de test". On the right, a "Traceability Matrix" window shows a table of test steps. A callout box points to the table with the text "Vue des étapes du test avec des informations plus détaillées sur le message".

Message / Field	Port / Field value	Time
1 UserInput	to userIn	0.0
input1	invite	
input2	sip:127.0.0.1:5061	
2 SIPReq	from netOut	0.0
op	INVITE	
param	sip:127.0.0.1:5061	
3 SIPResp	to netIn	0.0
status	180	
cseq		
4 UserInput	to userIn	0.0
input1	cancel	
input2		
5 SIPReq	from netOut	0.0
op	CANCEL	
param	sip:127.0.0.1:5061	
6 SIPReq	from netOut	0.5
7 SIPReq	from netOut	1.5
8 SIPReq	from netOut	3.5
9 SIPReq	from netOut	7.5
op	CANCEL	
10 TimeOutIndication	from userOut	8.0

# Génération de cas de test



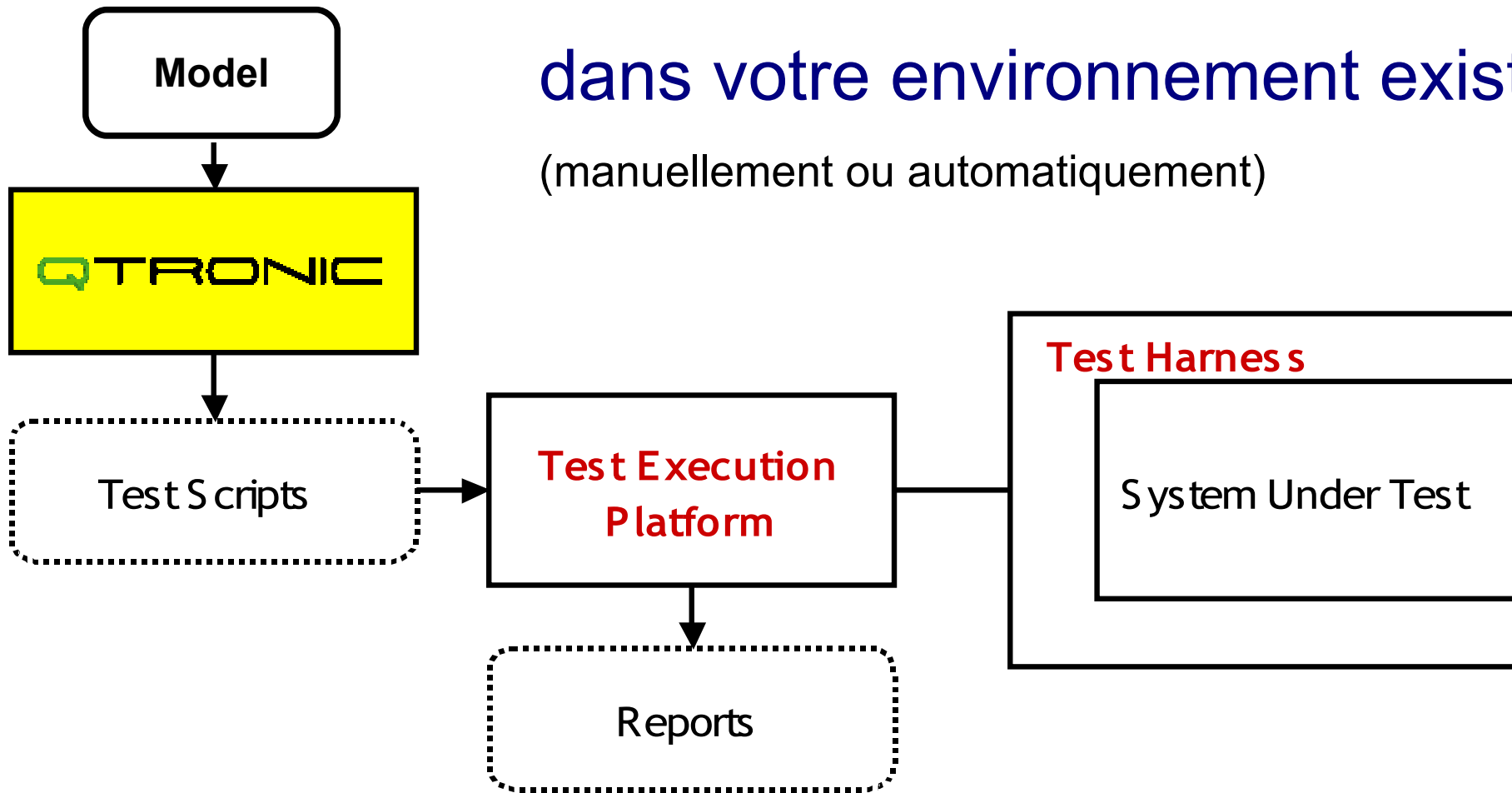
**Tracé de l'exécution:**  
Montre le tracé de l'exécution du cas de test dans le modèle

