*Δοκιμές λογισμικού βασισμένες σε μοντέλα – Το εργαλείο Model J Unit*

*Εισηγητής:****Παναγιώτης Κατσαρός***

*Κρητικός Απόστολος (249)  
e-mail:* [*akritiko@csd.auth.gr*](mailto:akritiko@csd.auth.gr)

*Επαλήθευση Λογισμικού  
Εαρινό Εξάμηνο, Ακ. Έτος 2008 – 2009*

*Πρόγραμμα Μεταπτυχιακών Σπουδών  
Τμήμα Πληροφορικής, Α.Π.Θ.*

Περιεχόμενα

[1. Εισαγωγή 3](#_Toc234868925)

[1.1. Δοκιμές Λογισμικού (Software Testing) 4](#_Toc234868926)

[1.2. Δοκιμές λογισμικού βασισμένες σε μοντέλα (Model based testing) 5](#_Toc234868927)

[2. Δοκιμές Λογισμικού Βασισμένες σε Μοντέλα 6](#_Toc234868928)

[2.1. Σκοπός 6](#_Toc234868929)

[2.2. Πλεονεκτήματα της μεθόδου (ή «Γιατί να την εφαρμόσω») 7](#_Toc234868930)

[2.3. Μοντέλα και σημειογραφία 7](#_Toc234868931)

[2.3.1. Τι είναι ένα μοντέλο; 7](#_Toc234868932)

[2.3.2. Ποια σημειογραφία χρησιμοποιούμε για να δημιουργήσουμε ένα μοντέλο; 8](#_Toc234868933)

[3. Σχετικά εργαλεία 10](#_Toc234868934)

[3.1. Εργαλεία Δοκιμών Βασισμένων σε Μοντέλα 10](#_Toc234868935)

[3.2. Εργαλεία αυτόματης παραγωγής δοκιμών 11](#_Toc234868936)

[3.2.1. Εμπορικά εργαλεία (commercial tools) 11](#_Toc234868937)

[3.2.2. Εργαλεία προστατευμένα από πνευματικά δικαιώματα (proprietary tools) 13](#_Toc234868938)

[3.2.3. Εργαλεία για ακαδημαϊκή χρήση (academic tools) 13](#_Toc234868939)

[4. Το εργαλείο Model J Unit 14](#_Toc234868940)

[5. Συμπεράσματα, προβληματισμοί 14](#_Toc234868941)

[6. Πιθανό μελλοντικό έργο 14](#_Toc234868942)

[7. Βιβλιογραφία 14](#_Toc234868943)

# Εισαγωγή

***“ Modern Civilization depends on software,   
so it needs to be as reliable as possible.  
But software is produced by humans,   
who are fallible… ”***

***The Economist,  
March 6th, 2008   
(printed version)***

Είναι αδιαμφισβήτητο το γεγονός πως τα τελευταία χρόνια το λογισμικό εξελίσσεται ολοένα και περισσότερο σε έναν βασικό άξονα ενός «μοντέρνου» πολιτισμού. Πλέον, εκτός των έργων λογισμικού που αφορούν τις μεγάλες βιομηχανικές επιχειρήσεις, τον στρατό ή κοιτίδες έρευνας, αναπτύσσεται λογισμικό προσανατολισμένο σε μικρομεσαίες επιχειρήσεις ή ιδιώτες, ενώ οι υπηρεσίες κοινωνικής δικτύωσης, που αποτελούν ουσιαστικά λογισμικό διαδικτύου, έχουν εξελιχθεί σε ισχυρό trend της εποχής.

Όλα τα παραπάνω κάνουν επιτακτική την ανάγκη του σχεδιασμού και υλοποίησης αξιόπιστου λογισμικού. Ωστόσο, το λογισμικό αναπτύσσεται από ανθρώπους που είναι επιρρεπείς σε σφάλματα. Έτσι, αστοχίες, σφάλματα και τρύπες ασφάλειας αποτελούν συχνά ψεγάδια σε ένα λογισμικό.

