Πανεπιστήμιο Θεσσαλίας

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων

Διδακτική των Λογιστικών Φύλλων (Spreadsheets)

Προσεγγίσεις, προβλήματα, προτάσεις

Παναγιώτης Μελίδης

ΑΜ: 1700001

Επιβλέποντες: ΗΛΙΑΣ ΧΟΥΣΤΗΣ,ΠΑΝΑΓΙΩΤΗΣ ΠΟΛΙΤΗΣ

ΓΙΑ ΤΟΝ ΛΕΩΝΙΔΑ ΚΑΙ ΤΗΝ ΑΓΓΕΛΙΚΗ

ΒΟΛΟΣ, 2007

Περιεχόμενα

1. Εισαγωγή

Δεδομένου ότι τα λογιστικά φύλλα είναι εξαιρετικά χρήσιμα και αναφαίρετο στοιχείο των καθημερινών συναλλαγών πολλών εταιριών και οργανισμών, διδάσκονται στα σχολεία, οφείλουμε να εξετάσουμε ποιά είναι η ορθή διδακτική προσέγγιση. Συγκεκριμένα, γιατί γίνονται τόσα λάθη σε ένα τόσο απλό πρόγραμμα? Ποιά είναι η σχέση των λογιστικών φύλλων με τις παραδοσιακές γλώσσες προγραμματισμού? Πως μπορούμε να κάνουμε ορθό έλεγχο λαθών, έτσι ώστε να καθοδηγήσουμε τους μαθητές στην διόρθωση και απαλοιφή αυτών των λαθών? Πόσοι τρόποι υπάρχουν για να γίνει αυτός ο έλεγχος? Και πόσοι τρόποι υπάρχουν αντίστοιχα για να γίνει η διόρθωση τους? Πως μοντελοποιείται ένα λογιστικό φύλλο? Μήπως μια προσέγγιση, θεωρώντας το λογιστικό φύλλο κανονική γλώσσα προγραμματισμού να βοηθούσε?

Το ιερό δισκοπότηρο στην έρευνα περί λογιστικών φύλλων είναι αυτός ακριβώς ο έλεγχος και η μετέπειτα διόρθωση των λαθών. Αυτό ακριβώς μας απασχολεί και στην παρούσα εργασία. Ας πάρουμε τα πράγματα όμως από την αρχή.

1.1 Σύντομη ιστορική αναδρομή

Ένα λογιστικό φύλλο (spreadsheet) μπορεί να αντιληφθεί σαν ένας μεγάλος πίνακας μέσα στον οποίο οι στήλες υποδεικνύονται χαρακτηριστικά από γράμματα και οι σειρές από αριθμούς. Η τομή μιας σειράς και μιας στήλης ορίζει το κελί: ένα κελί μπορεί να περιέχει έναν αριθμό, μία ετικέτα, ή έναν τύπο που το συσχετίζει με άλλα κελιά στο λογιστικό φύλλο. Η δυνατότητα να συσχετίζονται τα κελιά με τύπους είναι αυτή ακριβώς που παρέχει στα λογιστικά φύλλα την τεράστια δύναμή τους. Εάν το μοντέλο ενός συγκεκριμένου λογιστικού φύλλου είναι κατασκευασμένο και βασισμένο σε τύπους, η αλλαγή σε έναν ή δύο αριθμούς απεικονίζεται αμέσως σε όλο το λογιστικό φύλλο.

Λογιστικά φύλλα σε μορφή χαρτιού υπάρχουν εδώ και αιώνες, κυρίως για διατήρηση αρχείου και λογιστικές χρήσεις. Το πρώτο ηλεκτρονικό λογιστικό φύλλο ήταν το VisiCalc εμφανίστηκε το 1979 και δημιουργοί του ήταν ο Dan Bricklin (ο οποίος σκέφτηκε το concept) και ο Bob Frankston (ο οποίος το προγραμμάτισε) για την πλατφόρμα Apple II. Γεννήθηκε σαν ιδέα και αναπτύχθηκε σαν εργαλείο για να κάνει επαναληπτικούς υπολογισμούς χρήσιμους για τις σπουδές του Bricklin στο Harvard Business School. Δημιούργησαν μαζί την εταιρία, Software Arts, για να εμπορευτούν το προϊόν τους. Το 1981, ο Bricklin παρέλαβε το βραβείο Grace Murray Hopper από την ACM (Association for Computing Machinery) για την δημιουργία του VisiCalc. Έχει ειπωθεί ότι το VisiCalc ήταν η εφαρμογή που ώθησε στον να πωληθούν πάνω από ένα εκατομμύριο Apple II υπολογιστές. Χαρακτηριστικά ο Houghton σημειώνει:

Η ανακάλυψη του λογιστικού φύλλου έκανε τους προσωπικούς υπολογιστές να έχουν πραγματική αξία στην αγορά και καθιέρωσε την βιομηχανία ηλεκτρονικών υπολογιστών. Χωρίς την ανακάλυψη αυτής της κατηγορίας εφαρμογών, δηλαδή των λογιστικών φύλλων, ο αντίκτυπος των προσωπικών υπολογιστών στην καθημερινότητα μπορεί να καθυστερούσε για αρκετά χρόνια.

Οι Bricklin και Frankston πούλησαν τα δικαιώματα του VisiCalc στην Lotus Development Corporation, η οποία ανέπτυξε το Lotus 1-2-3, μια εφαρμογή με τεράστια δυναμική και απήχηση η οποία χρησιμοποιήθηκε για τον νέο τότε IBM PC (1982). Η εφαρμογή αυτή εισήγαγε ένα στοιχειώδες γραφικό περιβάλλον και βασική

υποστήριξη βάσεων δεδομένων, και ήταν η κυρίαρχη εφαρμογή της αγοράς καθόλη την δεκαετία του 1980. Έπειτα ήρθε το Microsoft Excel, το οποίο αρχικά είχε αναπτυχθεί για τον Apple Macintosh, αλλά ήταν επίσης και η πρώτη πραγματική εφαρμογή για τα Microsoft Windows. Για την ακρίβεια, οι πρώτες εκδόσεις του Excel τρέχαν στο MS-DOS, αλλά με ένα ειδικό Windows runtime environment. Άλλες σημαντικές εφαρμογές spreadsheet ήταν το SuperCalc (1980, για το CP/M λειτουργικό), Multiplan(Microsoft), PlanPerfect(WordPerfect Corp.), QuattroPro(Borland), VP-PLANNER και AsEasy As. Από τα μέσα της δεκαετίας του 90, η Microsoft διατηρεί το επικρατών μερίδιο της συγκεκριμένης αγοράς, είχε φτάσει στο σημείο να κατέχει το 90% πριν από μερικά χρόνια, αλλά την τελευταία διετία, το μερίδιο της έχει υποστεί μικρή κάμψη λόγω της ανόδου και της εν-μέρει επικράτησης των Open Source λογισμικών.

Το αρχικό πρότυπο που αποτελούταν από πίνακα με στήλες, γραμμές και αυτόματο συγχρονισμό και απεικόνιση των νέων αποτελεσμάτων έχει επεκταθεί με τη χρήση βιβλιοθηκών μαθηματικών και στατιστικών συναρτήσεων, ευπροσάρμοστων γραφημάτων, ισχυρά επιπρόσθετα στο πρόγραμμα όπως ο Solver του Excel, όμορφες, ελκυστικές και εξαιρετικά χρηστικές διεπαφές, και τη δυνατότητα εγγραφής προσαρμοσμένου κώδικα σε γλώσσες όπως η Microsoft Visual Basic for Applications.

Για επιπλέον πληροφορίες σχετικά με την ιστορία και την εξέλιξη των λογιστικών φύλλων ο αναγνώστης παραπέμπεται στους Power και Walkenbach. Επίσης, ο ίδιος ο Dan Bricklin έχει αρκετές πληροφορίες στην προσωπική του ιστοσελίδα.

1.2 Λογιστικά φύλλα και καθημερινότητα

Σήμερα τα λογιστικά φύλλα θεωρούνται το ποιό επιτυχημένο "προγραμματιστικό" εργαλείο για τον απλό χρήστη. Από την άποψη του αριθμού των χρηστών, μόνο τα πακέτα επεξεργασίας κειμένου υπερτερούν έναντι τους. Τα περισσότερα από αυτά τα λογιστικά φύλλα χρησιμοποιούνται ad-hoc και αχρηστεύονται μετά την αρχική τους χρήση. Πολλές φορές όμως δημιουργούνται για εφαρμογές που τρέχουν για μεγαλύτερο χρονικό διάστημα. Εκατομμύρια χρήστες υιοθετούν τις εφαρμογές λογιστικών φύλλων σε καθημερινή βάση. Υπολογίζεται ότι μέχρι το 2012 θα έχει φτάσει τα 90 εκατομμύρια σε σχέση με τους 4 εκατομμύρια παραδοσιακούς προγραμματιστές. Επίσης ο αριθμός των αμερικάνων εργατών που χρησιμοποιούν

λογιστικά φύλλα είναι ακόμα μεγαλύτερος, και υπολογίζεται γύρω στα 23 εκατομμύρια, ποσό που αποτελεί το 30% της εργατικής δύναμης.

Η αφθονία χαρακτηριστικών και εργαλείων σε αυτές τις εφαρμογές διευκολύνουν τους χρήστες στο να εκτελούν διάφορες σύνθετες διαδικασίες που ποικίλουν από υποθετικούς μικρουπολογισμούς έως και περιορισμένες μορφές διαχείρισης βάσεων δεδομένων. Ελέω των πολύ ισχυρών μαθηματικών συναρτήσεων που αποτελούν αναπόσπαστο κομμάτι των εμπορικών εφαρμογών spreadsheet,(a type system για αναφορές κάπου εδώ!!) δεν χρησιμοποιούνται μόνο σε επιχειρήσεις για λογιστικούς σκοπούς αλλά και για μαθηματικούς και επιστημονικούς υπολογισμούς, τόσο για εκπαίδευση μαθητών αλλά και για την ανάπτυξη εφαρμογών.

Βέβαια η εξαιρετικά διαδεδομένη χρήση των λογιστικών φύλλων έχει και δυσάρεστες συνέπειες. Πολλοί χρήστες έχουν σχετικά ελάχιστη επίσημη εκπαίδευση για χρήση υπολογιστή και οι περισσότεροι από αυτούς μαθαίνουν τις λειτουργίες των λογιστικών φύλλων κυρίως μέσω δοκιμής και λάθους (trial and error). Επίσης συντάσσουν λογιστικά φύλλα αντιγράφοντας τα αυτούσια είτε απλώς αντιγράφοντας τους τύπους κελιών από άλλα λογιστικά φύλλα. Συνεπώς, δεν εκπαιδεύονται στο να αναγνωρίζουν κοινά προγραμματιστικά σφάλματα, και μπορούν έτσι να αποτύχουν στο να αναγνωρίζουν τα λάθη τους. Βλέποντας το από κοινωνιολογική σκοπιά, πολλοί χρήστες είναι πιο επιρρεπείς στο να εμπιστεύονται την έξοδο ενός προγράμματος μόνο και μόνο επειδή το "έβγαλε ο υπολογιστής", χωρίς να έχουν πλήρως κατανοήσει τους τρόπους με τους οποίους μπορεί οι πράξεις τους να έχουν παραποιήσει το αποτέλεσμα.

1.3 Λογιστικά φύλλα vs. Παραδοσιακές Γλώσσες Προγραμματισμού

Από πολύ παλιά ακόμα, ένα καίριο ερώτημα στην έρευνα για την αλληλεπίδραση υπολογιστή και ανθρώπου ήταν το εξής: Πώς μπορεί να σχεδιαστεί ένα λογισμικό ώστε οι χρήστες που δεν είναι προγραμματιστές και δεν έχουν ούτε καν βασικές προγραμματιστικές γνώσεις να μπορούν να αναπτύσσουν τις δικές τους εφαρμογές?

Το 1984, ένα μόνο έτος αφού το Lotus 1-2-3 κατέστησε την παρουσία του αισθητή στην εμπορική αγορά, οι εκπαιδευτικοί άρχιζαν να συζητούν την εμπειρία τους με τη χρησιμοποίηση

λογιστικών φύλλων στην εκπαίδευση. Ο Hsiao [88] τονίζει ότι ενώ οι υπολογιστές είναι σαφώς χρήσιμα εργαλεία για την εκπαίδευση γενικά, ένα από τα κύρια μειονεκτήματα τους είναι ο προγραμματισμός τους. Σε πολλές περιπτώσεις γύρω στο 1985, οι μαθητές χρειάστηκε να μάθουν μια γλώσσα προγραμματισμού ώστε να έχουν ωφέλη από την εκπαίδευση με τον υπολογιστή. Ο Hsiao παρατηρεί ότι η χρήση των λογιστικών φύλλων αποτελεί μια σημαντική λύση για αυτό ακριβώς το πρόβλημα. Η άποψη αυτή υποστηρίζεται και από πολλούς άλλους ερευνητές όπως για παράδειγμα ο Morishita et al που αναφέρουν :

"Η εμπειρία μας στους υπολογιστές ήταν ότι πήρε πολύ χρόνο να μαθευτούν οι γλώσσες προγραμματισμού και ήταν μερικές φορές πολύ δύσκολο να επιτευχθούν τα κατάλληλα αποτελέσματα σε ένα περιορισμένο χρονικό διάστημα. Το λογιστικό φύλλο, εντούτοις, είναι μάλλον εύχρηστο και είναι δυνατές οι (σχεδόν) στιγμιαίες αριθμητικές προσομοιώσεις."

Η διαδικασία ανάπτυξης ενός λογιστικού φύλλου έχει αξιοπρόσεκτο παραλληλισμό με την ανάπτυξη παραδοσιακών πληροφοριακών συστημάτων. Υπάρχουν τρείς φάσεις κλειδιά: σχεδίαση, κατασκευή και επαλήθευση. Η ανάπτυξη του λογιστικού φύλλου μπορεί να αντιμετωπισθεί σαν software engineering task όπου οι πηγές των λαθών πρέπει να ταυτοποιηθούν και να εξαλειφθούν. Λόγω της ποικιλομορφίας των χρηστών των λογιστικών φύλλων, η διαδικασία της τεχνολογίας λογισμικού πρέπει να παρουσιαστεί σε μία μορφή όπου μπορεί να χρησιμοποιηθεί για πληθώρα περιπτώσεων.

Παρόλα αυτά, από τα πρώτα χρόνια της ανάπτυξης των λογιστικών φύλλων και της χρήσης τους για εμπορικούς σκοπούς, ποτέ δεν θεωρήθηκαν τυπική γλώσσα προγραμματισμού, με όσες συνέπειες κι αν είχε αυτό. Είναι χαρακτηριστικό ένα άρθρο του Romrnert J. Casimir στα μέσα της δεκαετίας του 80 με τίτλο Real programmers don't use spreadsheets απ' όπου και βλέπουμε:

"Η παραμέληση των spreadsheets από την επιστημονική βιβλιογραφία των

γλωσσών προγραμματισμού έχει προκληθεί από μία αποστροφή σε μία γλώσσα που δημιουργήθηκε από φοιτητές που είχαν την κακή έμπνευση να χρησιμοποιήσουν τις ιδέες τους για να αυξήσουν το εισόδημα τους αντί να ξεκινήσουν μια καριέρα στην Επιστήμη Υπολογιστών με σκοπό ένα βραβείο Turing. Από την άλλη, καθώς είναι λίγο απίθανο οι ειδικοί των γλωσσών προγραμματισμού να έχουν τέτοιες μοχθηρές σκέψεις, τα λογιστικά φύλλα ίσως να μην έχουν κανένα ουσιαστικό ενδιαφέρον. Παρόλο που τέτοιου είδους ερευνητικές υποθέσεις δεν έχουν βρει το δρόμο της δημοσίευσης, ο σκοπός αυτού του paper είναι να αναφέρει μερικά πράγματα που πρέπει να ξέρουν όσοι ασχολούνται με λογιστικά φύλλα και να προειδοποιήσουν τυχόν ερευνητές για την ανία της ενασχόλησης με τον συγκεκριμένο τομέα."

Όμως δεν είναι λίγες φορές τα τελευταία χρόνια που έχουν βγει στην επιφάνεια εργασίες ερευνητών που αντιπροτείνουν τα λογιστικά φύλλα και την ανάπτυξη τους παραλληλίζοντας τα ξεκάθαρα με παραδοσιακό προγραμματισμό, συνειφρείοντας στην έρευνα τους μεθόδους τεχνολογίας λογισμικού. Για πληθώρα τέτοιων δημοσιεύσεων ο αναγνώστης παραπέμπεται στην βιβλιογραφία και κυρίως στην ομάδα της Karin Hodnigg στο πανεπιστήμιο του Klagenfurt στην Αυστρία, όπου πραγματικά προτείνουν την αντιμετώπιση των spreadsheets σαν γλώσσα προγραμματισμού με απώτερο σκοπό την ορθή εξάλειψη των σφαλμάτων σε αυτά από το πρώτο στάδιο και την σωστή διδακτική προσέγγιση τους σε περιπτώσεις σχολείων και λοιπών ιδρυμάτων. Θα αναφερθούμε αναλυτικά στο μοντέλο που προτείνουν σε επόμενο κεφάλαιο.

ΛΟΓΙΣΤΙΚΑ ΦΥΛΛΑ ΚΑΙ ΛΟΓΙΣΜΙΚΟ: ΠΟΙΑ Η ΔΙΑΦΟΡΑ;

Το Λογισμικό έχει γραφτεί με επαγγελματική μέθοδο από Επαγγελματίες: το Λογιστικό Φύλλο έχει γραφτεί από Χρήστες! Παρόλο που η φράση αυτή είναι αληθής, αφήνει όμως λάθος εντυπώσεις. Οι επαγγελματίες προγραμματιστές, εάν εργάζονται επαγγελματικά, θα δημιουργήσουν τα προϊόντα τους βάση ενός σχεδίου, κάποιου εννοιολογικού πρότυπου ή κάποιων προδιαγραφών που έχουν άμεση συνάρτηση του προβλήματος εφαρμογής με μία αλγοριθμική λύση, όπου ο αλγόριθμος συνήθως υπολογίζει κάποιες ιδιοσυγκρασίες του υπολογιστή (δεδομένα εισαγωγής και εξόδου είναι τα βασικότερα). Οι χρήστες λογιστικών φύλλων είναι οι τελικοί χρήστες και δεν είναι επαγγελματίες προγραμματιστές. Αλλά δεν παύουν να είναι επαγγελματίες στον δικό τους τομέα εφαρμογής.

Με αυτή τους την ιδιότητα, όταν εκφράζουν γραπτώς προβλήματα/λύσεις, όπως ο οιονδήποτε επιχειρεί να γράψει κάτι με νόημα, εκφράζονται βάση κάποιου εννοιολογικού προτύπου. Αυτό συμβαίνει και όταν εκφράζουν τις σκέψεις τους σε ένα λογιστικό φύλλο. Η μόνη διαφορά που έχουν από τον επαγγελματία προγραμματιστή είναι ότι το πρότυπο που χρησιμοποιούν δεν περιέχει στοιχεία προγραμματισμού. Συσχετίζει εφαρμογές σε πίνακα δύο διαστάσεων με αριθμούς αναμεμιγμένους με επεξηγηματικό κείμενο. Οι αριθμοί είναι περαιτέρω εννοιολογικά συσχετιζόμενοι κατά έναν από τους δύο τρόπους που περιγράφουμε :

1 Ο δεδομένος αριθμός είναι αποτέλεσμα υπολογισμού άλλων αριθμών τοποθετημένων, (ή που θα γραφτούν αργότερα), σε μία συγκεκριμένη θέση.
2 Ο δεδομένος αριθμός είναι μέρος ενός συνόλου αριθμών οι οποίοι έχουν εννοιολογικά όλοι τον ίδιο ρόλο. Ο «ρόλος» αυτός συνήθως εκφράζεται από την γεωμετρική τους γειτνίαση (φυσικό πεδίο). Όμως, κατόπιν θα αναφερθούμε και σε περιπτώσεις όπου ο εννοιολογικός συσχετισμός των αριθμών δεν παρουσιάζει γεωμετρική γειτνίαση (λογικό πεδίο).

Εμπειρογνώμονες στα λογιστικά φύλλα θα παρατηρήσουν ότι οι δύο αυτές περιπτώσεις δεν είναι απόλυτα περιεκτικές. Εμείς όμως υποστηρίζουμε ότι καλύπτουν ένα ευρύ φάσμα, τουλάχιστον το φάσμα των «μη εμπειρογνωμόνων» το οποίο χρησιμοποιούν οι χρήστες.