avec des informations plus détaillées sur le message



# Exécution des cas de test

Exécution des cas de test  
dans votre environnement existant  
(manuellement ou automatiquement)



# Programme académique

## Programme académique

Testwell CTC++

CTC++ Add-on pour Java et C#

Testwell CMT++ /

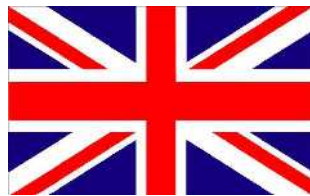
Testwell CMTJava

Conformiq Qtronic



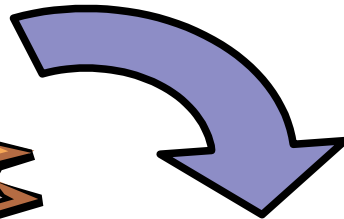
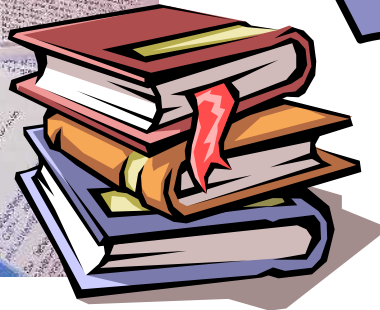
# Bibliothèque de test

## Bibliothèque de test en ligne



→ Vous trouverez des publications portant sur les thèmes de la qualité logiciel mais également test logiciel.

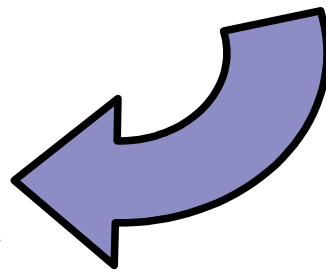
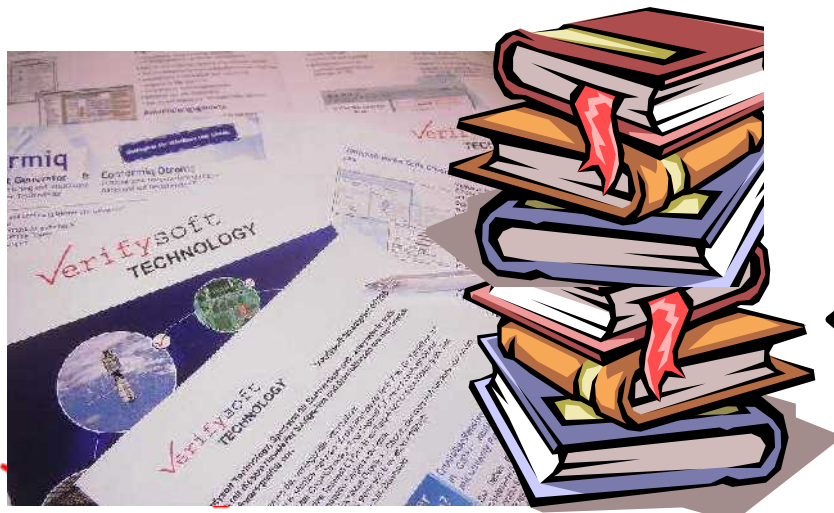
# Bibliothèque de test



Vous pourrez  
télécharger ses  
documents



mais également



y soumettre vos  
propres travaux



# Pour plus d'information



Contacts: Anne-Sophie OBER  
Klaus LAMBERTZ

Tél. France: 03 68 33 58 84

france@verifysoft.com

[www.verifysoft.com](http://www.verifysoft.com)



**Merci !**