Όπως πολύ εύστοχα σχολιάζει το περιοδικό economist στο άρθρο του “*Software that makes software better”*, είναι ειρωνικό το γεγονός ότι λύση στα παραπάνω προβλήματα δίνεται με την προσθήκη επιπλέον κώδικα. Εν γένει, το λογισμικό πρέπει να δοκιμάζεται σε διαφορετικά στάδια κατά την ανάπτυξή του. Τα διάφορα διαθέσιμα εργαλεία ελέγχου λογισμικού μπορούν να διευκολύνουν τη συνεργασία των προγραμματιστών, να επιτελέσουν εξονυχιστικό έλεγχο του λογισμικού σε επίπεδο εκτέλεσης ή κώδικα για αστοχίες και σφάλματα ή, σε ένα υψηλότερο επίπεδο, να βοηθήσουν τον διοικητή του έργου να ποσοτικοποιήσει την ποιότητα του λογισμικού, την παραγωγικότητα των προγραμματιστών, την επίδραση που θα έχει στο κόστος παραγωγής η διόρθωση ενός σφάλματος, κ.ο.κ.

Σημειωση:

Η παρούσα εργασία βασίζεται στο περιεχόμενο του συγγράμματος [***Practical Model-Based Testing: A Tools Approach***](http://www.cs.waikato.ac.nz/~marku/mbt)*των Mark Utting και Bruno Legeard* το οποίο στο εξής θα αναφέρεται ως [4].

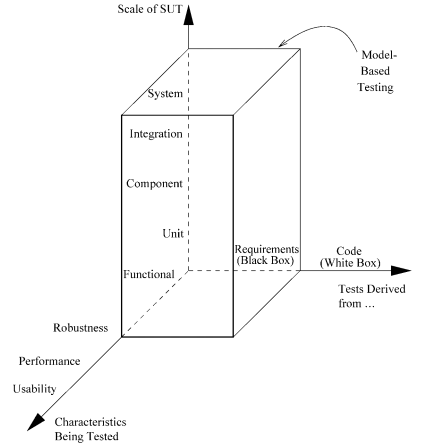
## Δοκιμές Λογισμικού (Software Testing)

Οι δοκιμές λογισμικού στοχεύουν κατά βάσην στην ποσοτικοποίηση της ποιότητας του συστήματος ή της υπηρεσίας υπό έλεγχο (system or service under test – SUT ) πάντοτε σε συμφωνία με το πλαίσιο στο οποίο καλείται να λειτουργήσει αφότου ολοκληρωθεί η υλοποίησή του. Επιπλέον, οι δοκιμές λογισμικού μπορούν να χρησιμοποιηθούν ως μηχανισμός αναγνώρισης και ποσοτικοποίησης του ρίσκου ανάπτυξης του εκάστοτε λογισμικού.

Όπως ήδη αναφέραμε, οι διαδικασία των ελέγχων μπορεί να εκτελεστεί κατά τη διάρκεια εκτέλεσης ενός προγράμματος για εύρεση σφαλμάτων (bugs) ή ακόμη και μετά την υλοποίηση για επαλήθευση των απαιτήσεων του έργου. Το σύνηθες στάδιο στο οποίο εκτελούνται οι δοκιμές λογισμικού παραμένει ακόμη το σημείο αμέσως μετά την περάτωση της φάσης της υλοποίησης αν και η τεχνική του *σχεδιασμού λογισμικού προσανατολισμένου σε δοκιμές (test-driven design)* φαίνεται ότιαρχίζει να υιοθετείται από όλο και περισσότερους.

Πριν επικεντρωθούμε περισσότερο στην κατηγορία δοκιμών που θα μας απασχολήσει στη συνέχεια της παρούσας εργασίας, θεωρούμε δόκιμο να παραθέσουμε τις πιο γνωστές μεθόδους διεξαγωγής δοκιμών λογισμικού:

* Δοκιμές μαύρου κουτιού (Black box testing)
* Δοκιμές λευκού κουτιού (White box testing)
* Δοκιμές γκρίζου κουτιού (Grey box testing)
* Δοκιμές αποδοχής (Acceptance testing)
* Δοκιμές παλινδρόμησης (Regression testing)
* Δοκιμές μη λειτουργικών απαιτήσεων (Non functional software testing)



Εικόνα 1-1. Διαφορετικά ήδη δοκιμών λογισμικού

## Δοκιμές λογισμικού βασισμένες σε μοντέλα (Model based testing)

Οι δοκιμές λογισμικού βασισμένες σε μοντέλα αναφέρονται συχνά τα τελευταία χρόνια. Ωστόσο, η αφαίρεση του όρου έχει οδηγήσει στο να χρησιμοποιείται για πλειάδα τεχνικών παραγωγής δοκιμών λογισμικού. Στο [4] γίνεται αναφορά στις τέσσερις σημαντικότερες προσεγγίσεις που εμπίπτουν στην κατηγορία των δοκιμών λογισμικού βασισμένων σε μοντέλα. Αυτές είναι:

1. Παραγωγή δοκιμών δεδομένων εισόδου από μοντέλα πεδίου (domain models)
2. Παραγωγή δοκιμών από μοντέλα περιβάλλοντος (environment models)
3. Παραγωγή δοκιμών κάνοντας χρήση «χρησμών» (oracles) από μοντέλα συμπεριφοράς (behavior models)
4. Παραγωγή κώδικα δοκιμών κάνοντας χρήση αφηρημένων δοκιμών (abstract tests)

# Δοκιμές Λογισμικού Βασισμένες σε Μοντέλα

Στα πλαίσια της παρούσας εργασίας θα ασχοληθούμε με την Παραγωγή δοκιμών κάνοντας χρήση «χρησμών» (oracles) από μοντέλα συμπεριφοράς (behavior models). Σύμφωνα με το [4] αποτελεί την εκδοχή που πλησιάζει περισσότερο την πραγματική έννοια των δοκιμών λογισμικού βασισμένων σε μοντέλα. Δεδομένου λοιπόν ότι στη συνέχεια του εγγράφου θα παρουσιάσουμε το εργαλείο Model J Unit, πνευματικό παιδί των συγγραφέων του προαναφερθέντος συγγράμματος ([4]), θεωρήσαμε δόκιμο να παραθέσουμε το αντίστοιχο θεωρητικό υπόβαθρο από την σκοπιά της ίδιας ομάδας ερευνητών.

## Σκοπός

Σκοπός μας είναι η παραγωγή εκτελέσιμων δοκιμών που εμπεριέχουν πληροφορίες «χρησμών» (oracles), όπως για παράδειγμα την αναμενόμενη τιμή εξόδου του συστήματος υπό έλεγχο ή κάποιους αυτοματοποιημένους ελέγχους στις πραγματικές τιμές εξόδου για να δούμε αν είναι σωστές.

Για να μπορέσουμε να παράγουμε επιτυχώς, δοκιμές με χρήση «χρησμών» θα πρέπει η γεννήτρια δοκιμών να γνωρίζει την αναμενόμενη συμπεριφορά του συστήματος υπό έλεγχο ώστε να μπορεί να προβλέπει ή να ελέγχει της τιμές εξόδου του συστήματος υπό έλεγχο. Με άλλα λόγια, στην συγκεκριμένη εκδοχή δοκιμών λογισμικού βασισμένων σε μοντέλα, το μοντέλο πρέπει να περιγράφει την αναμενόμενη συμπεριφορά το συστήματος υπό έλεγχο, ως σχέσεις μεταξύ των μεταβλητών εισόδου και εξόδου.

Αφήνοντας ασχολίαστες τις υπόλοιπες τρεις εκδοχές των δοκιμών λογισμικού βασισμένων σε μοντέλα, ο αναγνώστης μπορεί να αναρωτηθεί γιατί και σε τι βαθμό διαφέρει η παρούσα προσέγγιση από τις υπόλοιπες τρεις που δεν περιλαμβάνονται στην παρούσα εργασία.

Παραγωγή δοκιμών κάνοντας χρήση «χρησμών» (oracles) από μοντέλα συμπεριφοράς (behavior models) είναι η μοναδική προσέγγιση από τις προαναφερθείσες τέσσερις που αντιμετωπίζει το ζήτημα του ελέγχου του σχεδιασμού του λογισμικού στην ολότητά του από την επιλογή τιμών εισόδου και την παραγωγή ακολουθιών από κλήσεις διαδικασιών εσωτερικά του μοντέλου, μέχρι την παραγωγή εκτελέσιμων περιπτώσεων χρήσεων που εξετάζουν τμήματα του λογισμικού στα οποία υπεισέρχεται υποκειμενικότητα.

## Πλεονεκτήματα της μεθόδου (ή «Γιατί να την εφαρμόσω»)

Ακόμη και αν δεχθούμε τον ισχυρισμό ότι η μέθοδος που περιγράφουμε είναι πολύ πιο εξεζητημένη από τις υπόλοιπες τρεις αξίζει στην πραγματικότητα να δαπανήσει κάποιος χρόνο και πνευματικό κεφάλαιο για να την εφαρμόσει;

Η παραπάνω μέθοδος μπορεί να μας προσφέρει:

* Αυτοματοποίηση της διαδικασίας σχεδιασμού, δεδομένου ενός κατάλληλου μοντέλου και παραγωγή ακολουθιών δοκιμών που μπορούν να μετασχηματιστούν σε εκτελέσιμα κομμάτια κώδικα δοκιμών.
* Αυτοματοποίηση, ουσιαστικά, της διαδικασίας διεξαγωγής δοκιμών μαύρου κουτιού. Η βασική διαφορά με την κλασική προσέγγιση έγκειται στην παραγωγή δοκιμών βασισμένων σε ένα μοντέλο προσομοίωσης του συστήματός αντί της συγγραφής των τεστ «με το χέρι». Επομένως, κάνοντας χρήση του μοντέλου, καλύπτουμε σίγουρα ένα μέρος των απαιτήσεων υλοποίησης. Στη συνέχεια, με χρήση κατάλληλων εργαλείων παράγουμε δοκιμές από το μοντέλο.