Τα λογιστικά φύλλα είναι συστήματα εύχρηστα, δεν απαιτούν πολύ εκπαίδευση σε τυπικές μεθόδους σχεδιασμού και προγραμματισμού, και τα αποτελέσματα τους είναι εμφανή κατά τη διάρκεια της όλης διαδικασίας, κάτι που δεν ισχύει με τα συμβατικά προγράμματα, και είναι επίσης γραμμένα με διαφορετικό τρόπο. Υπάρχει μία αίσθηση άμεσης πληροφόρησης όταν το περιεχόμενο ενός κελιού προσδιορίζεται. Αυτός ο εύκολος τρόπος της άμεσης πληροφόρησης οδηγεί στην ανάπτυξη μιας εμπειρικής μεθόδου, με επιλογές διαγραφής και επικόλλησης, αντιγραφής και τροποποίησης- ένα μίγμα που θα φαίνεται τρομακτικό σε έναν τυπικό μεθοδικό προγραμματιστή. Λαμβάνοντας όλα αυτά υπόψη είναι εμφανές ότι, ανεξάρτητα από την πραγματική τους ιδιότητα, τα λογιστικά φύλλα - «λογισμικά» δεν εφαρμόζουν συμβατικές γνώσεις λογισμικού ελέγχου, ή τις εφαρμόζουν σε περιορισμένη έκταση. Εφαρμόζεται ειδικά από την άποψη των υπολογισμών των λογιστικών φύλλων οι οποίοι είναι βασικά αριθμητικοί υπολογισμοί. Θα επανέλθουμε σε αυτή τους την ιδιότητα στην ενότητα 5.2.

Για αυτό και εμείς βασίζουμε τη προσέγγισή μας στις ιδιότητες

των λογιστικών φύλλων που ήδη υπάρχουν, και στις ήδη υπάρχουσες διαδικασίες εγγραφής των λογιστικών φύλλων, παρά σε μία διαδικασία σχεδιασμού και ποιοτικής εξέλιξης. Ειδικά η μελέτη μας επικεντρώνεται στα «λογιστικά φύλλα όπως ήδη υπάρχουν». Αυτό μας οδηγεί στη συζήτηση περί οπτικής αναπαράστασης του προτύπου του φυσικού επακόλουθου και τον έλεγχο ευλογοφάνειας. Για την οπτική αναπαράσταση του προτύπου προτείνουμε στους χρήστες να μετατρέψουν τα προβλήματα/λύσεις σε δομές δύο διαστάσεων και να τονίσουν ιδιαίτερα παρατυπίες σε αυτή την μετατροπή. Όσο για τον έλεγχο ευλογοφάνειας βασιζόμαστε στη διαίσθηση του χρήστη ως προς τα νοηματικά όρια των δεδομένων (αριθμών!) που επεξεργάζεται στο λογιστικό φύλλο. Το να αναλύσουμε στρατηγικές που θα εξασφάλιζαν την ποιότητα των λογιστικών φύλλων (με επίκεντρο την εξέλιξη των λογιστικών φύλλων) είναι πέρα από τους σκοπούς αυτής της μελέτης. Πριν ερευνήσουμε αυτές τις δύο πτυχές του θέματος, θα ξεκινήσουμε με τον προσδιορισμό μερικών βασικών όρων που θα μας φανούν χρήσιμοι στη συνέχεια, και θα αναφέρουμε μερικά τυπικά σφάλματα σε λογιστικά φύλλα και τις σχετικές τους κατηγορίες.

Σύμφωνα με έναν βασικό ορισμό [Filby, 1998], ο βασικός στόχος προγραμματισμού λογιστικών φύλλων είναι «διαχείριση και παρουσίαση των δεδομένων που βρίσκονται σε μορφή πίνακα». Η ευκολονόητη λειτουργία των λογιστικών φύλλων αποκρύπτει σε μεγάλο βαθμό την ιδιότητά τους ως προγράμματα. Η εισαγωγή σταθερών τιμών σε κάποια κελιά και ενός τύπου σε άλλα κελιά δεν εμφανίζεται ως μορφή προγραμματισμού. Μάλλον συγκρίνεται με τη χρήση φορητής αριθμομηχανής. Η άμεση παρουσίαση του αποτελέσματος στηρίζει αυτή την αντίληψη. Στη πραγματικότητα όμως, αυτό είναι λογικό αποτέλεσμα και συνέπεια της γλώσσας τύπων των λογιστικών φύλλων η οποία είναι εγγενώς συναρτησιακής φύσης.

Αυτό επιτρέπει την εισαγωγή των αρχαρίων με ένα μάλλον «απλό» παράδειγμα στη δημιουργία λογιστικών φύλλων, το οποίο επίσης είναι και η αιτία της υψηλής του απήχησης. Η δημοτικότητα αυτή οφείλεται σε αρκετούς παράγοντες όπως:

- Τα προγράμματα λογιστικών φύλλων παρέχουν άμεσα αποτελέσματα. Άμεση αξιολόγηση του τύπου (μία άμεση επιβεβαίωση του ερωτήματος «είναι αυτή η αναμενόμενη τιμή;) και άμεσα ορατή αναπαραγωγή αλλαγών (εάν ένα κελί αλλάζει, αλλάζουν και τα εξαρτώμενα από αυτό κελιά) παρέχουν διαβεβαίωση στον χρήστη.
- Το πλέγμα του πίνακα παρέχει ένα πλαίσιο όπου το πρόβλημα μπορεί να ρυθμιστεί. Δεν υπάρχει κάποιο καινούριο πρότυπο ή

κάποια δομή δεδομένων που πρέπει ο χρήστης να εφεύρει. Αυτό προσφέρει ένα επίπεδο περισπασμού, το οποίο στηρίζει προβλήματα διαμόρφωσης και αναφέρεται άμεσα στον προβληματικό τομέα του χρήστη.

- Η πολυπλοκότητα είναι κρυμμένη πίσω από τους τύπους που χειρίζονται μόνο ένα ιδιαίτερο υπό - πρόβλημα. Αυτή η έννοια του διαμορφώσιμου είναι ένα ακόμη πλεονέκτημα για τον χρήστη: η σκέψη σε επίπεδο κελιών, διαιρεί και απλουστεύει το πρόβλημα, «ένα κελί μετά από το άλλο» μέχρι το κάθε υπό - πρόβλημα να αντιμετωπισθεί και να λυθεί ολόκληρο το πρόβλημα.
- Ένα άλλο σημαντικό χαρακτηριστικό των λογιστικών φύλλων είναι η διάταξή τους. Η σχέση των κελιών εντός του χώρου και οι αλληλεξαρτήσεις τους δεν διευκολύνουν μόνο τον αναγνώστη τους. Ορισμένα στάδια κατά τη δημιουργία τους (αντιγραφή/επικόλληση, συναρτήσεις συνάθροισης) εξαρτώνται από τη διάταξη.
- Η χρήση ερμηνευτικής διάταξης στα λογιστικά φύλλα (με το πλέγμα και ένα μειωμένο σύνολο συναρτήσεων και αριθμητικών πράξεων) είναι περισσότερο ενορατικό από τη χρήση μιας διαδικαστικής ή μιας γλώσσας προγραμματισμού με προσανατολισμό τον αντικειμενικό σκοπό του προγράμματος. Σύμφωνα με τους Navarro - Prieto και Canas, οι προγραμματιστές των λογιστικών φύλλων έχουν αναπτύξει καλές διανοητικές δομές και για τον έλεγχο και για τη ροή δεδομένων πληροφόρησης. Οι προαναφερόμενοι συγγραφείς αναφέρουν ότι τα οπτικά χαρακτηριστικά των λογιστικών φύλλων διευκολύνουν τους προγραμματιστές να αναπτύξουν μια αναπαράσταση του προγράμματος βασισμένη στη ροή δεδομένων.

Αλλά αυτά τα ίδια χαρακτηριστικά γνωρίσματα είναι επίσης υπεύθυνα για την πολυπλοκότητα μεγάλων και συνεχώς εξελισσόμενων προγραμμάτων λογιστικών φύλλων. Στα προγράμματα λογιστικών φύλλων μόνο οι τιμές αναφέρονται (οι τύποι εύκολα εκτιμώνται), και οι τύποι του φύλλου είναι κρυμμένοι, εκτός εάν ο χρήστης επιλέξει το εν λόγω κελί. Αυτό οδηγεί σε ένα συγκεκριμένο πρόβλημα των προγραμμάτων των λογιστικών φύλλων: η απόκρυψη (ελέγχου και) ροής δεδομένων πίσω από «στατικές» αξίες δεν διευκολύνει την κατανόηση ενός υπάρχοντος προγράμματος λογιστικών φύλλων. Έχει μάλλον επιζήμια αποτελέσματα στη συντήρησή του. Η συνύφανση στη διάταξη των αποτελεσμάτων και οι εξαρτήσεις των υπολογισμών είναι επίσης πηγές εννοιολογικής πολυπλοκότητας. Οι αναφορές των κελιών μπορούν να έχουν μεγάλη απόσταση και εφόσον οι σύνδεσμοι μεταξύ

κελιών δεν παρουσιάζονται με σαφήνεια, η κατανόηση της ροής δεδομένων ενός λογιστικού φύλλου είναι μία αξιόλογα δύσκολη εργασία. Η χωρική σχέση των κελιών μπορεί να επεκτείνεται και εκτός του τμήματος που είναι ορατό στην οθόνη του προγραμματιστή. Ο Sanjanemi (1998) αναφέρει, «*οι υπολογισμοί των λογιστικών φύλλων είναι δύσκολο να κατανοηθούν και κατά συνέπεια πολλά λάθη περνούν απαρατήρητα. Το πρόβλημα με τα κρυμμένα αυτά λάθη κείτεται στο ότι η δομή των υπολογισμών δεν είναι ορατή*».

Και η πολυπλοκότητα αυτή των εννοιών αλλά και η σπουδαιότητά τους ως βασικά εργαλεία στη λήψη αποφάσεων των επιχειρήσεων απαιτούν μία πιο σοβαρή και πιο συγκροτημένη διαχείριση των λογιστικών φύλλων. Το μεγάλο ποσοστό σφαλμάτων που βρίσκονται στα λογιστικά φύλλα των επιχειρήσεων είναι ενδεικτικά ότι το ζήτημα της ποιότητας των λογιστικών φύλλων δεν μπορεί να επιλυθεί με τη χρήση πιο ισχυρών εργαλείων. Πρέπει μάλλον να συμφωνήσουμε με τη δήλωση του Hoare (1999) ότι η «*η θεωρία δεν είναι απόλυτα κατανοητή αλλά το παράδειγμα πρέπει να μελετηθεί περαιτέρω*».

2. Το μοντέλο του λογιστικού φύλλου

Αναφορικά με ένα γενικευμένο μοντέλο των λογιστικών φύλλων και περεταίρω σύνδεσης

Χαρακτηριστικά του λογιστικού φύλλου

Το λογιστικό φύλλο σαν οντότητα βασίζεται σε μία ν-διάστατη απεικόνιση της πληροφορίας. Υπάρχει μια σειρά ιδιοτήτων η οποία παρέχεται από τις εφαρμογές λογιστικών φύλλων για να υποστηρίξουν τον επίδοξο προγραμματιστή.

- **Διεπαφή Χρήστη σε μορφή πίνακα**

Η οργάνωση της πληροφορίας σε κελιά, στήλες και γραμμές μπορεί να βοηθήσει τους χρήστες να χαρτογραφήσουν το μοντέλο τους σε ένα ήδη καλά ανεπτυγμένο πλαίσιο εργασίας παρά να ανακαλύψουν πάλι τον τροχό, δηλαδή μια νέα bottom up προσέγγιση. Το πλέγμα ενός λογιστικού φύλλου μπορεί να γεμίσει με τιμές, ετικέτες και τύπους για να λυθεί το πρόβλημα που έχει ο τελικός χρήστης.

- Τοπικότητα

Οι προγραμματιστές λογιστικών φύλλων τείνουν να μην σκέφτονται σε σφαιρικό επίπεδο, σε γενικούς όρους. Αντίθετα, μειώνουν την τεράστια πολυπλοκότητα του προβλήματος θεωρώντας κάθε κελί σαν ανεξάρτητη οντότητα. Έτσι επικεντρώνονται σε μία πολύ τοπική όψη του θέματος τους. Η εκπαίδευση στα λογιστικά φύλλα πρέπει να το λαμβάνει πάντα αυτό υπόψιν της, καθώς η εμβέλεια είναι μια έννοια που πρέπει να κατανοηθεί πλήρως

- Κατασκευάσματα Υψηλού Επιπέδου

Η γλώσσα με την οποία γράφονται οι τύποι στα λογιστικά φύλλα είναι υψηλού επιπέδου και περιέχει απλούς ελέγχους όπως το if-clause. Επίσης προσφέρει μία πληθώρα από αριθμητικές, οικονομικές, στατιστικές και λογικές συναρτήσεις. Συνδυάζοντας τα όλα αυτά και εμφανίζοντας τα ενδιάμεσα αποτελέσματα τους οδηγούμαστε στην μεγάλη εκφραστικότητα των λογιστικών φύλλων χωρίς απώλεια της απλότητας τους.

-Βαθμός εκμάθησης

Ξεκινώντας την ανάπτυξη ενός λογιστικού φύλλου για κάποια ωφέλιμη χρήση, χρειάζεται μόνο πενιχρή γνώση για συγκεκριμένες συναρτήσεις και κατασκευαστές. Όσο αυξάνεται η εμπειρία, ο προγραμματιστής αποκτά νέες γνώσεις για τα στοιχεία της εφαρμογής και τα χρησιμοποιεί αυξητικά.

Συνδυάζοντας μία εκφραστικά απλή γλώσσα προγραμματισμού υψηλού επιπέδου με ισχυρό οπτικό σχήμα (ανάπαρασταση) για οργάνωση της πληροφορίας και την επίδειξη της, οι εφαρμογές λογιστικών φύλλων παρέχουν ένα στιβαρό προγραμματιστικό περιβάλλον. Η δύναμη τους χαρακτηρίζεται από τους Nardi and Miller(1990) από τα εξής:

1. Ένα πεπερασμένο σύνολο από λειτουργίες υψηλού επιπέδου με σαφείς στόχους.

Η δηλωτική και εκφραστική γλώσσα των συναρτήσεων επηρεάζει την μοντελοποίηση των εφαρμογών λογιστικών φύλλων.

2. Ένα ισχυρό οπτικό σχήμα

Το δεύτερο μείζον στοιχείο της εφαρμογής είναι η διάταξη της πληροφορίας. Η πινακοειδής διάταξη υποστηρίζει την λύση τριών σημαντικών προβλημάτων

α. αντίληψη της πληροφορίας

β. οργάνωση της πληροφορίας

γ. απεικόνιση της πληροφορίας

2.1 Το Μοντέλο

Για να προσδιορίσουμε το στόχο αυτής της έρευνας, μπαίνουμε στον πειρασμό να αναρωτηθούμε αν υπάρχει «γλώσσα λογιστικών φύλλων», και αν υπάρχει, ποια μπορεί να είναι αυτή η γλώσσα. Για ένα ορισμένο γινόμενο, δεν υπάρχει διαφορά αν κάποιος πληκτρολογήσει = IF (A1=B1;…) ή αν πληκτρολογήσει =WENN(A1=B1;…). Παρομοίως, δεν έχει σημασία αν η εντολή πληκτρολογήθηκε, ή επιλέχθηκε με το ποντίκι από κάποιον πίνακα ελέγχου ή έχει αντιγραφεί από άλλο κελί που περιέχει τον ίδιο τύπο και κατόπιν έχει περαιτέρω επεξεργασθεί. Η ιδέα είναι ότι το σύστημα παρέχει την έννοια μιας εναλλακτικής λύσης και η έννοια αυτή παρουσιάζεται με διαφορετική γλωσσική μορφή στο χρήστη. Οι διαφορές όμως, στη γλωσσική μορφή είναι μάλλον επιφανειακές και οι χρήστες πρέπει να αναπτύξουν ένα νοητικό πρότυπο βασισμένο στις έννοιες των τύπων που εφαρμόζονται στα διάφορα γινόμενα του λογιστικού φύλλου. Εφόσον όλες οι συναρτήσεις βασίζονται σε μαθηματικές έννοιες, οι χρήστες δεν μπορούν να κατηγορηθούν εάν υποθέσουν ότι το φύλλο συμπεριφέρεται με ακριβώς τον ίδιο τρόπο συμπεριφοράς των μαθηματικών συναρτήσεων.

Όπως τα μαθηματικά των συναρτήσεων των λογιστικών φύλλων είναι γνωστά στους εμπειρογνώμονες του τομέα, η συναρτησιακή φύση των υπολογισμών στα κελιά καθιστά εντυπωσιακά απλό τον προγραμματισμό των λογιστικών φύλλων. (Mostrom, 1998). Οι Mostrom και Carr εντάσσουν όπως:

1 Υπάρχει ένα πινακοειδές πλέγμα από (απευθυνόμενα) κελιά.
2 Το κελί μπορεί να περιέχει ένα τύπο ή μία σταθερή τιμή.
3 Η γλώσσα του τύπου είναι ερμηνευτική, και έχει τη μορφή =<cell_addr> <operator>
<cell_addr> or =<function>(<cell_addr1>,..,< cell_addrn>

[;<system parameter>]).

1 Τα κελιά μπορούν να αναφερθούν ως «μοναχικά» ή ως σειρά αναφοράς (A1:A12). Μία αναφορά μπορεί να είναι είτε απόλυτη είτε σχετική.

Το παράδειγμα συναρτησιακού προγραμματισμού είναι σαφώς αντίθετο του «συμβατικού» προγραμματισμού. Η γλώσσα των λογιστικών φύλλων είναι πολύ ερμηνευτική όπως προσδίδει «ιδιαίτερη έμφαση στην αξιολόγηση των εκφράσεων» (Montigel, 2002) και είναι μαθηματικά εξακριβώσιμη. Η γλώσσα των λογιστικών φύλλων εστιάζεται στις χωρικές σχέσεις των δεδομένων, όχι στη χρονική τους σειρά.

Υπάρχουν πολύ συγκεκριμένες έννοιες για την εφαρμογή ενός πρώτου προγράμματος λογιστικών φύλλων. Για αυτό και η υπόθεση ότι η γλώσσα των λογιστικών φύλλων είναι ¨μια γλώσσα προγραμματισμού για τις μάζες» (Molstrom, 1998) φαίνεται να δικαιολογείται. Ακόμη και χωρίς ειδική κατάρτιση ο καθένας μπορεί να την χειριστεί στον τομέα του. Αυτή η ευκολία στη χρήση (προσδιορίζοντας τι να κάνει και πώς να το κάνει) όπως και το πινακοειδές πλέγμα ως πολύ ισχυρή οπτική διασυνδετική διάταξη συντελούν στην μεγάλη επιτυχία των λογιστικών φύλλων.

2.1.1 Σημειογραφία και Γλώσσα των Λογιστικών Φύλλων

Για τη διευκόλυνση της περαιτέρω ανάγνωσης μερικοί βασικοί όροι αναθεωρούνται εν συντομία.

Η διεύθυνση ενός κελιού CA είναι n-πολλαπλό. Σε ένα κοινό λογιστικό φύλλο δύο διαστάσεων η διεύθυνση του κελιού είναι ένα ζεύγος (γραμμής και στήλης) που προσδιορίζει τη τοποθεσία του κελιού σε αυτό πλέγμα δεδομένων το οποίο έχει δύο διαστάσεις. Η τιμή v είναι ένα στοιχείο αποτέλεσμα ομάδας τιμών v= **Αριθμοί – σειρά συμβόλων-{σφάλμα}- {απροσδιόριστο}.**

Ένα λογιστικό φύλλο S αποτελείται από μια ομάδα κελιών. Ένα κελί C αντιπροσωπεύεται από ένα τριπλό (ca:διεύθυνση, v:τιμή, f(<ορίσματα>)à Τιμή), ca είναι η μοναδική διεύθυνση του κελιού, v είναι η τιμή που παρουσιάζεται από τον υπολογισμό του τύπου f. (v=eval(f)). Σε περίπτωση που το f υποβαθμισθεί σε ένα σταθερό eval(f) τότε αποδίδει την ταυτότητα αυτού του σταθερού.

Ορίσματα της συνάρτησης f μπορούν να είναι είτε σταθερά είτε συναρτήσεις σε αναφορές κελιών. Μια συνάρτηση αναφοράς κελιού cref (scr:CA, id:CA) à η τιμή που αποδίδεται στο κελί στη διεύθυνση scr+id, όπου Scr είναι η διεύθυνση του παραλήπτη κελιού οπου κάθε στοιχείο του πολλαπλού id προστίθεται για σχετική αναφορά. Με την απόλυτη τοποθέτηση η αντίστοιχη θέση του id προστίθεται άμεσα.