## Μοντέλα και σημειογραφία

Από την μέχρι τώρα ανάλυσή μας γίνεται φανερό ότι η μέθοδος που περιγράφουμε βασίζεται στην δημιουργία ενός μοντέλου συμπεριφοράς του συστήματος που καλούμαστε να επαληθεύσουμε.

### Τι είναι ένα μοντέλο;

Οι συγγραφείς του [4] βασισμένοι στο American Heritage Dictionary παραθέτουν σε ελεύθερη μετάφραση τους ακόλουθους δύο ορισμούς για τον όρο μοντέλο:

* Ένα μικρό αντικείμενο, συνήθως σχεδιασμένο σε κλίμακα, που αναπαριστά λεπτομερώς ένα άλλο, συνήθως μεγαλύτερο αντικείμενο.
* Μια σχηματική αναπαράσταση ενός συστήματος, μιας θεωρίας, ή ενός φαινομένου που είναι άξιο αναφοράς χάρη στις γνωστές αλλά και τις εν δυνάμει συναγόμενες ιδιότητές του και μπορεί να χρησιμοποιηθεί για περαιτέρω μελέτη των χαρακτηριστικών του.

Οι ορισμοί αυτοί μας επιτρέπουν να επισημάνουμε τα χαρακτηριστικά εκείνα που είναι απαραίτητα για την ομαλή διεξαγωγή των δοκιμών της μεθόδου που εξετάζουμε. Πιο συγκεκριμένα:

* Ένα μοντέλο πρέπει να είναι μικρό σε σχέση με το μέγεθος του συστήματος που θέλουμε να επαληθεύσουμε έτσι ώστε να μην είναι πολύ κοστοβόρα στην παραγωγή τους.
* Ένα μοντέλο πρέπει να είναι αρκετά λεπτομερές ώστε να περιγράφει επαρκώς τα χαρακτηριστικά του συστήματος που θέλουμε να επαληθεύσουμε.

Όπως είναι προφανές τα δύο αυτά χαρακτηριστικά, μικρό μέγεθος και ικανοποιητική λεπτομέρεια είναι, εν γένει, αντικρουόμενα. Έτσι θα πρέπει αυτός που καλείται να κατασκευάσει το μοντέλο να έχει εμπειρία στην μηχανική λογισμικού. Από την άλλη μεριά βέβαια, το γεγονός ότι ο έλεγχος του λογισμικού θα πραγματοποιηθεί με χρήση ενός μοντέλου, διευκολύνει τα πράγματα, αφού τα μοντέλα προσφέρουν τη δυνατότητα περιγραφής χαρακτηριστικών με έναν σχετικά «φυσικό» τρόπο.

### Ποια σημειογραφία χρησιμοποιούμε για να δημιουργήσουμε ένα μοντέλο;

Πλειάδα σημειογραφιών για δημιουργία μοντέλων υπάρχουν στην βιβλιογραφία. Η εκτενής αναφορά σε αυτές σαφώς ξεφεύγει από τα πλαίσια της παρούσας εργασίας επομένως επιλέξαμε να περιοριστούμε στην παρουσίαση ορισμένων γενικών κατηγοριών αυτών των σημειογραφιών:

**Προ / Μετά (ή βασισμένες σε καταστάσεις) σημειογραφίες:**  
(Pre / Post (or state – based) notations)

Μοντελοποιούν το σύστημα σαν μια συλλογή από μεταβλητές, οι οποίες αναπαριστούν ένα στιγμιότυπο της εσωτερικής κατάστασης του συστήματος συμπεριλαμβανομένων κάποιων λειτουργιών που αλλάζουν αυτές τις μεταβλητές.