Η γλώσσα των τύπων των λογιστικών φύλλων είναι η βάση της έκφρασης του τύπου f. Το f προσδιορίζεται ως f(<όρισμα>)à Τιμή και το f καθορίζεται από μια σειρά συμβόλων που ξεκινούν με το «=» και συνεχίζουν είτε με μία συνάρτηση είτε με μία μαθηματική πράξη επί σταθερών ή επί αναφορικών κελιών, (Clermont, 2002). Οι σταθερές τιμές δεν θεωρούνται ως εισαγωγές του τύπου επειδή δεν χρειάζονται εκτίμηση εκτός εάν ο ίδιος ο τύπος επανεκτιμείται.

Ένα πρόγραμμα λογιστικών φύλλων ορίζεται ως ένα υποσύνολο του λογιστικού φύλλου, που περιέχει ολα τα μή κενά κελιά τύπου και τα κελιά με τις απόλυτες συναρτήσεις. Κελιά με σχετικές συναρτήσεις είναι μόνο μέρος του προγράμματος του λογιστικού φύλλου εάν περιέχουν ένα τύπο. Εάν περιέχουν σταθερά τότε θεωρούνται ως *κελιά εισαγωγής δεδομένων*. Μία *περίπτωση λογιστικού φύλλου* είναι ένα μίγμα προγράμματος λογιστικού φύλλου και εισαγωγής δεδομένων του φύλλου.

2.2 *Ορολογία*

Εφόσον ο όρος «λογιστικό φύλλο» είναι ήδη υπερβεβαρημένος θα εξηγήσουμε τη σημασία όρων σχετικών με τα λογιστικά φύλλα στα οποία θα αναφερθούμε σε αυτή μας τη μελέτη.

Το *κελί* είναι η ατομική μονάδα του λογιστικού φύλλου και υπάρχει σε πέντε καταστάσεις :α) μπορεί να είναι *κενό*, β) μπορεί να φέρει μία *σταθερή τιμή*, *γ)*μπορεί να έχει μία *εισαγομένη τιμή* την οποία προσδιορίζει ο χρήστης του λογιστικού φύλλου, δ) μπορεί να έχει τιμή που προσδιορίζει ένας *τύπος*, ή ε) μπορεί να φέρει μία *ετικέτα* η οποία προσδιορίζει τα περιεχόμενα ενός συνόλου κελιών.

Ένα λογιστικό φύλλο είναι ένας μαθηματικός πίνακας κελιών n- διαστάσεων. Κάθε κελί προσδιορίζεται από n-συντεταγμένες, μοναδικές για κάθε κελί. Εάν το n=2, όπως στις τυπικές περιπτώσεις, το κελί είναι μοναδικά προσδιορισμένο από την τομή

γραμμής και στήλης.

Η αναφορά του κελιού είναι αναφορά στη τιμή άλλου κελιού η οποία μπορεί να είναι σχετική ή απόλυτη. Η ακριβής θέση του κελιού προσδιορίζεται από δύο συντεταγμένες, στην πρώτη περίπτωση η προέλευση είναι το αναφερόμενο κελί και στην τελευταία η επάνω αριστερή γωνία του λογιστικού φύλλου.

Ο τύπος είναι μία μαθηματική έκφραση η οποία περιέχει αναφορές κελιών, μαθηματικά σύμβολα, συναρτήσεις, και σταθερές τιμές. Τουλάχιστον μια αναφορά κελιού πρέπει να υπάρχει σε κάθε υπολογιστική έκφραση του τύπου. Ο τύπος αποδίδει ακριβώς ένα αποτέλεσμα και δεν παρουσιάζει επιπλοκές.

Η Βασική Γλώσσα των Λογιστικών Φύλλων (Spreadsheet Core Language - SCL) είναι ένα σύνολο γλωσσικών εκφράσεων τα οποία προσδιορίζουν τη ροή των δεδομένων (αναφορές κελιών) και την χρήση των δεδομένων (τύποι) στο πρόγραμμα των λογιστικών φύλλων. Οι ιδιότητες των συναρτήσεων του λογιστικού φύλλου εκφράζονται με SCL. Η πρωτόγονη χρήση *αντιγραφής και επικόλλησης* επίσης θεωρούνται εκφράσεις της ίδιας γλώσσας, εφόσον η χρήση τους γίνεται σε πλαίσια λογικών πεδίων.

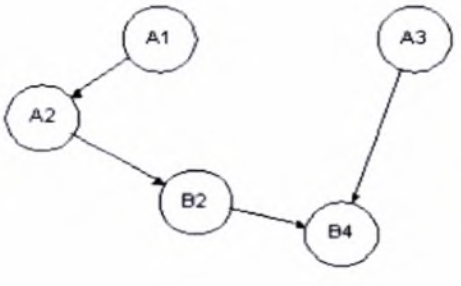
Η Γλώσσα των Λογιστικών Φύλλων (Spreadsheet Language-SL) περιέχει την Βασική Γλώσσα των Λογιστικών Φύλλων (SCL) μαζί με εκφράσεις για τη διαμόρφωση της διάταξης του λογιστικού φύλλου.
Το Πρόγραμμα Λογιστικών Φύλλων (Spreadsheet Program - SP) είναι το σύνολο των προδιαγραφών της ροής δεδομένων μεταξύ κελιών, η χρήση των δεδομένων στα κελιά και οι τιμές των σταθερών κελιών.
Ένα Στιγμιότυπο Λογιστικού Φύλλου (Spreadsheet Instance - SI) είναι πρόγραμμα λογιστικού φύλλου όπου όλα τα κελιά εισαγωγής έχουν ορισμένες τιμές. Ένα πρόγραμμα λογιστικών φύλλων μπορεί να αναπαραχθεί ως στιγμιότυπο πολλαπλές φορές. Αλλάζοντας μία από τις τιμές που εισάγουμε, το στιγμιότυπο λογιστικού φύλλου αλλάζει σε άλλο στιγμιότυπο του ιδίου προγράμματος.
Ένα Σύστημα Λογιστικών Φύλλων (Spreadsheet System) είναι ένα ολοκληρωμένο περιβάλλον, όπου προγράμματα λογιστικών φύλλων μπορούν να δημιουργηθούν, να αναπαραχθούν στιγμιότυπα και να γίνει η επιμέλειά τους. Το σύστημα των λογιστικών φύλλων ερμηνεύει μία συγκεκριμένη γλώσσα λογιστικών φύλλων.

Μία περιοχή είναι ένα σύνολο συναφών κελιών. Αν τα κελιά είναι σε γειτονικό χώρο εντός της περιοχής που επιλέγει ο προγραμματιστής, τότε αναφερόμαστε σε φυσική περιοχή. Μία φυσική περιοχή συνήθως χρησιμοποιείται για την ανεύρεση συναρτήσεων όπως Άθροισμα, Μέγιστη Τιμή, ή Μέσο Όρο (SUM, MAX, AVG). Εάν η σχέση δεν βασίζεται σε φυσική περιοχή, γειτονεύοντα κελιά, αλλά σε ομοιότητες από χρήση δεδομένων, ή από τον τρόπο δημιουργίας της

(π.χ. *αντιγραφή και επικόλληση*), τότε αναφερόμαστε σε λογική περιοχή. Τα κελιά των φυσικών περιοχών πρέπει να είναι γειτονικά σε χώρο, αλλά αυτό το κριτήριο δεν εφαρμόζεται και στις λογικές περιοχές . Η λογική περιοχή περιγράφει ένα είδος εννοιολογικής συνοχής μεταξύ των κελιών. Αν δεν γνωρίζουμε τον τρόπο με τον οποίο δημιουργήθηκαν τα κελιά (π.χ. *αντιγραφή και επικόλληση* από την ίδια προέλευση), τότε χρησιμοποιώντας εμπειρική ανιχνευτική μέθοδο βασισμένη σε ομοιομορφίες των αναφορών και τύπων μιας ομάδας κελιών προσδιορίζουμε τη λογική περιοχή.

2.3 Γραφική Αναπαράσταση

Κατά τη διαδικασία δόμησης ενός εννοιολογικού προτύπου μιας περίπτωσης λογιστικού φύλλου, μια αναπαράσταση της ροής δεδομένων διαισθητικά έρχεται στο μυαλό. Τα κελιά διασυνδέονται με αναφορές και πληροφορίες δίνονται από το ένα κελί στο άλλο. Οι αναφορές των κελιών αναπαριστούν τόξα όπου διαρρέουν τα δεδομένα. Διαφορετικές αναπαραστάσεις και διαφορετικές συναθροίσεις έχουν προταθεί για να απεικονίσουν τη ροή των δεδομένων. Για να παραμείνουμε σε στοιχειώδες επίπεδο, θα εστιάσουμε την προσοχή μας στο Γράφημα Εξάρτησης Δεδομένων (Data Dependency Graph -DDG).



	A	B	C
1	1		
2	=A1*10	=A2*10	
3	100		
4		=(B2+A3)/(B2*A3)	
5			

Εικόνα 1: Παράδειγμα λογιστικού φύλλου και το αντίστοιχο γράφημα εξάρτησης δεδομένων

Γράφημα Εξάρτησης Δεδομένων

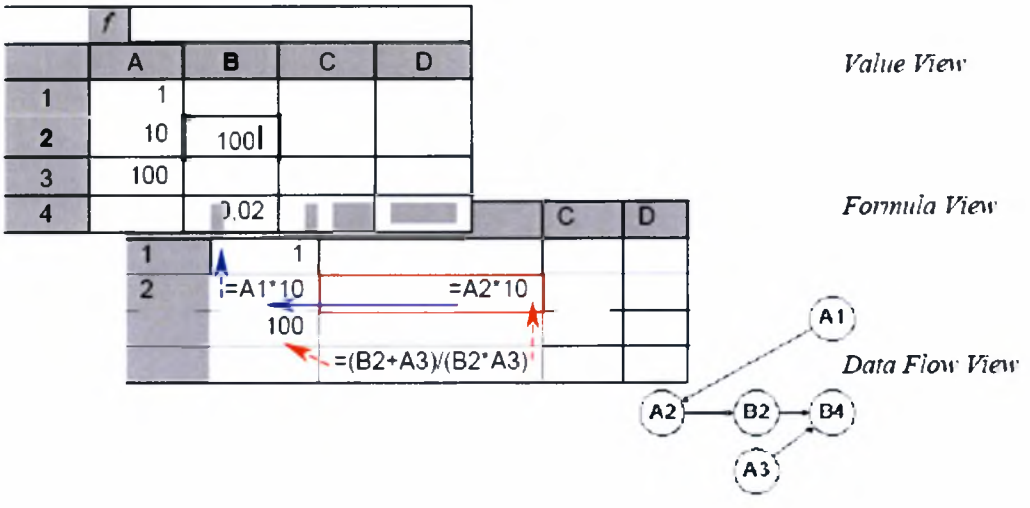
Κάθε προσδιορισμένο κελί («με περιεχόμενο») ενός λογιστικού φύλλου αναπαρίσταται από ένα κατακόρυφο σημείο, οι σχέσεις μεταξύ των κελιών (αναφορές κελιών) αναπαρίστανται με κατευθυνόμενα

γραφήματα. Η αναπαράσταση ενός λογιστικού φύλλου ως Γράφημα Εξάρτησης Δεδομένων είναι διπλής χρήσης. Η μία χρήση του είναι ότι αποτελεί τη βάση επεξεργασίας της υπολογιστικής διαδικασίας. Η δεύτερη χρήση του ενισχύει τη σύλληψη εννοιών από τους χρήστες ως προς το τι έχει καθορισθεί από ένα σύνολο, ειδάλλως ανεξάρτητων, τύπων που περιέχουν αναφορές στα άμεσα ορίσματά τους. Ωστόσο, όπως παρουσιάζεται κατωτέρω, η σημασιολογία της ροής δεδομένων δεν είναι σε όλες τις περιπτώσεις κατάλληλη και ως σημασιολογία των λογιστικών φύλλων.

Ο Εννοιολογικά Λανθασμένος Συνδυασμός

Για τον προγραμματιστή ενός λογιστικού φύλλου που αρχίζει από μηδενική βάση, η δημιουργία του λογιστικού φύλλου είναι σαν να κτίζει ένα σπίτι. Όπως θα τοποθετούσε πλίνθο σε πλίνθο, τύποι εισάγονται στα κελιά που μπορούν να αξιολογηθούν βάση των ήδη καθορισμένων κελιών. Με αρκετά μικρά προβλήματα, όπου η εισαγωγή στα κελιά είναι άμεσα ορατή, αυτή η κελί-με-κελί προσέγγιση μπορεί να ακολουθήσει ένα υπονοούμενο (DIG) Γράφημα Εξάρτησης Δεδομένων, σχεδόν σε χρόνο μηδέν, όπου η γεωμετρική τοποθέτηση των κελιών είναι άσχετη.

Αρκετά διαφωτιστικό είναι να μελετήσουμε ένα πρόγραμμα λογιστικών φύλλων από την άποψη του συγγραφέα του. Παρατηρώντας κάθε κελί, θα δούμε την τιμή του και (αν έχει προσδιορισθεί) τον τύπο να εμφανίζεται στη γραμμή εργαλείων της συνάρτησης. Οι σταθερές αξίες και οι τύποι έχουν μια αόρατη αλληλεξάρτηση. Επιλέγοντας ένα από αυτά τα κελιά, το Γραφικό Σύστημα Επικοινωνίας (GUI) εμφανίζει τις «πρώτου επιπέδου»- εξαρτήσεις, και τις διευθύνσεις των πηγών τους. Ένα τέτοιο παράδειγμα βλέπουμε στην εικόνα 2.



Εικόνα 2

Τα αποτελέσματα των Navarro-Prieto και Canas (1999) δείχνουν ότι οι δημιουργοί λογιστικών φύλλων έχουν αναπτύξει καλές διανοητικές δομές για τη ροή δεδομένων. Ο Tukianen (2001), ωστόσο, υποδεικνύει ότι είναι αναγκαίο να απομνημονεύει κανείς (αόρατες) συνοχές σε λογιστικά φύλλα χωρίς να υπάρχει σαφής αναπαράσταση. Εμείς προβλέπουμε ότι αυτό γίνεται συνεχώς πιο δύσκολο με την ανάπτυξη του μεγέθους του φύλλου και θα έχει συγκεκριμένες επιρροές όταν οι συναρτήσεις είναι εκτός του ορατού παράθυρου της οθόνης. Η κατάσταση επιδεινώνεται όταν επιστρέψουμε μετά από καιρό στο φύλλο για τη συντήρησή του ή αν χρειασθεί να το επεξεργασθεί τρίτο πρόσωπο. Σε αυτές τις περιπτώσεις οι θεωρίες Navarro-Prieto και Canas είναι αβάσιμες.

Ένα άλλο θέμα είναι ότι τα λογιστικά φύλλα μπορεί να παρουσιάζουν μια εννοιολογική ανομοιοτυπία μεταξύ της εικονιζόμενης τιμής και του κρυμμένου προτύπου του υπολογισμού. Ο χρήστης «δεν βλέπει» τις εξαρτήσεις των δεδομένων: είναι κρυμμένες πίσω από τιμές (παρανοημένες στατικές). Εάν ένας προγραμματιστής λογιστικού φύλλου χτίζει ένα εννοιολογικό πρότυπο του λογιστικού φύλλου, θα πρέπει να επισκεφθεί κάθε κελί που περιέχει δεδομένα (είτε με το χέρι είτε με τη βοήθεια κάποιου εργαλείου). Κάθε τέτοια επίσκεψη διευρύνει το εννοιολογικό πρότυπο του χρήστη, έτσι ώστε μόνο το «τελικό» (το πιο πρόσφατο) πρότυπο αντιστοιχεί στην πραγματική ροή δεδομένων του προγράμματος.

2.4 Χαρακτηριστικά του προγραμματισμού των λογιστικών φύλλων

Η σημασιολογία της ροής δεδομένων φτάνει στα όρια της σε κρίσιμες καταστάσεις, αν και αρκετά ενορατική. Όπως δηλώνει ο Clermont (2002), τα προγράμματα λογιστικών φύλλων «μοιράζονται πολλά χαρακτηριστικά με τις έννοιες περί ροής δεδομένων», αλλά « μερικές από τις βασικές έννοιες, όπως η κατανάλωση σημείων» δεν αποτελούν μέρος του παραδείγματος λογιστικού φύλλου. Εξετάζοντας αυτή τη δήλωση, καταλαβαίνουμε ότι η σημασιολογία τοων λογιστικών φύλλων είναι εξαρτώμενη από τα εργαλεία και καθιερωμένα πρότυπα τα οποία περιγράφουν σημασιολογία γλώσσας προγραμματισμού δεν ισχύει απόλυτα και ως εκ τούτου η ευρέως κοινή ερμηνεία των λογιστικών φύλλων ως προγραμμάτων ροής δεδομένων δεν έχει υπόσταση.

2.4.1. Στρατηγικές Εκτίμησης

Μια καθοριστική διαφορά μεταξύ ενός συμβατικού (συναρτησιακού) προγράμματος και ενός προγράμματος λογιστικών φύλλων είναι ο χρόνος και η διαδικασία που απαιτούνται για την εκτίμηση. Συμβατικά προγράμματα είναι πλήρως προσδιορισμένα πριν εκτιμηθούν ενώ η εκτίμηση ενός λογιστικού φύλλου είναι διαδικασία την χαρακτηρίζει η έντονη αδημονία. Ένας τύπος εκτιμάται το συντομότερο δυνατόν [Clermont, 2002] οποιαδήποτε στιγμή κατά την διαδικασία της ανάπτυξης του προγράμματος. Κατά συνέπεια το τέλος της διαδικασίας δεν διευκρινίζεται ποτέ στο σύστημα. Τελειώνει μόνο όταν ο χρήστης είναι ικανοποιημένος με το αποτέλεσμα του υπολογισμού. Μέχρι τώρα ήμασταν ασαφείς αναφερόμενοι αδιακρίτως στο συναρτησιακό πρότυπο και το παράδειγμα ροής δεδομένων. Επειδή όμως εξετάζουμε το θέμα της εκτίμησης, πρέπει να αναγνωρίσουμε ότι αυτές οι έννοιες διαφέρουν στην εκτίμηση ως προς την διάταξη και τις έννοιές τους.

Σημασιολογία Ροής Δεδομένων

Τα προγράμματα ροής δεδομένων (Data Flow Programs-DFP) όπως και τα προγράμματα για λογιστικά φύλλα εκτελούν χωρίς την προδιαγραφή της αντίληψης του ελέγχου. Στα Προγράμματα Ροής Δεδομένων, τα δεδομένα ρυθμίζουν τη σειρά της εκτίμησης των

τύπων. Καθώς τα προγράμματα ροής δεδομένων συνήθως αποτυπώνονται σε ένα γράφημα ροής δεδομένων, μπορεί κανείς να διανοηθεί την εκτίμηση ενός σημείου μόλις εισαχθούν τα δεδομένα σε όλες τις περιμέτρους του, δηλαδή όλη η απαιτούμενη πληροφόρηση έχει δοθεί. Τα αναπαραγόμενα δεδομένα είναι ένα σύμβολο που περιέχει το αποτέλεσμα του υπολογισμού του σημείου που τοποθετείται προς τα πάνω. Αυτή η έννοια εκτίμησης είναι, εν τούτοις, βασισμένη στην κατανάλωση συμβόλων. Επανεκτίμηση του προγράμματος ροής δεδομένων απαιτεί επαναυπολογισμό όλων των συμβόλων.

Στη σημασιολογία των λογιστικών φύλλων δεν υπάρχει κάποια «τιμή» η οποία να ορίζει τα όρια του Γραφήματος Εξάρτησης Δεδομένων (Data Dependence Graph - DDG) παρά μόνο ορίζει την «αλλαγή». Ο Yoder Cohn (2002) επίσης αναφέρετε στη σημαντική διαφορά μεταξύ προγραμμάτων ροής δεδομένων και λογιστικών φύλλων: σε ένα πρόγραμμα ροής δεδομένων, ένα κελί επανεκτιμάται , εάν όλες οι πηγές του έχουν (νέες) τιμές προς επεξεργασία. Στα λογιστικά φύλλα, όμως, μία μόνο επανεκτίμηση είναι αρκετή να προκαλέσει επανεκτίμηση του συνόλου. Ως εκ τούτου, τα προγράμματα ροής δεδομένων (Data Flow Programs-DFP) διαφέρουν από τα προγράμματα λογιστικών φύλλων. Επιπλέον, για την αντιμετώπιση των βρόχων τα προγράμματα ροής δεδομένων (DFPs) περιέχουν και σημεία βρόχων ως ιδιαίτερες έννοιες. Αλλά οι βρόχοι δεν είναι χαρακτηριστικό των προγραμμάτων λογιστικών φύλλων.[1]

Σημασιολογία Μειουμένου Γραφήματος

Η αντίληψη του ελέγχου είναι πιο συναφής στα συναρτησιακά προγράμματα και την σημασιολογία του μειουμένου γραφήματος. (Sestoft, 2001), (Dermoudy, 2003). Εδώ κάθε τύπος ερμηνεύεται ως συναρτησιακή δήλωση. Η σημασιολογία του μειουμένου γραφήματος υπονοεί ότι ένας τύπος και τα ορίσματά του αντικαθίστανται απο το αποτέλεσμα της εφαρμογής του τύπου. Εφόσον το αποτέλεσμα εφαρμογής ενός τύπου μπορεί να χρησιμοποιηθεί περισσότερο απο μια φορά εντός του προγράμματος, η μείωση πρέπει να επαναληφθεί για κάθε περίπτωση. (Clermont, 2002).