**Σημειογραφίες βασισμένες στις μεταβάσεις:**(Transition-based notations)

Ασχολούνται με την περιγραφή των μεταβάσεων από μία κατάσταση του συστήματος σε μία άλλη. Ουσιαστικά πρόκειται για μια αναπαράσταση σε μορφή γράφου, π.χ. Μηχανή Πεπερασμένων Καταστάσεων (Finite State Machine). Ως κόμβοι, απεικονίζονται τα βασικά τμήματα του λογισμικού και ως ακμές οι λειτουργίες (ή ενέργειες ) του συστήματος.

**Σημειογραφίες βασισμένες σε ιστορικά:**(History-based notations)

Το σύστημα μοντελοποιείται σαν ένα σύνολο από επιτρεπτά ίχνη της συμπεριφοράς του κατά την εξέλιξή του στο χρόνο. Αξιοσημείωτο είναι το γεγονός ότι διαφορετικές εκφάνσεις του χρόνου είναι επιτρεπτές (διακριτός ή συνεχής, γραμμικός ή διακλαδισμένος, κλπ.)

**Διαδικαστικές Σημειογραφίες:**(Functional notations)

Περιγράφουν το σύστημα ως μια συλλογή από μαθηματικές συναρτήσεις.

**Λειτουργικές Σημειογραφίες:**(Operational notations)

Χρησιμοποιούνται για να περιγράψουν ένα σύνολο από εκτελέσιμες διαδικασίες οι οποίες συνήθως, εκτελούνται παράλληλα. Τέτοιου είδους σημειογραφίες είναι πολύ καλές για επαλήθευση κατανεμημένων ή παράλληλων συστημάτων.

**Στατιστικές Σημειογραφίες:**(Statistical notations)

Εξετάζουν το σύστημα σαν ένα πιθανοκρατικό μοντέλο των γεγονότων και των μεταβλητών εισόδου του. Συνήθως χρησιμοποιούνται περισσότερο σε περιβάλλοντα μοντέλων και όχι σε συστήματα και υπηρεσίες υπό έλεγχο.

**Σημειογραφίες Ροής Δεδομένων:**(Data Flow notations)

Επικεντρώνονται στη ροή των δεδομένων εντός του συστήματος και όχι στη ροή ελέγχου του.