Σύμφωνα με τον Clermont (2002), οι εκτιμήσεις στα προγράμματα προγραμμάτων λογιστικών φύλλων ακλουθούν τις αρχές των μειούμενων γραφημάτων. Εν τούτοις, υπάρχουν δύο κύρια επιχειρήματα που

[1] Το πρόγραμμα EXCEL επιτρέπει αναδρομές με περιορισμένες επαναλήψεις. Αυτό όμως δεν είναι καθιερωμένο χαρακτηριστικό των λογιστικών φύλλων και έχει επιπλέον εννοιολογικούς περιορισμούς.

δείχνουν ότι τα προγράμματα λογιστικών φύλλων δεν είναι καθαρά προγράμματα μειούμενων γραφημάτων: οι βρόχοι και η διάδοση αλλαγών.

1) Αναδρομή και Βρόχοι

Το παράδειγμα του συναρτησιακού προγράμματος δεν περιλαμβάνει βρόχους, αλλά η αναδρομή είναι κύρια έννοια σε αυτά τα προγράμματα. Αντιθέτως, η αναδρομή δεν είναι χαρακτηριστικό του παραδείγματος λογιστικών φύλλων διότι η αναδρομή εμποδίζει την παρουσίαση ενδιάμεσων αποτελεσμάτων και προϋποθέτει εγγενώς τη παροχή καθολικού ελέγχου ροής.

2) Διάδοση Αλλαγών

Η διαδραστική ιδιότητα των προγραμμάτων λογιστικών φύλλων οδηγεί σε μία περίπλοκη τεχνική διάδοσης αλλαγών [Clermont, 2002] [Yoder Cohn, 2002]. Εάν γίνει μία αλλαγή σε κάποιο κελί, για να διατηρηθεί η συνάφεια συμβαίνουν τα εξής τρία πράγματα:

- Η τιμή του τύπου επανεκτιμάται,
- Εξαρτημένοι τύποι πρέπει να επανεκτιμηθούν, και
- Τύποι εντός του μεταβατικού εγκλεισμού πρέπει να επανεκτιμηθούν.

Η διαδικασία της επανεκτίμησης καθοδηγείται κυρίως από τη ροή δεδομένων. Οι [Burnet κ.α.,2001] αναφέρουν ότι ο όρος «συνεχής εκτίμηση» είναι επινοητός, και εννοεί ότι τα κελιά αναφέρουν τις πραγματικές τιμές τους όταν εμφανίζονται.

2.4.2. Ασυνάρτητες Στρατηγικές Εκτίμησης

Βάση του Clermont (2002) τα προγράμματα λογιστικών φύλλων πρέπει να θεωρηθούν μερικώς ως προγράμματα μειούμενων γραφημάτων και μερικώς ως προγράμματα ροής δεδομένων. Ποιό από τα δύο ισχύει και πότε έχει ιδιαίτερο ενδιαφέρον κυρίως κατά τη συντήρηση του λογιστικού φύλλου. Η διάκριση πρέπει να γίνει ανάμεσα σε τοπικής και καθολικής εκτίμησης :

- Τοπική εκτίμηση

Η εκτίμηση της τιμής του κελιού (ξεκινώντας από τον τύπο του) βασίζεται σε μειούμενο γράφημα το οποίο βασίζεται στο Γράφημα Εξάρτησης Δεδομένων (Data Dependence Graph - DDG) του λογιστικού

φύλλου.

- Καθολική εκτίμηση

Αλλαγές σε όλα τα κελιά τα οποία εξαρτώνται από τις αλλαγές στην τιμή ενός κελιού διαδίδονται με σύμβολα αλλαγής μέσω του γραφήματος ροής δεδομένων και είναι παρόμοια με το Γράφημα Εξάρτησης Δεδομένων (Data Dependence Graph - DDG) του λογιστικού φύλλου.

Συνεπώς, τα λογιστικά φύλλα ενσωματώνουν και τις δύο έννοιες ανάλογα με τη άποψη.

Αυτό που αντιπαρατίθεται στην προηγούμενη έννοια είναι η στρατηγική εκτίμησης. Η διαδραστικότητα απαιτεί η ολική εκτίμηση του λογιστικού φύλλου να είναι ενεργή. Στα συναρτησιακά προγράμματα μία τιμή υπολογίζεται όταν ζητηθεί. Έτσι, η νωθρή εκτίμηση είναι μέρος της εκτίμηση τύπου ειδικά για κελιά. Για παράδειγμα, το Excel εκτιμά κατ' αυτό το τρόπο την IF-συνιστώσα, αρχικά εκτιμάται η συνθήκη και τότε μόνο το επιλεγμένο εναλλακτικό εκτιμάται.

Υπάρχει και άλλη μία διαφορά μεταξύ συναρτησιακών προγραμμάτων και προγραμμάτων λογιστικών φύλλων. Από τη φύση του παραδείγματος του λογιστικού φύλλου, κάθε κελί του πρέπει να θεωρείται αποτέλεσμα, ενώ το συναρτησιακό πρόγραμμα έχει ακριβώς ένα σαφές αποτέλεσμα. Ο Yoder Cohn (1994), παρουσιάζει μία προσέγγιση βασισμένη σε κατά παραγγελία εκτίμηση, όπου μόνο τα κελιά ενδιαφέροντος εκτιμούνται. Ο συγγραφέας όμως βασίζεται στην αρχή ότι «τα κελιά παραμένουν ενημερωμένα».
Πέρα από αυτές τις διαφορές, το πρόγραμμα λογιστικών φύλλων ενώνει τις έννοιες των συναρτήσεων και της ροής δεδομένων. Όπως έχει ήδη αναφερθεί, η φυσική βάση της γλώσσας των συναρτήσεων είναι γραφήματα ροής δεδομένων, όπου οι συναρτήσεις είναι σημεία και οι ακμές αναπαριστούν τις εξαρτήσεις των δεδομένων.

2.5. *Απροσάρμοστα Στοιχεία*

Όλα τα στοιχεία της γλώσσας γενικής χρήσης δεν προσαρμόζονται απόλυτα στη μεταφορά των λογιστικών φύλλων. Η αναδρομή και η ανάθεση με επιπλοκές είναι τα κυριότερα από αυτά.

2.5.1. Αναδρομή

Είναι αρκετά εύκολο να εισάγουμε αναδρομή στη γλώσσα ενός κελιού, αλλά είναι δύσκολο να την αναπαραστήσουμε σε σχέσεις μεταξύ κελιών. Η δυσκολία έγκειται στην ιδιότητα του λογιστικού φύλλου να παρέχει άμεσα αποτελέσματα. Σε έναν επαναληπτικό βρόχο ο χρήστης μπορεί να παρατηρεί αλλαγές στις τιμές ενός κελιού καθώς ο βρόχος εξελίσσεται. Αυτό είναι ιδιαίτερα εμφανές όταν το σύστημα του λογιστικό φύλλο παρέχει διαδοχική εκτέλεση.

Ένα πρόγραμμα αναδρομής, όμως, λειτουργεί δημιουργώντας μία στοίβα νέων τιμών. Η ορατή αναπαράσταση αυτής της διαρκώς αυξανόμενης ή μειωνόμενης στοίβας, με τον ίδιο διαισθητικό τρόπο αναπαράστασης των άλλων τιμών του λογιστικού φύλου, παραμένει ένα άλυτο πρόβλημα. Ειδικά η αναπαράσταση της ροής των τιμών από ένα επίπεδο της στοίβας σε άλλο είναι ιδιαίτερα δύσκολη.

Επειδή ο σκοπός μας είναι να χαρτογραφήσουμε όλες τις έννοιες και δομές γλωσσών γενικής χρήσης στα λογιστικά φύλλα, πρέπει να συμπεριληφθούν και οι αναδρομές. Αλλά δεδομένου ότι η επανάληψη έχει προνοήσει, και οι δομές αναδρομικών προγραμμάτων μπορούν να εκτεθούν από επαναληπτικά προγράμματα, η έλλειψη μιας κατάλληλης αναπαράστασης για την αναδρομή δεν είναι σοβαρή.

2.5.2. Αναθέσεις με επιπλοκές

Στις γλώσσες γενικής χρήσης οι αναθέσεις με επιπλοκές είναι ένα πολύ κοινό χαρακτηριστικό.
Αν και η κοινότητα των συναρτησιακών γλωσσών δεν τις θεωρεί απαραίτητες, παραμένουν όμως ένας φυσικός τρόπος σκέψης των ανθρώπων, και χρήσιμος τρόπος για τη διαμόρφωση γεγονότων στον πραγματικό κόσμο.

Όταν μία σφαίρα σπάει ένα μπουκάλι, για παράδειγμα, δεν θεωρούμε ότι το μπουκάλι έσπασε τον εαυτό του σε απάντηση του ερχομού της σφαίρας. Αλλά αυτός είναι ο τρόπος με τον οποίο οι συναρτησιακές γλώσσες μας θέλουν να σκεπτόμαστε. Η παρουσίαση αυτού του συμβάντος με μία συναρτησιακή γλώσσα προϋποθέτει ότι τι μπουκάλι έχει μία συλλογή δυνάμεων οι οποίες ενεργοποιούνται με την προσέγγιση της σφαίρας με τέτοιο τρόπο ώστε να διαλύουν το μπουκάλι. Αυτό όμως είναι κίνηση οπισθοδρομική. Είναι αρκετά σαφές, ότι ένας εξωτερικός παράγοντας κατάστρεψε το μπουκάλι., μία δύναμη εφαρμόστηκε σε ένα σημείο της επιφάνειάς του, και η διάδοση αυτής της δύναμης έσπασε το μπουκάλι. Οι γλώσσες που

περιέχουν αναθέσεις με επιπλοκές επιτρέπουν την έκφραση της φυσικής διαίσθησης με φυσικό τρόπο.

Οι αναθέσεις με περιπλοκές μπορούν να προστεθούν στα λογιστικά φύλλα, αλλά με περιορισμούς. Επιτρέπουμε σε κελιά να προσδιορίσουν τιμές άλλων κελιών, θέτοντας όμως μία σειρά περιορισμών πρόσβασης. Επιπλέον, θέτοντας την τιμή ενός κελιού διαγράφεται ο τύπος που τυχόν περιείχε. Η λειτουργία της ανάθεσης προκαλεί μία Τρίτη μορφή εξαρτήσεων στο Γράφημα Εξαρτήσεων (DG). Εκτός των κανονικών εξαρτήσεων και των εξαρτήσεων αντιγραφής, προσθέτουμε και γραπτές εξαρτήσεις.

Αυτό έχει διπλό κόστος. Κατ' αρχάς δυσκολεύει την ανάλυση του Γραφήματος Εξαρτήσεων. Χωρίς τις γραπτές εξαρτήσεις, μπορεί να αποδειχθεί ότι προγράμματα που δεν περιέχουν εξαρτήσεις και συναντιούνται πριν από ένα κύκλο στο Γράφημα Εξαρτήσεων είναι πάντα καθοριστικά, και ακόμη και αυτά που τηρούν αυτή τη προϋπόθεση μπορεί να υπολογισθούν με τρόπους που τα καθιστά καθοριστικά. Αλλά με τη γραπτή εξάρτηση δεν υπάρχει αυτή η διαβεβαίωση, οι γραπτές εξαρτήσεις προσφέρουν πολλές ευκαιρίες αδυναμίας καθορισμού οι οποίες δυσκολεύουν τη δημιουργία προγραμμάτων τα οποία δεν εκπίπτουν στο χώρο του ακαθόριστου.

Το δεύτερο κόστος είναι ψυχολογικό. Η ανάθεση με επιπλοκές αφαιρεί από τα λογιστικά φύλλα το δηλωτικό τους χαρακτήρα ο οποίος είναι ένας κύριο χαρακτηριστικό της δημοτικότητάς τους. Αν και τα συμβατικά λογιστικά φύλλα χρησιμοποιούν την ανάθεση με περιορισμένους τρόπους, σε «μακροεντολές», η γενίκευση αυτής της πρακτικής σε όλους τους τύπους του λογιστικού φύλλου προκαλεί σύγχυση καθότι η έλλειψη καθορισμού θα πολλαπλασιασθεί σε όλο το λογιστικό φύλλο.

Παρόλο το κόστος, η εύρεση ενός «σωστού» τρόπου να προστεθεί η αναφορά στα λογιστικά φύλλα θα αποτελούσε θεμιτό στόχο για τον σχεδιαστή γλώσσας λογιστικών φύλλων. Λειτουργίες συστοιχίας, για παράδειγμα, αποτελούν καλό τρόπο χρήσης των αναθέσεων με επιπλοκές. Ας υποθέσουμε μία λειτουργία συστοιχίας, οπού τα κελιά Α, Β, Γ, περιέχουν το καθένα μπλοκ άλλων κελιών. Θέλουμε να προσδιορίσουμε , σε ένα τύπο ενός κελιού στο Γ, τα αποτελέσματα αυτής της λειτουργίας. Αυτό είναι εύκολο με τη χρήση της σχετικής διοχέτευσης. Αλλά αυτό σημαίνει ότι το κελί Γ πρέπει (τουλάχιστον έμμεσα) να προσδιορίσει τις τιμές των παιδιών-κελιών του. Το ίδιο πρόβλημα ισχύει με όλες τις λειτουργίες συστοίχησης. Είναι όμως κρίμα, διότι υπάρχει, στην επιφάνεια, μία

απλή και διαισθητική μέθοδος έκθεσης από τις δομές του APL και της θεωρίας συστοίχησης στη μεταφορά του λογιστικού φύλλου.
Βρισκόμαστε σε αμηχανία. Από μία άποψη, η ανάθεση με επιπλοκές είναι απεχθής. Αλλά από μία άλλη άποψη, μας παρέχει μία ανεκτίμητη περίληψη (λειτουργίες συστοίχισης), και είναι απόλυτα θεμιτή.

Μπορούμε εύκολα να παρακάμψουμε αυτό το πρόβλημα, σε σχέση με τις λειτουργίες συστοίχησης, επιτρέποντας στα κελιά να κληρονομούν τον τύπο τους από άλλα κελιά. Για τις περισσότερες λειτουργίες συστοίχησης είναι αρκετό να επιτραπεί στα κελιά να κληρονομούν μόνον τους τύπους των γονέων τους[2]. Αν πάρουμε το προηγούμενο παράδειγμα των κελιών-μπλοκ Α,Β,Γ, όπου Γ= Α * Β, και θέσουμε στο Γ τον τύπο (* A[-,0] B[0,-]) και προσδιορίσουμε ότι όλα τα κελιά του Γ κληρονομούν τους τύπους τους από το Γ (για τους σκοπούς αυτής της εργασίας παραλείπουμε τον μηχανισμό). Για οποιοδήποτε κελί, ο τύπος πολλαπλασιάζει το στοιχείο της ίδιας γραμμής της στήλης Α, με το στοιχείο της ίδιας στήλης της γραμμής Β. Η λειτουργία συστοίχησης που θέλαμε είναι έτσι στη θέση της.

Μπορεί μία συνάρτηση ή μία μακροεντολή να προσδιορισθεί για να παράγει το εξωτερικό γινόμενο; Ναι, μπορεί. Ας υποθέσουμε ότι το κελί συνάρτησης του εξωτερικού γινόμενου (outer product- OP) περιέχει τον τύπο (defmacro (A B) (* A[-,0] B[0,-]), και το κελί Γ τον τύπο
(OP A B).
Η επιθυμητή επίδραση συμβαίνει, η μακροεντολή OP υπολογίζεται στη θέση της, σε κάθε κελί του Γ, ώστε η σχετική διοχέτευση του αρχικού τύπου διατηρείται.
Μπορεί να υπάρχουν και άλλοι λόγοι για τη χρήση ανάθεσης με επιπλοκές στη γλώσσα του λογιστικού φύλλου. Μπορεί, παραδείγματος χάριν, να θέλουμε αναστοιχειοθετήσουμε τον δείκτη ενός βρόχου, σε ένα πρόγραμμα βρόχων. Ωστόσο, τα διαθέσιμα στοιχεία προτείνουν ότι η προσθήκη ανάθεσης πρέπει να γίνεται προσεκτικά και με μέτρο, αν είναι απαραίτητο, και μάλλον εναλλακτικές δομές θα ήταν καλύτερο να χρησιμοποιηθούν, εφόσον αυτές δεν κρύβουν τη δράση του προγράμματος περισσότερο από τις αναθέσεις.

[2] Η επίτρεψη άλλων κελιών να αποτελούν την προέλευση ενός τύπου μεταλλάσσει την γλώσσα των λογιστικών φύλλων προς την έννοια της εκπροσώπησης. Επίσης διευρύνει τη λειτουργία προς ένα σχήμα εκπροσώπησης πολλαπλών επιπέδων, όπου το κελί κληρονομεί κατ' αρχάς από την γραμμή του, ή αν αυτή είναι κενή, από την στήλη του, ή τελικά από το μπλοκ των γονέων του. Η ιδέα είχε προταθεί από τον Bill Wadge υπό άλλη έννοια.

2.6. *Διοχέτευση*

Οι γλώσσες γενικής -χρήσης που προσφέρουν συστοιχίες επίσης προσφέρουν και τρόπους για τη διοχέτευση διαφόρων στοιχείων εντός της συστοιχίας. Τα λογιστικά φύλλα, όμως, παραθέτουν νέες μορφές διοχέτευσης που δεν υπάρχουν σε σχεδόν καμία από τις επικρατούσες γλώσσες γενικής - χρήσης. Η διοχέτευση στα λογιστικά φύλλα, συμβατικά και αυτά με τάσεις GSM, μπορεί να ταξινομηθεί σε τρεις κατηγορίες: *απόλυτη, σχετική και περιοχής*. Επίσης, όταν εμφωλευμένα μπλοκ εισάγονται, περισσότερη ελευθερία παρέχεται με άμεσες και έμμεσες μορφές διοχέτευσης. Αυτές αντιστοιχούν με τον τρόπο γίνεται χρήση ονομάτων σε αντικειμενοστρεφείς γλώσσες.

Σε αυτήν την ενότητα θα κάνουμε χρήση συμβολισμού με βάση τα μαθηματικά για τις δομές διοχέτευσης, αντί των συμβατικών διοχετεύσεων λογιστικών φύλλων που χρησιμοποιήσαμε μέχρι τώρα. Ελπίζουμε ότι έτσι οι έννοιες θα έχουν περισσότερη διαύγεια για τον αναγνώστη. Η παρουσίαση συμβόλων συμβατικών λογιστικών φύλλων σε μαθηματικά σύμβολα που θα χρησιμοποιήσουμε εδώ δεν είναι δύσκολη. Επειδή δεν θα μας διαφώτιζαν περαιτέρω, δεν θα αναφερθούμε σε αυτά.

2.6.1. Απόλυτη διοχέτευση

Ας υποθέσουμε ότι το κελί Α περιέχει ένα μπλοκ κελιών, αν προσδιορίσουμε συγκεκριμένα κελιά του μπλοκ, π.χ. Α[3,2] και Α[2,5], αυτό αποτελεί απόλυτη διοχέτευση, και είναι ακριβώς αντίστοιχη με τη χρήση της σε γλώσσες γενικής χρήσης.

2.6.2. Σχετική Διοχέτευση

Η σχετική διοχέτευση είναι τελείως διαφορετικό θέμα. Η σχετική μορφή διοχέτευσης είναι πολύ συνήθης στα λογιστικά φύλλα. Η κανονική διαδικασία, όταν αναπτύσσεται ένα λογιστικό

φύλλο, είναι η προσεκτική κωδικοποίηση μερικών αντιπροσωπευτικών κελιών με τύπους, και μετά η αντιγραφή και επικόλλησή τους σε άλλα κελιά για την συμπλήρωση του λογιστικού φύλλου. Για παράδειγμα, ένα λογιστικό φύλλο για τον υπολογισμό των βαθμών μαθητών, είναι συνήθως δομημένο ώστε το αρχείο κάθε μαθητή είναι σε μία μόνο γραμμή. Οι βαθμοί του μαθητή βρίσκονται σε διάφορα κελιά της ίδιας γραμμής, και κάπου στα δεξιά αυτών, υπάρχουν ένα ή περισσότερα κελιά στα οποία γίνονται οι υπολογισμοί του γενικού βαθμού. Η σωστή κατασκευή ενός τέτοιου βιβλίου εργασίας προϋποθέτει ότι όλοι οι απαραίτητοι τύποι είναι σωστοί για ένα μαθητή, μετά η αντιγραφή και η επικόλλησή τους σε αντίστοιχες θέσεις στις σειρές άλλων μαθητών. Η σχετική διοχέτευση φροντίζει για τα υπόλοιπα.