# Σχετικά εργαλεία

## Εργαλεία Δοκιμών Βασισμένων σε Μοντέλα

* [All4Tec - MaTeLo](http://www.all4tec.net/index.php?option=com_content&task=blogcategory&id=5&Itemid=75&lang=en): is a statistical model-based testing tool. From usage models, MaTeLo generates efficient tests sequences in TTCN-3, National Instruments TestStand (c) and XML.
* [AsmL Test Tool](http://research.microsoft.com/fse/asml/) can generate tests directly from an AsmL model
* [ATD-Automated Test Designer](http://www.atyoursideconsulting.com/) is a model-based testing tool that generates automatically test cases, test data and test automation scripts from requirement specifications.
* [ATGT ASM Tests Generation Tool](http://cs.unibg.it/gargantini/projects/atgt/) generates tests from ASM models according to structural, fault, and combinatorial criteria
* [AutoFocus](http://autofocus.informatik.tu-muenchen.de/) (in German) is a graphical tool for developing and modeling distributed systems with integrated testing facilities
* [Classification-Tree Editor for Embedded Systems (CTE/ES)](http://www.razorcat.com/frame_main/produkte/unterpunkte/cte_e.html) is a test-design tool based on the [Classification-Tree Method for Embedded Systems](http://ctmemb.googlepages.com/)
* [Conformiq Qtronic](http://www.conformiq.com/qtronic.php) is a system requirements model driven automated test design tool which can generated executable test cases in C, Java, Perl, TCL, TTCN-3
* [EmbeddedTester](http://www.osc-es.de/index.php?lang=2&idcat=17) is an automatic test generator and test execution tool for the Matlab/Simulink/TargetLink Environment
* [EmbeddedValidator](http://www.osc-es.de/index.php?lang=2&idcat=18) is a Matlab/Simulink/Stateflow/Targetlink Formal Verification Environment
* [errfix](http://code.google.com/p/errfix/) is an open-source model based testing library written in Ruby
* [GATeL](http://www-list.cea.fr/labos/gb/LSL/test/gatel/index.html) is a tool that automatically generates test sequences from SCADE/Lustre models, according to a user defined test objective
* [HOL-TestGen](http://www.brucker.ch/projects/hol-testgen/) is a test case generation tool based on the interactive theorem prover [Isabelle](http://en.wikipedia.org/wiki/Isabelle_theorem_prover).
* [KeY](http://www.key-project.org/) is a software verification tool for [Java](http://en.wikipedia.org/wiki/Java_programming_language) that can generate model-based [white box](http://en.wikipedia.org/wiki/White_box_testing) tests.
* [Lurette](http://www-verimag.imag.fr/~synchron/index.php?page=lurette/lurette) is an automated testing tool of reactive programs written in Lustre
* [mbt.tigris.org](http://mbt.tigris.org/) is an open-source model-based testing tool that uses [graphml](http://graphml.graphdrawing.org/" \o "http://graphml.graphdrawing.org/) models to generate test sequence files
* [ModelJUnit](http://www.cs.waikato.ac.nz/~marku/mbt/modeljunit) is an open-source model-based testing tool that uses models written as annotated Java classes.
* [NModel](http://www.codeplex.com/NModel/) is an open-source model-based testing and analysis framework for model programs written in C#.
* [Rhapsody TestConductor](http://www.osc-es.de/index.php?lang=2&idcat=22) is a UML (Unified Modeling Language) compliant scenario-based test and validation suite for real-time embedded applications
* [Rhapsody ATG](http://www.osc-es.de/index.php?lang=2&idcat=23) automatic test sequences generator tool from Rhapsody UML models
* [Reactis Tester](http://www.reactive-systems.com/) is another model-based testing tool that focuses on control systems
* [Simulink Tester](http://www.t-vec.com/solutions/simulink.php) is a tool that translates Simulink and Stateflow models for model analysis and test generation by the [T-VEC Vector Generation System](http://www.t-vec.com/solutions/tvec.php)
* [Smartesting Test Designer](http://www.smartesting.com/) is a model-based testing tool based on UML for enterprise business information systems and eTransaction applications
* [Spec Explorer](http://research.microsoft.com/specexplorer/) is a model exploration and test suite generation tool from Microsoft that uses Spec#, C#, or AsmL to describe models
* [Statemate ATG](http://www.osc-es.de/index.php?lang=2&idcat=21) automatic test vectors generator tool from Statemate models
* [TestOptimal](http://TestOptimal.com/) is a model-based testing tool with 5 different test sequencers to test web and java applications, it can be easily integrated with IDEs, JUnit and other testing frameworks/tools.
* [TGV](http://www-verimag.imag.fr/~async/TGV/index.shtml.en) is a tool for the generation of conformance test suites for protocols
* [Time Partition Testing](http://www.piketec.com/products/tpt.php?lang=en) ([TPT](http://en.wikipedia.org/wiki/TPT_(Software))) is a model-based testing tool for control systems software. [TPT](http://en.wikipedia.org/wiki/TPT_(Software)) supports automatic testing and test result assessment in almost any test execution environment e.g. MATLAB/Simulink, TargetLink, C-Code...
* [T-VEC Tabular Modeler](http://www.t-vec.com/solutions/ttm.php) is a tool that supports requirement modeling and test generation through the [T-VEC Vector Generation System](http://www.t-vec.com/solutions/tvec.php)
* [TorX](http://fmt.cs.utwente.nl/tools/torx/introduction.html) is also a prototype testing tool for conformance testing of reactive software
* [CTESK](http://www.unitesk.com/content/category/7/14/33/), [JavaTESK](http://www.unitesk.com/content/category/7/28/74/" \o "http://www.unitesk.com/content/category/7/28/74/) are toolkits for testing applications developed in [C](http://en.wikipedia.org/wiki/C_(programming_language)) and [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) respectively. The tools are based on [UniTESK](http://en.wikipedia.org/w/index.php?title=UniTESK&action=edit&redlink=1" \o "UniTESK (page does not exist))[[1]](http://www.unitesk.com) technology.
* [eXpecco](http://www.expecco.de) from eXept Software AG is a model-based testing tool, where executable activity diagrams are use to model test suites for [black-box testing](http://en.wikipedia.org/wiki/Black-box_testing)

## Εργαλεία αυτόματης παραγωγής δοκιμών

### Εμπορικά εργαλεία (commercial tools)

**TVEC**

T-VEC Technologies has a Test Vector Generation System [29] which accepts models of the

requirements and behavior of a system in a proprietary language called T-VEC Linear Form. The

product also comes with a graphical environment for creating SCR models and automatically

translating them into the Linear Form. Translators exist for two other specification languages namely

SCR [9] and MATRIXx [27].

The test generator generates test cases based on domain testing theory. It generates test cases by

analyzing the logical expressions in the specification model, and creating test cases that reach extreme

values in the decision path (a kind of branch coverage of the specification). The test generation

directives appear to be implicit in the algorithm used to generate test cases.

The test cases produced by T-VEC include expected outputs.

The T-VEC system also has an interface for translating the abstract test cases into executable test

programs (test driver generation).

The tool seems to have been used mainly in the aerospace industry. It does not appear to have

any support for UML or other widely used modeling languages.

**CONFORMIQ TEST GENERATOR**

Conformiq Software Ltd. has announced a new product which is due for release in autumn 2002.

The literature on their website [2] indicates that their tool will be a true model-based test generator

according to the definition of this paper. The tool accepts UML state diagrams as the model of the

system under test, together with support for real time properties. The main product of the

generator’s analysis is a behavior simulator which may be used in three different ways:

Batch mode: To generate test cases using the TTCN notation. These test cases may be run

against the unit under test at a later time. The test cases contain expected results and verdicts.

Interactive mode: Running the test generator in synchronized mode with the unit under test. The

behavior of the system is verified online.

Trace mode: A trace of system activity is run against the test generator and its output is verified

as conforming to the specification model (or not).

The literature on the website is not specific about the coverage directives used by the tool, but it

indicates that the tool includes a measure of the coverage of the system achieved by the test.

**REACTIS**

Reactis [21] is a tool produced by Reactive Systems Inc – a company whose focus is on tools for

the development of embedded systems. It accepts models in the SimuLink and StateFlow modelling

language [28].

4

The model is compiled and a model simulator is produced which can be used for manual or

random test generation. Reactis also has an automatic test suite generation capability and the

coverage directives are related to syntactic and structural coverage of the model (branches, states,

transitions, conditions etc.)

The test cases contain expected results, but there does not appear to be any facility for translating

and executing the test cases in a test harness.

**TAU TTCN SUITE**

Amongst the Telelogic products is a suite of tools for the creation, simulation, and manipulation

of SDL models. The Telelogic Tau TTCN Suite [24] webpage describes an ability to generate TTCN

test cases from an SDL model. No details are given concerning the use of testing directives, but

TTCN test cases certainly included expected results derived from the behavior specified in the SDL

model.

**TESTMASTER**

The Teradyne Corporation produced a model based testing tool called Test Master (see [26]).

This tool had a proprietary graphical language for specifying the system under test. The tool

produced test suites by exhaustive traversal of the finite state machine used in the specification. Test

Master had no special features for translating abstract tests to executable scripts.

The tool is not currently supported or sold.

**UNITESK**

The UniTesK (Unified Testing and Specification Toolkit) produced by ISPRAS (Institute for

System Programming of the Russian Academy of Sciences) is on the border between a commercial

and an academic tool (see [12]).

The model is specified in either J@VA or C@++, which are specification languages designed for

Java and C++ code respectively. The specifications are in the form of pre- and post-conditions, and

are coded as comments in the classes and methods to be tested.

As in the T-VEC tool, the test cases are generated by applying branch coverage of the

specification of the post-condition, and this test directive seems to be implicit in the test generation

process.

Since UniTesK is close to the source code, the test drivers are created as part of the test

generation process, and not in a separate abstract to concrete translation phase.

The tool has been used by NorTel in testing parts of the kernel of a real time operating system.

### Εργαλεία προστατευμένα από πνευματικά δικαιώματα (proprietary tools)

**PROPRIETARY TOOLS**

**GOTCHA-TCBEANS**

GOTCHA-TCBeans is one of the ancestors of the AGEDIS test generation and execution tool

suite. The GOTCHA-TCBeans tools are described in two papers [4][5].

5

The modeling language used by GOTCHA is an extension of Murphi [3]. The Murphi language

is extended by the addition of test generation directives which allow for the specification of arbitrary

projections of the state space to be used as coverage criteria (see [5]).

The test cases include expected outputs, and the test translation framework is provided by

TCBeans.

The tools have been used within the IBM Corporation and the AGEDIS consortium, but are not

publicly available.

**ASML**

AsmL is the Abstract State Machine Language [15]. It is an executable specification language

based on the theory of Abstract State Machines. The current version, AsmL 2 (AsmL for Microsoft

.NET), is embedded into Microsoft Word and Microsoft Visual Studio.NET. It uses XML and Word

for literate specifications.

There is a test generation tool which works with this specification language, and Microsoft

researchers have reported on its characteristics in [8]. Their test generation algorithm appears to be

total transition coverage of the derived finite state machine. The tool is not commercially available.