2.6.3 Διοχέτευση περιοχής

Επίσης πολύ κοινή μέθοδος διοχέτευσης των λογιστικών φύλλων είναι και η διοχέτευση περιοχής. Η διοχέτευση περιοχής προσδιορίζει περιοχές, δηλαδή υπό-μπλοκ, και είναι ιδιαίτερα χρήσιμη σε συνδυασμό με λειτουργίες συστοίχισης, όπως αυτές που αναφέραμε στην παράγραφο 3.2., εάν παρέχονται.

2.6.4. Άμεση και έμμεση διοχέτευση

Ένα λογιστικό φύλλο που περιέχει εμφωλευμένα μπλοκ μπορεί να ωφεληθεί από την πρόσθεση έμμεσης διοχέτευσης.
Η διαφορά μεταξύ άμεσης και έμμεσης διοχέτευσης έχει την ίδια σχέση με την εμφώλευση μπλοκ, όπως η απόλυτη και η σχετική διοχέτευση έχουν με την παράταξη του μπλοκ. Δηλαδή, η έμμεση διοχέτευση είναι απλώς σχετική διοχέτευση εφαρμοσμένη σε επίπεδο περιστολής. Μπορούμε να πούμε ότι η έμμεση διοχέτευση είναι ίδια με τη *σχετική προοπτική*.

Αντικειμενοστραφής γλώσσες (Object-oriented -OO) χρησιμοποιούν κάτι που μοιάζει με σχετική προοπτική, οι μέθοδοι εντός ενός αντικειμένου δεν απαιτούν την δήλωση των ονομάτων των στιγμιαίων μεταβλητών του αντικειμένου. Αλλά μία γλώσσα λογιστικών φύλλων περιέχει πολύ περισσότερα από τη λειτουργία της σχετικής προοπτικής. Οι τύποι εύκολα αντιγράφονται και επικολλώνται από το ένα μπλοκ σε άλλο, αλλά οι μέθοδοι σπάνια έχουν αυτή την ιδιότητα από μία τάξη σε άλλη. Ακόμη και αν αυτό

γίνει, παραδείγματος χάριν με μηχανισμό κληρονομιάς, τα αποτελέσματα δεν είναι τα ίδια, διότι οι αντικειμενοστραφής γλώσσες επιτρέπουν πρόσβαση στα δεδομένα μόνο ονομαστικά και όχι με βάση τη θέση τους.

Η γεωμετρική φύση των υπολογισμών με λογιστικό φύλλο είναι σε θεμελιώδη αντίθεση με την εξ ολοκλήρου αλγεβρική φύση των σύγχρονων γλωσσών προγραμματισμού. Αυτό το χαρακτηριστικό είναι που προσδίδει στην έμμεση και σχετική διοχέτευση τον διακριτικό τους χαρακτήρα. Το κελί ενός λογιστικού φύλλου μπορεί να αναφερθεί σε άλλο κελί δύο κελιά δεξιά του, ή να χρησιμοποιήσει τη θέση στο μπλοκ των γονέων του για να διοχετεύσει κάποιο άλλο κελί σε ένα τελείως διαφορετικό μπλοκ. Παρόμοιες λειτουργίες στη γλώσσα της άλγεβρας είναι κατανοητές μόνο εντός των πλαισίων της στοίχισης. Διαφορετικά αναγκαζόμαστε να κάνουμε υποθέσεις ως προς την παράταξη των αντικειμένων στη μνήμη, οι αλγεβρικές γλώσσες είναι συνήθως σχεδιασμένες να κρύβουν αυτό το επίπεδο λεπτομέρειας.

2.6.5. Διοχέτευση και οι γλώσσες γενικής χρήσης

Οι προγραμματιστές γλωσσών γενικής χρήσης δεν γνωρίζουν την ποικιλία μορφών διοχέτευσης που παρέχουν οι γλώσσες των λογιστικών φύλλων. Και είναι αμφίβολο αν θα έπρεπε να γνωρίζουν. Σε τελευταία ανάλυση, οι μορφές διοχέτευσης στα λογιστικά φύλλα προκύπτουν από την πινακοειδή παράταξη που επιβάλλει η μεταφορά, οι συμβατικές γλώσσες δεν έχουν αυτόν τον περιορισμό.

Σε περίπτωση που κάποιος θα ήθελε να προσθέσει, παραδείγματος χάριν, σχετική διοχέτευση σε μία συμβατική γλώσσα, ποια θα ήταν η σωστή προσέγγιση; Η σχετική διοχέτευση δύναται να εφαρμοσθεί μόνο όταν η γλώσσα παρέχει κάποια έννοια πλαισίου, και οι γλώσσες γενικής χρήσης δεν έχουν αυτό το χαρακτηριστικό, τουλάχιστον από την γεωμετρική έννοια.

Ίσως είναι δυνατόν να εμπνευστούμε από τη Lucid [AFJ 91], μία εννοιολογική γλώσσα ροής δεδομένων, η οποία παρέχει την έννοια του πλαισίου. Στην Lucid, τα μεταβλητά αντιπροσωπεύουν άπειρες χρονικές ακολουθίες τιμών. Αυτές οι ακολουθίες αντιπροσωπεύουν την έννοια, την ένωση όλων των πιθανών πλαισίων για το μεταβλητό. Οι ακολουθίες είναι δομημένες από τελεστές, όπως, f επί («ακολουθούμενος από »)

Ζυγοί = 2 f επί ζυγοί +2
το οποίο δομεί την άπειρη ακολουθία που περιέχει όλους τους θετικούς ζυγούς αριθμούς. Μία ακολουθία είναι εννοιολογική διότι το αποτέλεσμα μίας έκφρασης (η επέκταση), ποικίλλει ως προς την διάσταση του χρόνου εντός του ευρετηρίου. Αυτό, όμως, δεν είναι διαφορετικό από το να λέμε ότι η συστοίχηση σε ένα συμβατικό πρόγραμμα είναι εννοιολογική επειδή το αποτέλεσμα μιας λειτουργίας ευρετηρίου ποικίλει ανάλογα με την τιμή του ευρετηρίου. Πολυδιάστατες παραλλαγές του Lucid επιτρέπουν πολυδιάστατες ακολουθίες.

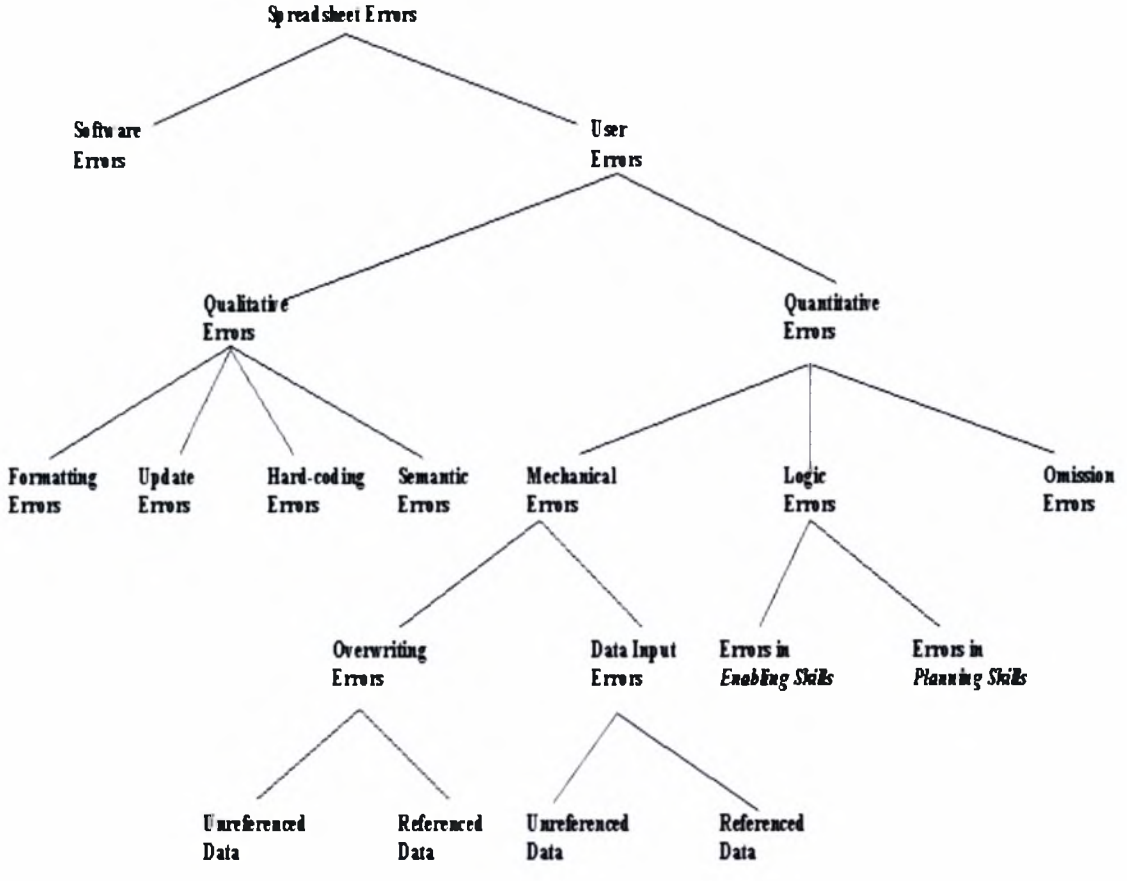
Άλλες γλώσσες που παρέχουν νωθρούς υπολογισμούς, όπως η SASL [Kam 90], έχουν την δυνατότητα να δημιουργούν άπειρες ακολουθίες χωρίς την έννοια του πλαισίου. Μία ακολουθία υπολογίζεται στο βαθμό που απαιτείται, το υπόλοιπο της ακολουθίας αντιπροσωπεύεται από έναν εγκλεισμό, ο οποίος περιέχει τη τελευταία γνωστή τιμή και μία συνάρτηση για τον υπολογισμό της επόμενης.

Η Plane Lucid [DW 90] είναι μία παραλλαγή της Lucid με επιπρόσθετα πρωτόγονα για να χειρισθούν στοιχεία εντός του επιπέδου. Αυτά τα πρωτόγονα, όπως, δεξιά, αριστερά, πάνω, κάτω και τα λοιπά, αποδίδει σε αυτή τη «συμβατική» γλώσσα γεωμετρικές έννοιες. Αλλά αυτή η γλώσσα χρησιμοποιήθηκε πρωτίστως για την εφαρμογή των λογιστικών φύλλων.

Παρόλο που η Lucid και οι παραλλαγές της παρουσιάζουν ενδιαφέρον και απονέμουν μία ιδέα πλαισίου σε αλγεβρικά μεταβλητά, είναι δύσκολες για τους περισσότερου χρήστες. Είναι δύσκολο να συνηθίσει κανείς στην ιδέα ότι όλα τα μεταβλητά είναι άπειρες ακολουθίες. Όταν αυτό επεκτείνεται στις πολυδιάστατες περιπτώσεις, γίνεται ακόμη πιο δύσκολο. Αυτό είναι σε έντονη αντίθεση με τις εμπειρίες που έχουν οι χρήστες των λογιστικών φύλλων, τα οποία τα βρίσκουν διαισθητικά και ευκολονόητα.

Το συμπέρασμα φαίνεται να είναι ότι τα λογιστικά φύλλα διαφέρουν με θεμελιώδη τρόπο από τις γλώσσες γενικής χρήσης, και ότι η διαφορά αυτή εδρεύει στην έννοια του γεωμετρικού πλαισίου, το οποίο προκύπτει φυσικά από την γραφική διασύνδεση που προσφέρει στον χρήστη. Επιπλέον, είναι δύσκολο να προστεθούν γεωμετρικά πλαίσια σε αλγεβρικές γλώσσες χωρίς να χρειασθεί να γίνει ορατή η διάταξη της μνήμης του προγράμματος. Οι γλώσσες των λογιστικών φύλλων, όμως, δεν αποκαλύπτουν στον χρήστη τίποτα από την διάταξη της μνήμης.

3. Λάθη και λογιστικά φύλλα



Εικόνα 3: Συγκεντρωτικός πίνακας: κατηγοριοποίηση λαθών

Είναι γενικά αποδεκτό ότι 9 από τα 10 λογιστικά φύλλα που αναπτύσσονται πάσχουν από κάποιο λάθος και πολλές φορές τα αποτελέσματα είναι πολύ σοβαρά:

- Ένα λάθος κατά τη διαδικασία αντιγραφής-επικόλλησης κόστισε στην εταιρία TransAlta 24 εκατομμύρια δολάρια όταν μειοδότησε λιγότερο απ'όσο έπρεπε για ένα συμβόλαιο ηλεκτροδότησης.

- Ένα πρόσημο μείον που έλειπε οδήγησε την Fidelity's Magellan Fund στο να υπερεκτιμήσει προβλεπόμενα έσοδα κατά 2.6 δισεκατομμύρια (!!) δολάρια και έτσι να χάσει ένα πολλά υποσχόμενο μέρισμα.
- Λανθασμένα συνδεδεμένα λογιστικά φύλλα επέτρεψαν απάτη που έφτασε τα 700 εκατομμύρια δολάρια στην Allied Irish Bank.
- Η εισαγωγή ενός νέου φαρμάκου στην αγορά καθυστέρησε αρκετούς μήνες λόγω μιας μακροεντολής που πέρασε χωρίς έλεγχο, και αυτό κόστισε πολλά εκατομμύρια δολάρια στην φαρμακευτική εταιρία.

Έτσι γίνεται εύκολα αντιληπτό ότι πρέπει να επικεντρώσουμε τα διδακτική μέθοδο στην εξάλειψη αυτών των λαθών, αφού πρώτα τα κατανοήσουμε και τα ερμηνεύσουμε.

3.1 Λανθασμένα λογιστικά φύλλα και κατηγορίες σφαλμάτων

Σε αυτή την ενότητα θα περιγράψουμε μία δομή που μας επιτρέπει να ταξινομήσουμε τα λάθη σε σχέση με τις έννοιες των λογιστικών φύλλων. Επίσης θα προσδιορίσουμε τρεις κατηγορίες λαθών οι οποίες συσχετίζονται με φυσικά πεδία, λογικά πεδία ή γενικά λάθη. Μερικά παραδείγματα είναι ενδεικτικά της προέλευσης των σφαλμάτων. Ασφαλώς, όλα τα προβλήματα που φέρνουμε ως παράδειγμα θα μπορούσαν να είχαν επιλυθεί με άλλο τρόπο, χωρίς να παρουσιάζουν λάθη.

Ένα σχέδιο ταξινόμησης πρέπει να απευθύνεται στα πιο συχνά και βασικά λάθη. Επίσης, για την αξιολόγηση της αποτελεσματικότερης πρόληψης λαθών και τεχνικής ανεύρεσής τους, αρκεί η ταξινόμησή τους να είναι ενδεικτική του είδους, της συχνότητας και των πιθανών αιτιών. Όμως, όπως αναφέρει ο Beizer, δεν υπάρχει μία παγκοσμίως αναγνωρισμένη ορθή ταξινόμηση των λαθών. Ένα οποιοδήποτε λάθος μπορεί να περάσει σε διαφορετικές κατηγορίες ανάλογα με την αντίληψη του ελεγκτή και την πηγή του λάθους.

Μερικά σχέδια ταξινόμησης υπάρχουν για λάθη λογιστικών φύλλων. Οι Panko και Halverson παρουσιάζουν μία ταξινόμηση βασισμένη σε τρεις βασικές κατηγορίες σφαλμάτων: μηχανικά λάθη, λάθη λογικής και παραλείψεις. Μηχανικά είναι τα λάθη που οφείλονται σε

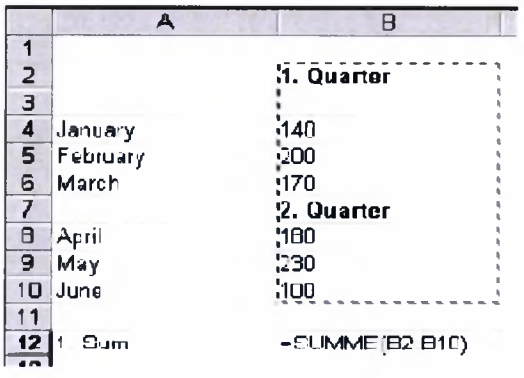
τυπογραφικά λάθη ή λάθος τοποθέτηση. Λογικά είναι τα λάθη που προκύπτουν από λάθος κατανόηση της λογικής του απαιτούμενου αλγόριθμου για τη δημιουργία ενός τύπου. Παραλείψεις σε απαιτούμενα στοιχεία ενός προγράμματος επίσης καταλήγουν σε λάθη. Αυτή η ταξινόμηση είναι βασισμένη στα αίτια των σφαλμάτων. Μία πιο γενική ταξινόμηση, που περιέχει και την ταξινόμηση των Panko και Halverson, δίνει ο Rajalingham κ.α..

Οι Saariluoma κ.α στην πειραματική τους μελέτη, ταξινόμησαν τα λάθη των λογιστικών φύλλων σε δύο βασικές κατηγορίες : Λάθη Τοποθέτησης και Τύπου. Λάθη τοποθέτησης είναι κοινώς γνωστά ως λάθη αναφοράς. Αυτά τα λάθη είναι χαρακτηριστικά των λογιστικών φύλλων. Τα λάθη τύπου περιέχουν τυπογραφικά λάθη στα συστατικά στοιχεία ενός τύπου και αυτό που αποκαλούν μαθηματικά σφάλματα. Μαθηματικά σφάλματα προκύπτουν όταν η απαιτούμενη μαθηματική έκφραση σε ένα τύπο είναι ανεπαρκώς προσδιορισμένη.

Σε αντίθεση με άλλα σχέδια ταξινομήσεων, εμείς δεν θα ταξινομήσουμε τα σφάλματα βάση των αιτιών τους, αλλά βάση της έννοιας του λογιστικού φύλλου με την οποία συσχετίζονται. Σε περαιτέρω κριτήρια μας δεν θα διαφοροποιήσουμε μεταξύ λογικών, μαθηματικών ή τυπογραφικών σφαλμάτων, διότι από το σφάλμα καθαυτό δεν μπορούμε να βρούμε την αιτία του.

Κατηγορία 1η : Σφάλματα φυσικής περιοχής

Τυπικά λάθη των φυσικών περιοχών είναι συνήθως αποτέλεσμα τιμών που δεν έχουν προσδιοριστεί στην περιοχή, ή τιμών λάθος προσδιορισμένων κάπου στη περιοχή. Ονομάζουμε αυτά τα λάθη, *λάθη αναφοράς σε κενό κελί*, ή αντίστοιχα *λάθη αναφοράς σε κελί με τιμή λάθος τύπου*. Σε ορισμένες περιπτώσεις αυτές οι τιμές εισέρχονται με το σκεπτικό να επιτύχουμε μία καλύτερη δόμηση και/ή αναγνωσιμότητα του προγράμματος του λογιστικού φύλλου. Σε άλλες περιπτώσεις οι τιμές αυτές είναι αποτέλεσμα σφάλματος.

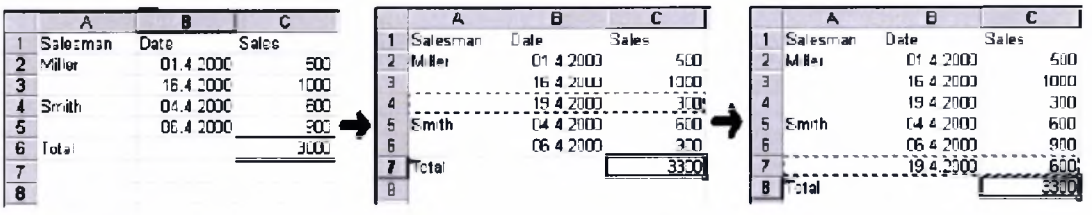


	A	B
1		
2		1. Quarter
3		
4	January	140
5	February	200
6	March	170
7		2. Quarter
8	April	180
9	May	230
10	June	100
11		
12	1. Sum	=SUMME(B2:B10)

Εικόνα 4:Αναφορά σε κενό κελί/λάθος τύπου

Άλλο χαρακτηριστικό πρόβλημα της φυσικής περιοχής είναι η επίδραση στα αποτελέσματα όταν νέες τιμές εισέρχονται στη περιοχή. Εάν η καινούρια τιμή εισαχθεί κάπου στο μέσον της φυσικής περιοχής, αυτόματα η περιοχή διευρύνεται ώστε να συμπεριλαμβάνει την καινούρια και τις παλαιότερες τιμές. Εάν όμως οι καινούργιες τιμές προστεθούν με επισύναψη στη περιοχή , τότε αυτή δεν διευρύνεται. Το αποτέλεσμα είναι *εσφαλμένος προσδιορισμός περιοχής*.