### Εργαλεία για ακαδημαϊκή χρήση (academic tools)

**ACADEMIC TOOLS**

**SPECTEST**

SpecTest [7] is an automatic test case generator from George Mason University. It accepts

models written in SCR or UML, and generates test cases based on a choice of two coverage criteria,

with a further two planned for implementation.

The tools are not yet available for downloading.

**MULSAW**

The MulSaw project [16] at MIT incorporates two tools for the generation of test cases for Java

programs, one tool (TestEra) accepts input in the Alloy modelling language, and the other (KORAT)

accepts input in the Java Modelling Language (JML). The JML is similar to the language used by

UniTesK, in that it places pre-conditions and post-conditions in a Java-like syntax as comments at

the head of a Java method.

The KORAT test cases are generated to cover all instances of the method preconditions, and

provide expected results based on the post-conditions.

The tools are not available for experimentation.

**TOSTER**

TOSTER (**T**he **O**bject-oriented **S**oftware **T**esting **E**nvi**R**onment) [33] is a test generation and

execution system produced by the Warsaw University of Technology. It incorporates technology for

mapping the information in UML diagrams to the source code of an application. It also generates

and runs test cases based on expected results derived from the UML state diagrams. There appear to

be two test generation algorithms, but no explicit testing directives.

6

**TGV/CADP**

TGV (Test Generation with Verification) [11] is a test case generator developed at the IRISA and

VERIMAG laboratories. It is an ancestor of the AGEDIS test generation tools.

It accepts input in the form of a LOTOS, SDL, or IF specification model. The output of the test

generator is in the TTCN format, and contains the expected results of the test. The test generation

directives may be in the form of FSM models of the test purposes. The tool has been used

extensively in experimental and industrial situations, mainly in the telecommunications industry.

The test generator is incorporated in the CADP package and may be downloaded from the

website at [32].

**TORX/CADP**

The TorX is an architecture for test generation and execution from the University of Twente.

Within this architecture there is a test generator which accepts test purposes in a similar format to

TGV. The modeling languages supported by TorX are LOTOS, PROMELA, and SDL.

The TorX test generator is also incorporated in the CADP package [32].

# Το εργαλείο Model J Unit

# Συμπεράσματα, προβληματισμοί

# Πιθανό μελλοντικό έργο

# Βιβλιογραφία

1. ["Software that makes Software better"](http://www.economist.com/science/tq/displaystory.cfm?story_id=10789417) - economist.com [online]
2. [Future of Software Testing](http://www.applabs.com/internal/app_whitepaper_future_software_testing_1v001.pdf) [online]
3. Eckard Bringmann, Andreas Krämer; [Model-based Testing of Automotive Systems](http://www.piketec.com/downloads/papers/Kraemer2008-Model_based_testing_of_automotive_systems.pdf) In: ICST, pp.485-493, 2008 International Conference on Software Testing, Verification, and Validation, 2008.

1. *[Practical Model-Based Testing: A Tools Approach](http://www.cs.waikato.ac.nz/~marku/mbt" \o "http://www.cs.waikato.ac.nz/~marku/mbt)*, Mark Utting and Bruno Legeard, [ISBN 978-0-12-372501-1](http://en.wikipedia.org/wiki/Special:BookSources/9780123725011), Morgan-Kaufmann 2007.
2. [*Model-Based Software Testing and Analysis with C#*](http://www.cambridge.org/us/catalogue/catalogue.asp?isbn=9780521687614), Jonathan Jacky, Margus Veanes, Colin Campbell, and Wolfram Schulte, [ISBN 978-0-521-68761-4](http://en.wikipedia.org/wiki/Special:BookSources/9780521687614), Cambridge University Press 2008.
3. [*Model-Based Testing of Reactive Systems*](http://www.springer.com/west/home/computer/programming?SGWID=4-40007-22-52081580-detailsPage=ppmmedia%7CaboutThisBook) Advanced Lecture Series, LNCS 3472, Springer-Verlag, 2005.
4. Hong Zhu et al. (2008). [*AST '08: Proceedings of the 3rd International Workshop on Automation of Software Test*](http://portal.acm.org/citation.cfm?id=1370042). ACM Press. [ISBN 978-1-60558-030-2](http://en.wikipedia.org/wiki/Special:BookSources/9781605580302).
5. <http://en.wikipedia.org/wiki/Software_testing> [online]
6. <http://en.wikipedia.org/wiki/Model-based_testing> [online]
7. <http://www.agedis.de/documents/ModelBasedTestGenerationTools_cs.pdf> [ebook]
8. <http://en.wikipedia.org/wiki/Model-based_testing_tools> [online]