Γενικώς, τα σφάλματα λάθους προσδιορισμού φυσικής περιοχής οφείλονται σε κελιά εκτός περιοχής, τα οποία όμως πρέπει να είναι εντός της περιοχής. Για τον χρήστη δεν είναι τόσο εμφανές ότι αυτά τα κελιά είναι πλέον εκτός περιοχής , και είναι σύνηθες να υποθέσει ότι αυτά τα κελιά επηρεάζουν το αποτέλεσμα της συνάρτησης που έχει εφαρμοστεί στη φυσική περιοχή .



Εικόνα 5:Λάθος προδιαγραφών φυσικής περιοχής

Μία τρίτη κατηγορία χαρακτηριστικών σφαλμάτων είναι η *κατά λάθος διαγραφή ενός κελιού* εντός μίας φυσικής περιοχής . Αποτέλεσμα το ήδη αναφερθέν *λάθος αναφοράς σε κενό κελί*. Επίσης η εισαγωγή ενός στοιχείου που δεν πρέπει να εισαχθεί εντός της περιοχής επιφέρει το ίδιο αποτέλεσμα.

Μία τέταρτη κατηγορία σφαλμάτων είναι το *μπέρδεμα της φυσικής περιοχής*. Ενώ οι προηγούμενες κατηγορίες σφαλμάτων είναι βασισμένες στο ότι οι χρήστες δύσκολα ξεχωρίζουν μεταξύ προγραμμάτων λογιστικών φύλλων και παραδειγμάτων λογιστικών φύλλων, (οι εισαγωγές δεν έχουν τον διακριτό ρόλο που έχουν σε συμβατικά προγράμματα), αυτή η κατηγορία σφαλμάτων είναι αποτέλεσμα των ιδιοτήτων του προγράμματος λογιστικών φύλλων, που είναι ένα μίγμα εργαλείου επίλυσης προβλημάτων και εργαλείου παρουσίασης. Το πρόβλημα παρουσιάζεται όταν δύο διαφορετικές φυσικές περιοχές εμπλέκονται. Σε αυτή τη περίπτωση μία από τις περιοχές δεν μπορεί πια να προσδιοριστεί ως φυσική περιοχή από τον χρήστη. Οι συναρτήσεις ομαδοποίησης πρέπει να αντικατασταθούν από άλλες εκφράσεις (π.χ το SUM - άθροισμα από πολλαπλά σημεία +). Για τον χρήστη δεν είναι εμφανές ότι μπορεί να προσδιορίσει δύο φυσικές περιοχές σε δύο στήλες, αλλά δεν του επιτρέπεται να τις συγχωνεύσει σε μία στήλη, αντίστοιχα το αποτέλεσμα της ομαδοποιημένης συνάρτησης που εφαρμόζεται σε μία από τις φυσικές περιοχές δεν είναι πλέον σωστή.

Κατηγορία 2η : Σφάλματα λογικής περιοχής

Όπως προσδιορίσαμε στην ενότητα 3 μία λογική περιοχή αναφέρεται σε κάποια συνοχή μεταξύ των κελιών της . Συνήθως, μία λογική

περιοχή προέρχεται από την επικόλληση της ίδιας πηγής πολλαπλές φορές και ο χρήστης δεν αντιλαμβάνεται τη λογική περιοχή στην οποία ανήκει το κελί.

Χαρακτηριστικό σφάλμα είναι *η αντικατάσταση μιας συνάρτησης με μία σταθερή τιμή*. Αυτό το σφάλμα μπορεί να έχει πολλές αιτίες, όπως, λάθη σε στρογγύλεμα τιμών ή απρόβλεπτα αποτελέσματα του τύπου. Ο χρήστης απλά αντικαθιστά το αποτέλεσμα της συνάρτησης στο κελί με μία σταθερή τιμή. Σαφώς, αυτή η τιμή παραμένει ακόμη και όταν οι τιμές των πρώην συσχετισμένων κελιών αλλάζουν.

Άλλο χαρακτηριστικό σφάλμα των λογικών περιοχών είναι η *αντιγραφή λάθος αναφοράς*. Σε αυτή την περίπτωση, μία σταθερή τιμή ή μία απόλυτη αναφορά προσδιορίζεται σε ένα τύπο, αντί μιας σχετικής αναφοράς. Αυτό το σφάλμα δεν παρατηρείται άμεσα και έως ο τύπος να αντιγραφεί σε άλλο κελί. Εάν ένα σταθερό κελί συσχετισθεί με μία σχετική αναφορά, το ίδιο πρόβλημα θα παρατηρηθεί, όταν γίνει αντιγραφή του τύπου του κελιού.

Κατηγορία 3η : Γενικά Σφάλματα

Τα γενικά σφάλματα δεν είναι ρητώς συνδεδεμένα με τις φυσικές ή λογικές περιοχές και μόνα μερικά συμβαίνουν κατά την εισαγωγή τιμών στα κελιά. Τα περισσότερα είναι αποτέλεσμα εσφαλμένου προσδιορισμού τύπων. Ένα σφάλμα σχετικό με την εισαγωγή δεδομένων σε κελί είναι μόνο τυπογραφικό λάθος. Εσφαλμένη χρήση μορφοποιήσεων επίσης επηρεάζει τον τρόπο παρουσίασης μιας τιμής. Κάποιος μπορεί να μορφοποιήσει μία τιμή ως 0,2% ενώ εννοούσε 20%. Αυτό μπορεί να συμβεί και σε εισαγόμενα κελιά και σε κελιά τύπου. Επίσης, εάν ένα αριθμητικό δεδομένο μορφοποιηθεί σε δεδομένο ετικέτας, τότε θα επηρεάσει και τον υπολογισμό της τιμής ενός τύπου.

Η άλλη ομάδα γενικών σφαλμάτων γίνεται κατά των προσδιορισμό των τύπων. Όπως αναφέραμε στην ενότητα 3 ο τύπος περιέχει αναφορές κελιών, μαθηματικά σύμβολα, συναρτήσεις και σταθερές τιμές. Κάποιο σφάλμα μπορεί να γίνει με οποιοδήποτε από αυτά τα

στοιχεία, ή από τυπογραφικό λάθος, ή από αδυναμία στη σωστή διατύπωση της απαιτούμενης μαθηματικής έκφρασης. Αυτά τα λάθη περιλαμβάνουν λάθη στη χρήση μαθηματικών συμβόλων, λάθη σε όρια, λάθη παρενθέσεων και λάθη συναρτήσεων.

3.2 Ποσοστά λάθους

Εισαγωγή στα ποσοστά λάθους

Από τις πρώτες ημέρες, έχουν υπάρξει ανησυχίες για τα λάθη από τα αναπτυχθέντα από χρήστες λογιστικά φύλλα. Ο Davis προειδοποίησε ότι *"τα συστήματα μπορούν να είναι επικίνδυνα στην οργάνωσή σας "* γενικά, επειδή οι τελικοί χρήστες εφαρμόζουν σπάνια τις πειθαρχίες που είναι πολύ γνωστές για να είναι απαραίτητοι στην ανάπτυξη συστημάτων που χτίζονται με τον προγραμματισμό. Με τα προγράμματα λογιστικών φύλλων οι τελικοί χρήστες έγιναν ικανοί στις αναλύσεις που περιέχουν χιλιάδες κελιά. Κάτω από αυτούς τους όρους , εκτός αν το ποσοστό των ανακριβών κελιών είναι σχεδόν μηδέν, εκεί θα είναι πολύ υψηλή πιθανότητα λαθών του τελικού αποτελέσματος.

Κατά τη διάρκεια των ετών, διάφορα ενοχλητικά γεγονότα ανάπτυξης λογιστικών φύλλων έχουν αναφερθεί (Panko, 2005b). Στις περισσότερες περιπτώσεις, η κάθε εταιρία που έκανε το λάθος ήταν αναγκασμένη να το ανακαλύψει ή οι σύμβουλοί που είναι εξοικειωμένοι με τα λάθη. Λαμβάνοντας υπόψη την κανονική απροθυμία των οργανώσεων να μιλήσουν για τα λάθη τους, πρέπει να αναρωτηθούμε εάν αυτά τα γεγονότα ήταν σπάνιες περιπτώσεις ή εάν τα λάθη λογιστικών φύλλων είναι μάλλον συχνά μέσα στην πρακτική.

Ευρύτερα, διάφοροι σύμβουλοι, βασισμένοι στην πρακτική εμπειρία, έχουν πει ότι το 20% 40% όλων των λογιστικών φύλλων περιέχουν λάθη (Panko, 2005b). Σε μια προσωπική επικοινωνία με το συντάκτη, Dent (1995) περιέγραψε έναν λογιστικό έλεγχο σε μια επιχείρηση μεταλλείων όπου βρήκε λάθη σε περίπου 30% των ελεγχόμενων λογιστικών φύλλων. Ο Freeman αναφέρει τα στοιχεία από την εμπειρία μιας συμβουλευτικής εταιρίας, Coopers -Lybrand στην Αγγλία, που βρήκε ότι το 90% όλων των λογιστικών φύλλων με περισσότερες από 150 στήλες που έλεγξε περιλάμβαναν λάθη.

Ένας σύμβουλος της Price - Waterhouse έλεγξε τέσσερα μεγάλα φύλλα και βρήκε 128 λάθη (Ditlea, 1987).

Σήμερα, έχουμε κινηθεί πέρα από τέτοια ανεκδοτικά στοιχεία, στη σφαίρα συστηματικού λογιστικού ελέγχου τομέων και εργαστηριακών πειραμάτων. Ο πίνακας 1 συνοψίζει τα βασικά στοιχεία από αυτές τις μελέτες. Περιέχει τα στοιχεία από 13 λογιστικούς ελέγχους που περιλαμβάνουν ρεαλιστικά λογιστικά φύλλα. Από το 1995, όταν άρχισαν να χρησιμοποιούν καλές (αν και συνήθως όχι εξαιρετικές) μεθοδολογίες, 94% των 88 λογιστικών φύλλων ελεγχόμενων σε 7 μελέτες έχουν λάθη, που απεικονίζουν τη βελτίωση στον έλεγχο των μεθοδολογιών.

Επιπλέον, αρκετές από αυτές τις μελέτες εξέθεσαν μόνο σημαντικά λάθη.

Πίνακας 1

Study	**Sample**	**Study**	**Cell Error Rate (CER)**	**Pct. of Models with Errors**
Field Audits				
Davies & Ikin [1987]	19 operational spreadsheets	14 had qualitative errors. Methodology unspecified.		21%
Butler [1992]	273 operational spreadshjeets audited by 143 United Kingdom tax inspectors in 1992.	Only counted "material" errors. Inspectors used a spreadsheet analysis program designed to identify suspicious		10.7%

		parts of a model. Such programs identify only some errors.		
Cragg & King [1993]	20 operational spreadsheets from 10 firms.	150 to 10,000 cells. Had been in use for median of 6 months		25%
Hicks [1995]	1 module with 19 submodules about to enter operation.	Part of a capital budgeting system for NYNEX. Checked heavily ahead of time. Code inspection by three analysts. Found 45 errors in 3,856 cells.		100% (1)
Coopers & Lybrand [1997]	23 spreadsheets from industry	Spreadsheets off by at least 5%		91%
KPMG [1997]	22 spreadsheets from industry	Spreadsheets containing major errors.		91%
Butler [2000]	7 spreadsheets for tax submissions	Spreadsheets audited with enhanced methodology and software. Single auditor.		86%

Total	367 spreadsheets	Weighted average		24%
Since 1997	54 spreadsheets	Weighted average		91%
Cell Entry Experiments		*Errors counted when made, even if corrected later*		
Olson & Nilsen [1987-1988]	14 experienced Lotus 1-2-3 users.	Filled in formulas in skeleton model. 4 formula cells per person. 56 total.	21% (2)	
Floyd & Pyun [1987]		Reanalyzed Olson & Nilsen [1985, 1987-1988] for errors in text cells.	12.5%	
Lerch [1988]	21 professionals with at least one year of Lotus 1-2-3 experience.	Filled in formulas in template. CER based on formulas only. CER especially high for formulas referring to cells in both different rows and different columns.	11.3% (2)	

Development Experiments		*Errors counted at end of development process*		
Brown & Gould [1987]	9 highly experienced spreadsheet developers	Developed 3 models apiece. All made an error in at least one model. Minimum definition of errors, excluding the omission of requirements.		44%
Brown & Gould [1987]		Broader definition of errors including the omission of requirements.		63%
Hassinen [1988]	92 novice spreadsheet students developed 355 spreadsheets	Paper and pencil exercise. Subjects filled in formulas in a skeleton model containing organization and numbers.	4.3% (2)	55%
Hassinen [1988]	10 novice students developing 48 spreadsheets	Computer exercise for same task.		48%

Janvrin & Morrison [1996, 2000]	61 upper division business and graduate accounting students	Study 1: Developed model with multiple worksheets. Had template with filled-in values. Measured incorrect links between worksheets. Model had 51 links. Students had 16 days to do task. Worked an average of 8.8 hours	7%-14% (3)	84%-95%
Janvrin & Morrison [1996, 2000]	88 senior-level accounting students	Study 2: Developed model with multiple worksheets. 66 links between worksheets. No templates to work from; only 1 check figure. CER is percent of incorrect links between spreadsheets.	8%-17% (3)	
Panko & Halverson [1997]	35 undergraduate business students working alone	Developed pro forma income statement based on the Galumpke task.	5.4%	80%
Panko & Halverson [1997]	40 undergraduate business students	Developed pro forma income statement based on	2.0%	60%

	working in groups of 4	the Galumpke task		
Panko & Sprague [1999]	102 undergraduate business students and 50 MBA students. 17 MBAs had more than 250 hours of experience	Developed a model based on the Wall task designed to be relatively simple and free of domain knowledge. No difference in errors across groups.	2.0%	35%
Teo & Tan [1997]	168 undergraduate business students taking second-year IS course	Developed a model based on the Wall task, then did a what-if analysis	2.0%	42%
author (unpublished)	80 undergraduate business students	Developed spreadsheet working alone or in triad (group of 3). Error rates for individual, triad.	4.6%, 1.0%	86%, 27%
Code Inspection Experiments				
Galletta *et al.* [1993]	30 MBA students and 30 CPA accountants	examined 6 small model with 1 seeded error in	34%-54% (4)	

		each. Subjects with 250 hours or more of spreadsheet experience did not find more errors than those without experience.		
Galletta *et al.* [1997]	113 MBA students	Finding eight seeded errors in a student budgeting model.	45%-55% (4)	
Panko [1999]	33 undergraduate MIS majors	Code inspected 11 spreadsheet models, individually and then in groups. Missed errors for individuals, groups.	40%, 17% (4)	
Panko and Sprague [1998]	23 undergraduate subjects with errors in initial model.	Study described above. Code inspected own models. Fixed 18% of errors. Only 13% of spreadsheets were fixed completely. One was made worse.	81% (4)	

Notes:
(1) Errors per model and percent of models with errors computed on basis of submodules.
(2) Errors per formula cell.
(3) Errors per inter-spreadsheet link
(4) Percent of seeded errors not detected

3.3 Συνέπεια με άλλα στοιχεία ανθρώπινου λάθους

Όταν οι περισσότεροι άνθρωποι εξετάζουν τον παραπάνω πίνακα η πρώτη αντίδρασή τους είναι ότι τέτοια ποσοστά υψηλού λάθους είναι αδύνατον να υπάρχουν.

Στην πραγματικότητα, είναι όχι μόνο δυνατόν αλλά είναι εξ ολοκλήρου σύμφωνο με στοιχεία που αφορούν τα ποσοστά ανθρώπινου λάθους από άλλες περιοχές εργασίας. Η ιστοσελίδα ανθρώπινου λάθους (Panko, 2005[α]) παρουσιάζει τα στοιχεία από διάφορες εμπειρικές μελέτες. Μιλώντας γενικά, όταν οι άνθρωποι κάνουν απλές μηχανικές πράξεις , όπως η δακτυλογράφηση, κάνουν μη ανιχνεύσιμα λάθη στο περίπου 0,5% από το ποσό όλων των ενεργειών. Όταν κάνουν τις πιο σύνθετες λογικές δραστηριότητες, όπως το γράψιμο προγραμμάτων, έχουμε άνοδο ποσοστού λάθους σε περίπου 5%. Αυτοί δεν είναι σκληροί και γρήγοροι αριθμοί, επειδή το πόσο λεπτά το καθορίζει "η δράση" έχει επιπτώσεις στο ποσοστό λάθους. Εντούτοις, οι λογικοί στόχοι που χρησιμοποιήθηκαν σε αυτές τις μελέτες είχαν γενικά το σχεδόν ίδιο πεδίο όπως τη δημιουργία μιας συνάρτησης σε ένα λογιστικά φύλλα.

Το πληρέστερο σύνολο στοιχείων όσον αφορά τα ποσοστά λάθους προέρχεται από τον προγραμματισμό, ο οποίος είναι τουλάχιστον συγγενής της ανάπτυξης λογιστικών φύλλων. Στον προγραμματισμό, πολλές επιχειρήσεις πραγματοποιούν επιθεώρηση στις ενότητες προγράμματος. Στην επιθεώρηση κώδικα, οι ομάδες των επιθεωρητών επιθεωρούν αρχικά κάθε ενότητα χωριστά και έπειτα συναντιούνται ως ομάδα για να ελέγξουν συνολικά την ενότητα πάλι (Fagan, 1976). Προφανώς, υπάρχει μια απαίτηση να αναφερθεί ο αριθμός λαθών που βρίσκονται κατά τη διάρκεια της επιθεώρησης κώδικα. Αυτό έχει οδηγήσει στη δημοσίευση των στοιχείων από κυριολεκτικά χιλιάδες επιθεωρήσεις κώδικα (Panko, 2005α). Το στοιχείο από αυτές τις μελέτες παρουσιάζει ισχυρή σύγκλιση. Η επιθεώρηση κώδικα βρίσκει συνήθως λάθη σε περίπου 5% όλου του προγράμματος αφού οι υπεύθυνοι για την ανάπτυξη έχουν τελειώσει την οικοδόμηση και τον έλεγχο της ενότητας (Panko, 2005α). Ενώ υπάρχει κάποια παραλλαγή από μελέτη σε μελέτη, ένα μεγάλο μέρος αυτής της παραλλαγής εμφανίζεται να κάνει τις διαφορές στον προγραμματισμό της γλώσσας, δυσκολία ενότητας, και, δυστυχώς, στο «άγριο» της ανάπτυξης.

Σημειώστε ότι έχουμε μιλήσει για τα μη διορθωμένα λάθη. Οι άνθρωποι κάνουν πολλά λάθη όταν λειτουργούν, αλλά διορθώνουν

επίσης πολλά. Τα ποσοστά διορθώσεων είναι υψηλά για μηχανικά λάθη, αλλά είναι αρκετά χαμηλότερα για τα λάθη λογικής, και εάν έχουμε μια παράλειψη προκαλούμενη από μια εσφαλμένη διάγνωση της κατάστασης, το ποσοστό διορθώσεων είναι πολύ χαμηλό. Κατά τρόπο ενδιαφέροντα, τα στοιχεία από μια μελέτη από Rabbit και Vyas (Rabbit & Vyas, 1970) προτείνουν ότι στο μεγαλύτερο ποσοστό των λαθών η διόρθωση πραγματοποιείται κάτω από το επίπεδο συνειδητοποίησης, έτσι οι άνθρωποι είναι απληροφόρητοι πόσα λάθη κάνουν και πόσα διορθώνουν.

Υπάρχει ακόμη και μια νεοσύστατη θεωρία γιατί κάνουμε τόσα πολλά λάθη. Ο Reason, 1990 έχει παρουσιάσει το πληρέστερο πλαίσιο γιατί τα ανθρώπινα όντα σφάλλουν, αλλά πολλοί άλλοι συγγραφείς έχουν προσθέσει τις συνεισφορές τους (Baars, 1992).

Γενικά, η νεοσύστατη θεωρία υποστηρίζει ότι τα ανθρώπινα όντα είναι εκπληκτικά γρήγορα και εύκαμπτα (Reason , 1990) και μπορούν να κάνουν ταχυδακτυλουργικά πολλαπλάσιους στόχους και περιορισμούς (Flower & Hayes, 1980).

Εντούτοις, οι ίδιες γνωστικές διαδικασίες που επιτρέπουν να λειτουργήσουν με αυτόν τον τρόπο αναπόφευκτα οδηγούν στα περιστασιακά λάθη. Συνολικά, ο Alexander Pope έγραψε ότι "To err is human"

Σήμερα, μπορούμε και να εξηγήσουμε και να εκτιμήσουμε αυτή την δήλωση.

3.4 Μέτρηση των λαθών

Οι μελέτες λογιστικού ελέγχου τομέων που παρουσιάζονται στον πίνακα 1 εκθέτουν ένα απλό μέτρο των λογιστικών φύλλων -- το ποσοστό των λογιστικών φύλλων που περιέχουν τουλάχιστον ένα σοβαρό λάθος. (τα μικρότερα λάθη έχουν απορριφτεί). Ενώ αυτό είναι μια ζωηρή στατιστική, δεν μας λέει πόσα λάθη περιέχει ένα μέσο λογιστικό φύλλο.

Επιπλέον, ξέρουμε από καιρό ότι στους στόχους που περιέχουν πολλές δευτερεύουσες υποχρεώσεις, τα ποσοστά λάθους θα πολλαπλασιαστούν κατά την εκτέλεση δευτερευουσών υποχρεώσεων. Παραδείγματος χάριν, πολλά τελικά αποτελέσματα υπολογίζονται μέσω των μεγάλων σχηματισμών κελιών αριθμητικού και συναρτησιακού τύπου. Οι Lorge και Solomon (1955) έδωσαν διάφορους τύπους για τον υπολογισμό των ποσοστών λάθους. τελικών

αποτελεσμάτων Η εξίσωση 1 προσαρμόζεται από έναν από τους απλούστερους τύπους τους. Εδώ, το Ε είναι το ποσοστό λάθους τελικού αποτελέσματος, e είναι το ποσοστό λάθους στόχου (που υποτίθεται ότι ήταν ο ίδιος για όλους τους στόχους), και το n είναι αριθμός βημάτων στόχου.

Εξίσωση 1: E = 1 - (1- e)^n

Στον προγραμματισμό, αυτή η σύνδεση μεταξύ των μεμονωμένων ποσοστών λάθους και των ποσοστών λάθους σχηματισμών είναι εξετασμένος με τον υπολογισμό του ποσοστού λάθους ανά γραμμή κώδικα. Πιο συγκεκριμένα, οι προγραμματιστές μετρούν τον αριθμό λαθών ανά χίλιες γραμμές κώδικα πηγής

Αυτή η δήλωση του ποσοστού λάθους είναι κατά προσέγγιση ανεξάρτητη από το μέγεθος προγράμματος (Putnam & Myers, 1992). Αν και είναι σημαντικό να διενεργηθούν οι προσαρμογές για την πολυπλοκότητα, μέγεθος, και άλλα θέματα για τη τέλεια ανάλυση (Ferdinand , 1993), και ενώ θα ήταν καλύτερο να μετρηθούν τα λάθη ανά σημείο λειτουργίας, τα λάθη/KLOC είναι μια καλή μέτρηση των λαθών προγραμματισμού.

Στους υπολογισμούς με λογιστικά φύλλα, παρόμοια είναι μια μέτρηση του ποσοστού λάθους κελιών (CER). Αυτό είναι ο αριθμός των λαθών που διαιρούνται με το συνδυασμένο αριθμό αριθμητικών συναρτησιακών κελιών. (Τα κελιά ετικετών είναι όπως τα κελιά σχολίου, τα οποία αγνοούνται στα λάθη/KLOC). Ισοδύναμα, αυτό είναι ποσοστό των κελιών μη-ετικετών που περιέχουν τα λάθη. Ο πίνακας 1 δείχνει ότι ακριβώς ως τα λάθη /KLOC συνήθως μειώσεις σε μια στενή σειρά (Panko, 2005α), το CER που φαίνεται στις μελέτες λογιστικών φύλλων ήταν πολύ παρόμοιο. Οι σχετικά λίγοι διάφοροι αριθμοί που εμφανίζονται μέσα σε ένα στάδιο κύκλων ζωής, επιπλέον, έρχεται όταν μόνο εξετάζονται ορισμένα κελιά -- γενικά κελιά με υψηλές πιθανότητες για λάθος.

Λάθη από το στάδιο κύκλων ζωής

Στον προγραμματισμό, τα ποσοστά λάθους ποικίλλουν από το στάδιο κύκλων ζωής. Οι προγραμματιστές κάνουν πολλά λάθη όταν χτίζουν μια ενότητα αλλά πιάνουν πολλά από αυτά τα λάθη προτού να τελειώσουν την ενότητα και τα υποβάλλουν στην επιθεώρηση κώδικα, έλεγχο στοιχείων ή και τα δύο. Η επιθεώρηση κώδικα

ενότητας και ο έλεγχος πιάνουν τα περισσότερα λάθη, και ένα δεύτερο κύμα της επιθεώρησης και της δοκιμής κώδικα σε τελική μορφή πιάνει ακόμα άλλα λάθη. Σχετικά λίγα λάθη πρέπει να επιζήσουν στα τελικά λειτουργικά συστήματα.

Λάθη κατά τη διάρκεια της καταχώρησης κελιών

Μόνο δύο μελέτες έχουν εξετάσει τα λάθη κατά τη διάρκεια της καταχώρησης κελιών (Lerch, 1988 Olson & Nilsen, 1987-1988), δηλαδή όταν εισάγει ο υπεύθυνος για την ανάπτυξη τους τύπους. Μια ακόμη μελέτη εξέτασε λάθη όταν αυτοί δακτυλογραφούσαν τα κελιά ετικετών (Floyd & Pyun, 1987). Αυτά τα στοιχεία, ενώ είναι περιορισμένα , επιβεβαιώνουν ότι οι άνθρωποι κάνουν πολλά λάθη ενώ λειτουργούν και πιάνουν πολύ λίγα από αυτά (αλλά όχι όλα). Στις χρησιμοποιούμενες μεθοδολογίες, οι καταχωρητές διακόπηκαν από τους ερευνητές για να διορθώσουν τα λάθη ενώ γίνονταν , έτσι δεν ξέρουμε το τελικό ποσοστό διορθώσεων λάθους από αυτές τις μελέτες.

Ένα ενδιαφέρον σχέδιο από τη (1988) μελέτη Lerch ,είναι ότι τα μηχανικά ποσοστά λάθους αυξήθηκαν εντυπωσιακά όταν εξισώσεις που περιείχαν αναφορές σε κελιά που ήταν σε δύο διαφορετικές στήλες και διαφορετικές σειρές από το κελί που περιέχει τον τύπο.

Λάθη στο τέλος του σταδίου ανάπτυξης

Οι περισσότερες μελέτες στον πίνακα 1 εξέτασαν τα λάθη στο τέλος του σταδίου ανάπτυξης, όταν οι χρήστες είπαν ότι τελείωσαν την ανάπτυξη των λογιστικών φύλλων τους. Εκτός από έρευνα που εξέτασε μόνο τους επιλεγμένους υψηλού κινδύνου τύπους (Janvrin & Morrison, 1996 2000), τα ποσοστά λάθους κελιών σε αυτές τις μελέτες ήταν παρόμοια, παρά το γεγονός ότι οι χρήστες ήταν από τους αρχαρίους ως τους ιδιαίτερα πεπειραμένους υπεύθυνους για την ανάπτυξη λογιστικών φύλλων. Μια μελέτη (Panko & Sprague,)σύγκρινε προπτυχιακούς σπουδαστές διοίκησης επιχειρήσεων του 1998 , σπουδαστές MBA με λίγη εμπειρία με λογιστικά φύλλα και σπουδαστές MBA με περισσότερο από 250 ώρες εμπειρία ανάπτυξης λογιστικών φύλλων. Τα CERs τους ήταν πολύ παρόμοια. Ακόμα και όταν ο στόχος (ο Wall στόχος) επιλέχτηκε για να είναι πολύ απλός και σχεδόν απολύτως χωρίς απαιτήσεις γνώσης περιοχών (Panko &

Sprague, 1998 Teo & Tan, 1997), περίπου 40% όλων των λογιστικών φύλλων περιείχε τα λάθη, και το CER ήταν περίπου 2%,

Λάθη που βρίσκονται και που λείπουν στην επιθεώρηση κώδικα

Η επιθεώρηση κώδικα ή κάποια άλλη μορφή εντατικής δοκιμής μπορεί να απαιτηθεί για να ανιχνεύσει τα λάθη που παραμένουν στο τέλος του σταδίου ανάπτυξης. Ο Fagan (Fagan, 1976) ανέπτυξε μια περιεκτική πειθαρχία για την επιθεώρηση κώδικα. Όπως σημειώθηκε νωρίτερα, σε αυτή την μέθοδος έχουμε μια ομάδα από άτομα που επιθεωρούν μια ενότητα προγράμματος χωριστά με το πέρασμα της ενότητας γραμμή-γραμμή. Κατόπιν, η ομάδα συναντιέται ως ομάδα και περνά πάλι την ενότητα γραμμή-γραμμή.

Υπάρχουν ισχυρές οδηγίες για τα πράγματα όπως το μέγιστο μήκος για την ενότητα που εξετάζεται (συνήθως 100 έως 400 γραμμές κώδικα) και μέγιστα ποσοστά επιθεώρησης (συνήθως 100 έως 200 γραμμές ανά ώρα). Εάν αυτά τα ποσοστά ξεπερνιούνται, τα ποσοστά ανίχνευσης λάθους πέφτουν αρκετά (Panko, 200δ).

Η εμπειρία έχει δείξει ότι η επιθεώρηση κώδικα είναι πολύ δύσκολη. Στα πειράματα, διαπιστώθηκε ότι οι μεμονωμένοι επιθεωρητές πιάνουν μόνο κατά το ήμισυ ή λιγότερο των λαθών ανά ενότητα προγράμματος (Basili & Selby, 1986 Johnson & Tjahjono, 1997 Myers, 1978). Ακόμα και λειτουργική ομαδική επιθεώρηση συνήθως πιάνει μόνο 80% ή λιγότερο όλων των λαθών σε μια ενότητα προγράμματος (Panko, 2005α).

Η επιθεώρηση κώδικα, όπως και η ανάπτυξη, έχει παρουσιάσει ισχυρές παραλληλίες με τον προγραμματισμό. Σε τρία πειράματα που περιέλαβαν ως θέματα τη μεμονωμένη επιθεώρηση κώδικα ,λογιστικών φύλλων που σπέρνονται με λάθη, οι χρήστες πιάνουν επίσης τα μισά από όλα τα λάθη ή λιγότερο(Galletta *et al* , 1993 Galletta, Hartzel, Johnson, & Joseph, 1997 Panko & Sprague, 1998). Αυτές οι μελέτες, εντούτοις, δεν επέβαλαν τις παραδοσιακές χρονικές πειθαρχίες ή μια ομαδική φάση επιθεώρησης κώδικα .Σε μια μελέτη (Panko, 1999), οι παραδοσιακά χρονικές πειθαρχίες και η ομαδική φάση επιθεώρησης κώδικα χρησιμοποιήθηκε σε μια επιθεώρηση κώδικα ενός λογιστικών φύλλων με προπτυχιακούς φοιτητές MIS Τα άτομα βρήκαν 67% όλων των σπαρμένων λαθών σε αυτήν την μελέτη, και η ομαδική επιθεώρηση κώδικα έφερε το ποσοστό ανακαλύψεων σε 92%. Εντούτοις οι ομάδες δεν ανακάλυψαν κανένα λάθος που δεν ήταν ανακαλυμμένο κατά τη διάρκεια της μεμονωμένης επιθεώρησης κώδικα

Στην πραγματικότητα, μια ομάδα απέτυχε να καταγράψει ένα λάθος κατά τη διάρκεια της φάσης ομάδας εκτός από εκείνο που τα μέλη είχαν πιάσει κατά τη διάρκεια της μεμονωμένης φάσης.

Λάθη στους λειτουργικούς υπολογισμούς με λογιστικά φύλλα

Τα εργαστηριακά πειράματα δημιουργούν πάντα τα ερωτήματα στα μυαλά πολλών ανθρώπων. Ευτυχώς, έχουμε τα στοιχεία από τις επιθεωρήσεις κώδικα των λειτουργικών λογιστικών φύλλων , πέραν της μελέτη Hicks που περιγράφηκε ήδη. Αυτές οι επιθεωρήσεις στερήθηκαν της αυστηρής προσέγγισης που χρησιμοποιείται κατά την επίσημη επιθεώρηση κώδικα προγραμματισμού, εντούτοις, ίσως έτσι βρήκαν πιθανώς μόνο ένα μέρος λαθών στα λογιστικά φύλλα που εξέτασαν.

Η πρώτη μελέτη, από Davies και Ikin (Davies & Ikin, 1987), επιθεώρησε 19 λειτουργικά λογιστικά φύλλα από 10 υπεύθυνους για την ανάπτυξη σε 10 διαφορετικές εταιρίες. Οι υπεύθυνοι εξέφρασαν θέματα εμπιστοσύνη στην ακρίβεια των λογιστικών φύλλων τους. Εντούτοις τέσσερις από τα λογιστικά φύλλα (21%) βρέθηκαν με σοβαρά ποσοτικά λάθη, και 76% είχε ποσοτικά ή ποιοτικά λάθη.

Ένα λάθος περιέλαβε μια $7.000.000 μεταφορά κεφαλαίων μεταξύ των τμημάτων. Σε μια άλλη περίπτωση, εκεί ήταν ασυμβίβαστες μετατροπές νομίσματος στα διαφορετικά μέρη του λογιστικών φύλλων. Το τρίτο πρόβλημα ήταν μια αρνητική ισορροπία για το απόθεμα Υπήρξε ανεπαρκής τεκμηρίωση σε 68% των λογιστικών φύλλων . Τα δέκα από τα 19 απέτυχαν να χρησιμοποιήσουν την προστασία κελιών.

Μόνο ένα είχε χρησιμοποιήσει λογισμικό λογιστικού ελέγχου για να ελέγξει τα λογιστικά φύλλα, και οι όχι αυτόματοι λογιστικοί έλεγχοι ήταν "σπάνιοι." Δυστυχώς, οι συντάκτες δεν περιέγραψαν τη μεθοδολογία επιθεώρησής τους.

Αργότερα, Cragg και ο King (Cragg & King, 1993) επιθεώρησαν 20 λογιστικά φύλλα από 10 εταιρίες. Αυτός ο λογιστικός έλεγχος βρήκε σοβαρά λάθη σε 5 λογιστικά φύλλα (25%). Αυτή τη φορά, οι συντάκτες περιέγραψαν τη μεθοδολογία τους. Ένα μεμονωμένο πρόσωπο διεύθυνε μία δίωρη επιθεώρηση κώδικα. Επειδή τα λογιστικά φύλλα κυμάνθηκαν από 150 έως 10.000 κελιά στο

μέγεθος, ο χρόνος επιθεώρησης κώδικα ήταν πολύ πιο σύντομος από ότι οι πειθαρχημένες επιθεώρησης κώδικα θα επέτρεπαν.

Οι ίδιοι οι συντάκτες σημείωσαν ότι έπιασαν πιθανώς μόνο μερικά από τα λάθη μέσα στα λογιστικά φύλλα. Οι συντάκτες σημείωσαν ότι τα λογιστικά φύλλα χαρακτηρίστηκαν από φτωχό σχεδιάγραμμα, η χρήση της προστασίας κελιών σε μόνο 30%, η χρήση των ονομάτων σειράς σε μόνο 45%, και άλλες σχεδιαστικές ατέλειες Μόνο κατά το ήμισυ υπήρχαν σαφώς χωρισμένα τμήματα της εισαγωγής και του αποτελέσματος, και όλα ανακάτευαν τα κελιά υπολογισμού με τα κελιά αποτελέσματος. Μόνο δύο είχαν ελέγξει την είσοδο των τύπων πριν την εισαγωγή τους στα λογιστικά φύλλα. Ογδόντα τοις εκατό είχαν ελεγχθεί με τα δεδομένα εισόδου, αν και το βάθος τέτοιου ελέγχου ποίκιλε. Μόνο ένα είχε ελεγχθεί από ένα άλλο πρόσωπο. Κατά το ήμισυ χρησιμοποιημένες μακροεντολές, που δείχνουν την ιδιαίτερη πολυπλοκότητα και εκλέπτυνση.

Μια καθολική πτυχή των λογιστικών φύλλων Cragg και King (Cragg & King, 1993) ήταν άτυπη επαναληπτική ανάπτυξη. Κανένας από τους υπολογισμούς με λογιστικά φύλλα δεν χρησιμοποίησε επίσημη προδιαγραφή, σχέδιο, ή κωδικοποίηση. Επιπλέον, υπήρξε εκτενής αναθεώρηση λόγω φτωχού αρχικού σχεδίου. Είχε υπάρξει μια διάμεσος επτά αναθεωρήσεων, παρά το γεγονός ότι τα λογιστικά φύλλα ήταν μόνο σε χρήση έξι μηνών. Σε 17 λογιστικά φύλλα, η δομή τους έπρεπε να αλλάξει κατά τη διάρκεια των αναθεωρήσεων. Σε έξι περιπτώσεις, υπήρχαν προβλήματα με τα λογιστικά φύλλα στις προηγούμενες εκδόσεις

Αργότερα, σε μια προσωπική επικοινωνία με το συντάκτη, Ο Butler (1996) περιέγραψε το λογιστικό έλεγχο σε 273 λογιστικά φύλλα που υποβάλλονται σε στο τελωνείο και τη φορολογική υπηρεσία στο Ηνωμένο Βασίλειο. Η μεθοδολογία λογιστικού ελέγχου περιέλαβε τη χρήση ενός προγράμματος με σκοπό να ψάξει ασυνέπειες στον λογιστικά φύλλα. Εάν το πρόγραμμα σημάδευε ένα τμήμα του λογιστικού φύλλου, ένας μεμονωμένος ελεγκτής εξέτασε εκείνο το τμήμα του λογιστικού φύλλου για λάθη. Αυτό το πρόγραμμα έπιασε πιθανώς μόνο ένα μέρος όλων των λαθών, επειδή δεν μπόρεσε να πιάσει πράγματα όπως οι παραλείψεις, αριθμοί που λάθος δακτυλογραφημένοι, τους περισσότερους ανακριβείς τύπους, και πολλοί είδη λάθους. Συμπερασματικά , η ομάδα βρήκε λάθη σε 10,7% των λογιστικών φύλλων.

Από το 1995, πέντε μελέτες έχουν εξετάσει λογιστικά φύλλα χρησιμοποιώντας την καλύτερη μεθοδολογία. Ο Hicks (1995) έλεγξε έναν τεράστιο λογιστικό φύλλο σύνταξης προϋπολογισμού κεφαλαίου

με 3.856 κελιά. Αυτό ήταν μια επιθεώρηση τριών ατόμων, στον κώδικα κελί –κελί. Λάθη βρέθηκαν σε μόνο 1,2% όλων των γραμμών του κώδικα, αλλά τα λάθη θα κόστιζαν πέρα από ένα δισεκατομμύριο δολάρια εάν δεν είχαν πιαστεί.

Στην Αγγλία, ορισμένα λογιστικά φύλλα πρέπει να ελεγχθούν από το νόμο. Οι Coopers και Lybrand και η KPMG έχουν ενεργό συμβουλευτικό ρόλο σε γνωμοδοτήσεις ελέγχου για λογιστικά φύλλα. Μαζί, βρήκαν σημαντικά λάθη σε 45 λογιστικά φύλλα που έλεγξαν οι Coopers & Lybrand . Μετρούσαν μόνο τα λάθη που διαμόρφωναν τελικό αποτέλεσμα διαφορετικό κατά τουλάχιστον 5%, Η KPMG απαριθμούσε μόνο "σημαντικά λάθη." Το 2003,ο συντάκτη πέρασε από συνέντευξη σε ελεγκτές από αυτές τις δύο οργανώσεις χωριστά. Και οι δύο είπαν ότι δεν είχαν δει ποτέ ένα χωρίς λάθη λογιστικό φύλλο κατά τη διάρκεια των λογιστικών ελέγχων.

Επιπλέον, και οι δύο έδωσαν ανεξάρτητα στοιχεία που δείχνουν ότι περίπου 5% όλων των λογιστικών φύλλων που έλεγξαν είχαν "πολύ σοβαρά λάθη." Σημείωσαν ότι αυτό το ποσοστό μπορεί να είναι χαμηλό για άλλους τύπους λογιστικών φύλλων επειδή τα λογιστικά φύλλα που ελέγχονται έχουν καθορισμένο με σαφήνεια σχήμα. Επιπλέον, και οι δυο εταιρίες χρησιμοποιούν την επιθεώρηση από ένα μόνο άτομο του κώδικα.

Το 2000, ο Ray Butler, που έλεγξε λογιστικά φύλλα για την είσπραξη φόρων φόρου στο UK, βρήκε τα αγώγιμα λάθη σε έξι από επτά λογιστικά φύλλα χρησιμοποιώντας την επιθεώρηση από ένα μόνο άτομο του κώδικα

Αγώγιμα λάθη περιλάμβαναν απαιτήσεις για την πληρωμή πρόσθετων φόρων ή επιστροφές ποσών. (Στην πρώτη μελέτη του, ο Butler χρησιμοποίησε μια λιγότερο αυστηρή μέθοδο που σχεδιάστηκε για να ελαχιστοποιήσει το κόστος και βρήκε λάθη μόνο σε 10,7% των ελεγχόμενων λογιστικών φύλλων).

Πρόσφατα, ο Lawrence και ο Lee (2004) στην Αυστραλία έλεγξαν 30 λογιστικά φύλλα που δημιουργήθηκαν για να δικαιολογήσουν τη χρηματοδότηση προγράμματος. Αυτοί οι λογιστικοί έλεγχοι χρησιμοποίησαν την ενός ατόμου επιθεώρηση κώδικα. τα λογιστικά φύλλα περιείχαν κατά μέσο όρο 2.182 τύπους, και κατά μέσον όρο 6,9% των κελιών τύπου είχε ζητήματα ελέγχου. Πήρε έναν μέσο όρο έξι επαναλήψεων προτού να μπορέσουν τα λογιστικά φύλλα να πιστοποιηθούν.

Συνολικά, ότι βλέπουμε στα λειτουργικά λογιστικά φύλλα είναι σύμφωνο με τα στοιχεία από τις εργαστηριακές μελέτες. Αν

και οι εργαστηριακές μελέτες έχουν γενικά τα υψηλότερα ποσοστά λάθους, οι μελέτες τομέων δεν έκαναν μεθοδολογίες χρήσης που θα μπορούσαν πιθανώς να βρουν περισσότερα λάθη. Στους τέσσερις λογιστικούς ελέγχους τομέων για τους οποίους έχουμε τα ποσοστά λάθους κελιών, επιπλέον, τα CERs, που κυμαίνονται από 0,4% ως 6,9%, είναι παρόμοια στο μέγεθος με CERs που φαίνεται στα πειράματα. Επιπλέον, οι περισσότερες μελέτες χρησιμοποιούν ενός ατόμου επιθεώρηση κώδικα, η οποία χάνει μερικά λάθη.

Η ομαδική εργασία βοηθά;

Στην εθνογραφική μελέτη τους για 11 υπεύθυνους για την ανάπτυξη λογιστικών φύλλων, ο Nardi και Miller 1991 διαπίστωσαν ότι όλοι οι υπεύθυνοι για την ανάπτυξη χρησιμοποίησαν τουλάχιστον ένα άλλο άτομο κατά τη διάρκεια της ανάπτυξης. Σε μερικές περιπτώσεις, το άλλο πρόσωπο βοήθησε τον υπεύθυνο για την ανάπτυξη να κωδικοποιήσει τα δύσκολα μέρη από τα λογιστικά φύλλα. Σε άλλες περιπτώσεις, δύο ή περισσότεροι άνθρωποι χρησιμοποίησαν τον διαδικασία ανάπτυξης λογιστικών φύλλων ως τρόπο να μοιραστεί η γνώση περιοχών. Σε άλλες περιπτώσεις, το άλλοι πρόσωπο εισήγαγε αριθμούς και έλεγχε τα αποτελέσματα εισαγωγής , με αυτόν τον τρόπο βοηθώντας την ανίχνευση λάθους. Οι Nardi και Miller αποκαλούσαν αυτό το τελευταίο σχέδιο "συνεταιριστική διόρθωση."

Αυτό οδήγησε Panko και Halverson (Panko & Halverson, 1997) για να εξετάσει την ομάδα ανάπτυξης λογιστικών φύλλων . Τα θέματά τους ανέπτυξαν ενός προσχεδίου εισοδηματικής δήλωσης ,από ένα πρόβλημα λέξης, που λειτούργησε μόνοι, σαν δυάδες, ή σαν τετράδες. Οι δυάδες μείωσαν τα λάθη κατά ένα τρίτο, ενώ οι τετράδες κατά περίπου δύο τρίτα. Μόνο το τελευταίο ήταν στατιστικά σημαντικό. Επιπλέον, η ομαδική ανάπτυξη ήταν μόνο αποτελεσματική για ορισμένους τύπους λαθών.

Η αρχική (Panko & Halverson, 1997) μελέτη Panko και Halverson επέτρεψε σε μερικούς να εργαστούν έξω από το εργαστήριο, προκειμένου να αποφευχθεί το γεγονός ότι πρέπει να εργαστούν υπό τους σχεδιασμένους όρους. Φυσικά αυτό σημαίνει επίσης και χαμένο πειραματικό έλεγχος. Το1999, οι Panko και Halverson ξανάκαναν τη μελέτη στο εργαστήριο, χρησιμοποίηση μεμονωμένη και τριαδική ανάπτυξη. Επανέλαβαν επίσης το στόχο τους να αφαιρέσουν τις ασάφειες που στην πρώτη μελέτη βρέθηκε

εξαιρετικά δύσκολη. Λαμβάνοντας αυτές τις αλλαγές υπόψη, η νέα μελέτη έδωσε σχεδόν τα ίδια αποτελέσματα.

Τέλος, όπως σημειώνεται νωρίτερα, μια μελέτη από το συντάκτη (1999) βρήκε ότι η ομαδική επιθεώρηση κώδικα με προπτυχιακούς φοιτητές MIS

ανακάλυψε 83% από όλα τα σπαρμένα λάθη σε ένα λογιστικό φύλλο, αν και η ομάδα δεν βρήκε λάθη που προηγουμένως δεν είχαν βρεθεί από τα μέλη της ομάδας, που είχαν επιθεωρήσει μόνοι τους πριν από την ομαδική επιθεώρηση κώδικα.

Συνολικά, η ανάπτυξη ομάδας και η δοκιμή εμφανίζονται στις περιοχές που ακολουθούν.

3.5 Τύποι λαθών

Όταν πολλοί άνθρωποι σκέφτονται τα λάθη σε λογιστικά φύλλα, σκέφτονται κάποιον που πληκτρολογεί λάθος έναν αριθμό, που δακτυλογραφεί ένα σημείο συν αντί για το μείον σε έναν τύπο, ή που δείχνει το λανθασμένο κελί κατά την είσοδο ενός τύπου. Ενώ αυτά τα λάθη υπάρχουν, υπάρχουν πολλά άλλου τύπου λάθη στην ανάπτυξη λογιστικών φύλλων. Οι (Panko & Halverson, 1996) δίνουν μια ταξινόμηση των τύπων λάθους.

Κατ' αρχάς, υπάρχουν ποσοτικά λάθη, στα οποία ο υπολογισμός με λογιστικά φύλλα δίνει ένα ανακριβές αποτέλεσμα.

Οι μελέτες στον πίνακα 1 εξετάζουν μόνο τα ποσοτικά λάθη. Εντούτοις υπάρχουν επίσης ποιοτικά λάθη που μπορούν να οδηγήσουν στα ποσοτικά λάθη αργότερα, κατά τη διάρκεια της συντήρησης, της what-if ανάλυσης, ή άλλες δραστηριότητες. Ο Reason, 1990, χρησιμοποιεί τον όρο "λανθάνοντα λάθη" για τέτοια προβλήματα. Οι Teo και Tan, 1997, κατέδειξαν σε ένα πείραμα πώς ένας τύπος ποιοτικού λάθους οδήγησε στα ποσοτικά λάθη κατά τη διάρκεια της what-if ανάλυσης Οι Panko & Halverson, 1996, ακολουθώντας τον Allwood (1984) επίσης βρήκε χρήσιμο να διακρίνει μεταξύ τριών τύπων ποσοτικών λαθών. Τα Μηχανικά λάθη είναι απλά λάθη, όπως ο λάθος αριθμός ή υπόδειξη σε λανθασμένο κελί.

Τα λάθη λογικής περιλαμβάνουν την είσοδο του λανθασμένου τύπου λόγω ενός λάθους στο συλλογισμό. Όπως σημειώνεται νωρίτερα, τα ποσοστά λάθους λογικής είναι υψηλότερα από τα

μηχανικά ποσοστά λάθους. Τα Λάθη λογικής επίσης είναι δυσκολότερο να τα ανιχνεύσει και να διορθώσει (Allwood, 1984). Ο πιό επικίνδυνος τύπος λάθους είναι το λάθος παράλειψης, στο οποίο κάτι αφήνεται έξω. Τα λάθη παράλειψης εμφανίζονται να είναι εξαιρετικά δύσκολο να ανιχνευθούν (Allwood, 1984 Bagnara, Stablum, Rizzo, Fontana, & Ruo, 1987 Woods, 1984).

Όταν οι Panko και Halverson (Panko & Halverson, 1997) ανέλυσαν τους τύπους λαθών που κάνει κάποιος όταν αναπτύσσει ένα λογιστικά φύλλα, διαπίστωσαν ότι και οι τρεις μορφές στα λάθη ήταν κοινές. Αργότερα, Panko και Sprague (Panko & Sprague, 1998) βρήκαν το ίδιο ευρύ σχέδιο των λαθών. Οι Panko και Halverson συγκρίνανε τους διαφορετικούς τύπους λαθών με πολλαπλάσια θανατηφόρα δηλητήρια. Ακόμα κι αν όλα τα λάθη των άλλων δύο τύπων εξαλείφτηκαν, κάθε ένας τύπος λάθους μόνο θα είχε παραγάγει έναν απαράδεκτο αριθμό ανακριβούς λογιστικού φύλλου.

Οι τεχνικές για να μειωθούν τα λάθη μπορούν να είναι καλύτερες στην εύρεση μερικών τύπων λαθών από άλλα, έτσι είναι σημαντικό να αρχίσει να αναπτύσσει σώματα (συλλογές) λαθών για να μάθει πώς οι συχνοί διαφορετικοί τύποι λαθών είναι πραγματικά.

Πρακτικές και πολιτικές ανάπτυξης

Έρευνες υπεύθυνων για την ανάπτυξη

Μερικές μελέτες έχουν εξετάσει τις διαδικασίες που οι υπεύθυνοι για την ανάπτυξη χρησιμοποιούν για να δημιουργήσουν λογιστικά φύλλα. Αυτές οι μελέτες ζήτησαν από τον υπεύθυνο για την ανάπτυξη να επιλέξει ένα ενιαίο λογιστικό φύλλο και τον περιγράψει λεπτομερώς. Αυτή η μέθοδος οδήγησε πιθανώς στην προκατάληψη προς τα μεγάλα πλέον των συνηθισμένων λογιστικά φύλλα. Ακόμη και με αυτήν την προειδοποίηση, εντούτοις, τα αποτελέσματα ήταν εξαιρετικά ενδιαφέροντα.ι.

Οι αναφερθέντες από Davies & Ikin, 1987, και Cragg & King, 1993, πρώτοι λογιστικοί έλεγχοι ερώτησαν τους υπεύθυνους για την ανάπτυξη τομέων για λογιστικά φύλλα και τις πρακτικές ανάπτυξης. Και οι δύο βρήκαν πολύ άτυπες διαδικασίες ανάπτυξης, ελάχιστα συστηματική επιθεώρηση κώδικα, και λίγη χρήση πολλών άλλων σημαντικών πειθαρχιών ανάπτυξης .

Οι Cragg και King σημείωσαν ότι ι οι μισοί από τους υπολογισμούς με λογιστικά φύλλα τους ήταν χτισμένοι χωρίς προγενέστερο σχέδιο ή προγραμματισμό, σχεδόν κατά το ήμισυ είχε φτωχό σχέδιο, κατά το ήμισυ φτωχό σχεδιάγραμμα, και μόνο κατά το ήμισυ είχε τεκμηρίωση οποιουδήποτε τύπου. Ένας σχολίασε ότι είχε πρόβλημα να καταλάβει τα παλιά λογιστικά φύλλα.

Νωρίτερα, αναφέραμε τις εθνογραφικές συνεντεύξεις Nardi και Miller, 1991. Γενικά, διαπίστωσαν ότι οι υπεύθυνοι για την ανάπτυξη πήραν ιδιαίτερες προφυλάξεις αναπτύσσοντας τα λογιστικά φύλλα τους, αν και οι συνεντεύξεις δεν συγκέντρωσαν τα ποσοτικά στοιχεία.

Οι ερωτηθέντες γνώριζαν έντονα τους κινδύνους του λάθους και ξόδεψαν ιδιαίτερο χρόνο εργαζόμενο για να μειώσουν τα λάθη, συμπεριλαμβανομένης της δημιουργίας των διασταυρώσεων , αιφνιδιαστικούς ελέγχους τύπων με αριθμομηχανές και την συμμετοχή άλλων εξετάζουν να εξετάζουν το αποτέλεσμα. Εντούτοις μόνο μια από τους 11 που έδωσαν συνέντευξη είπε ότι είχε κάνει μια πλήρη επιθεώρηση κώδικα ενός λογιστικού φύλλου και αυτό ήταν ένα λογιστικό φύλλο που της δόθηκε από κάποιον άλλο (Nardi, 1993). Όπως σημειώνεται ανωτέρω, μερικοί έβαλαν άλλους να εξετάσουν τα αποτελέσματά τους, επειδή τα λάθη μπόρεσαν να γίνουν αόρατα στο συντάκτη μέσω της υπέρ-οικειότητας. Εντούτοις αυτό υπολείπεται αρκετά της επιθεώρησης κώδικα.

Αργότερα, (Hendry & Green, 1994) έκαναν μια παρόμοια εθνογραφική μελέτη, με 10 υπεύθυνους για την ανάπτυξη λογιστικών φύλλων . Προσθέσανε μια φάση στην οποία το θέμα περπάτησε μέσω λογιστικού φύλλου με έναν συντάκτη. Διαπίστωσαν ότι τα θέματα είχαν συχνά μια δύσκολη περίοδο εξηγώντας τα μέρη του λογιστικών φύλλων που οι ίδιοι είχαν χτίσει. Βρήκαν επίσης ότι μερικοί από τους υπεύθυνους για την ανάπτυξη είχαν σοβαρά προβλήματα προσπαθώντας να κάνουν τα μέρη της ανάλυση λογιστικών φύλλων να ταιριάζουν με το σχεδιάγραμμα σειρών -και-στηλών του λογιστικού φύλλου Αυτοί βρήκαν ότι η διόρθωση των λαθών να είναι ιδιαίτερα δύσκολη. Συνεπεία τέτοιων δυσκολιών , έκαναν διάφορα πράγματα για να διορθώσουν τα λάθη. Αναδρομικά η επικοινωνία με το συντάκτη, Hendry (1994) είπε ότι μόνο τρεις από τους υπεύθυνους για την ανάπτυξη ήταν "ιδιαίτερα ικανοί στους υπολογισμούς" και έκαναν πολλά πράγματα για να αποφύγουν τα λάθη. Σε άλλη προσωπική επικοινωνία με το συντάκτη, ο Greeno (1994) είπε ότι η λεπτομερής επιθεώρηση κώδικα den εμφανίζεται να είναι μέρος της κουλτούρας ανάπτυξης λογιστικών φύλλων ."

Συνολικά, και οι Nardi & Miller, 1991, και Hendry και Green, 1994; διαπίστωσαν ότι οι υπεύθυνοι κατέβαλαν ιδιαίτερες προσπάθειες όταν έχτισαν τα λογιστικά φύλλα .

Ακόμα τα προβλήματα που οι χρήστες αντιμετώπισαν, τα οποία ήταν ιδιαίτερα εμφανή σύμφωνα με την μεθοδολογία Hendry και Green, είναι ασαφές πόσο επιτυχή είναι το λάθος χρήστη και η μείωση των λαθών. Λαμβάνοντας υπόψη τη σποραδικότητα της περιεκτικής δοκιμής, εντούτοις, πλήρης επιτυχία φαίνεται απίθανη.

Άλλες μελέτες έχουν χρησιμοποιήσει τις έρευνες ερωτηματολογίων για να συλλέξουν τα στοιχεία από μεγαλύτερο αριθμό ερωτηθέντων. Παραδείγματος χάριν, οι Schultheis & Sumner, 1994, είχαν 32 MBA σπουδαστές όπου ο κάθε ένας συζητούσε το πιο πρόσφατο λογιστικό φύλλο τους. Μεγέθη για τα λογιστικά φύλλα δεν δόθηκε, και ένα τέταρτο των συμμετεχόντων είχε αναπτύξει πέντε ή λιγότερα λογιστικά φύλλα. Η μελέτη προέβλεπε οι συμμετέχοντες να αξιολογήσουν κάθε λογιστικό φύλλο σε 11 συστάδες των κινδύνων και η χρήση 9 συστάδων των πρακτικών ελέγχου κατά τη διάρκεια της ανάπτυξης. Κάθε συστάδα είχε διάφορες απαριθμημένους κίνδυνους ή πρακτικές ελέγχου. Οι συχνότεροι κίνδυνοι που αναφέρθηκαν περιλάμβανε την υπολογιζόμενη διάρκεια ζωής της εφαρμογής (που μπορεί να υπερβεί τη διάρκεια της θέσης του υπεύθυνου για την ανάπτυξη στην εταιρία), χρήση πέρα από τον υπεύθυνο για την ανάπτυξη μέσα στην εταιρία, χρήση πέρα από την εταιρία, αριθμός χρηστών, και το επίπεδο πολυπλοκότητας του λογιστικών φύλλων .

Λογιστικά φύλλα υψηλού κινδύνου χρησιμοποίησαν ελαφρώς περισσότερους ελέγχους (6.7) από τους υπολογισμούς με λογιστικά φύλλα χαμηλού κινδύνου (4,6), αλλά η χρήση των ελέγχων ήταν χαμηλή και στις δύο περιπτώσεις. Ο πιο διαδεδομένος έλεγχος ήταν η επαλήθευση της λογικής, αλλά αυτό περιέλαβε μια ευρύτητα προσεγγίσεων διαφορετικού ύφους. Η τεκμηρίωση ταξινομήθηκε ως τρίτη , μετά από τη συμμετοχή στην επίσημη κατάρτιση, αλλά "υπήρξε πολύ λίγη τεκμηρίωση," με ένα τέταρτο των λογιστικών φύλλων που δεν είχαν κανενός είδους. Οι επίσημοι λογιστικοί έλεγχοι και οι επίσημες αναθεωρήσεις ήταν μεταξύ των ελάχιστα-χρησιμοποιημένων ελέγχων. Από τα 32 λογιστικά φύλλα, μόνο πέντε αναθεωρήθηκαν από έναν ελεγκτή ή έναν σύμβουλο, και μόνο δέκα είχαν αναθεωρήσεις από καθένα εκτός από τον υπεύθυνο για την ανάπτυξη.

Σε μια μεγαλύτερη κλίμακα, οι Floyd, Wall και Marr ,1995, έστειλαν ερωτηματολόγια σε 72 τελικούς χρήστες σε τέσσερις

εταιρίες. Κάθε ένας κλήθηκε να περιγράψει ένα απλό λογιστικό φύλλο . Το μέσο μέγεθος των επιλεγμένων λογιστικών φύλλων ήταν 6.000 κελιά.

Οι ερωτηθέντες είπαν ότι τα λογιστικά φύλλα τους ήταν σημαντικα υπό την έννοια αυτή σημαντικές αποφάσεις βασίστηκαν σε αυτά Είπαν επίσης, αν και σε μια μικρότερη έκταση, ότι τα λογιστικά φύλλα τους άσκησαν υλική οικονομική επίδραση στην εταιρία. Η σχετική σημασία, επ' ευκαιρία δεν συσχετίστηκε με το μέγεθος, και τα "μικρά και μεγάλα λογιστικά φύλλα ήταν εξίσου υπεύθυνα για υλικές αποφάσεις υποστήριξης." .Ογδόντα τοις εκατό είπαν ότι θα ήταν σε θέση να λειτουργήσουν χωρίς τα λογιστικά φύλλα, αν και με κάποια δυσκολία. Ένας από έξι, εντούτοις, τους είπε ότι δεν θα ήταν σε θέση να εργαστεί χωρίς τα λογιστικά φύλλα. Η ανάπτυξη ήταν έντονα επαναληπτική.

Υπήρξε ένα υψηλό επίπεδο εμπιστοσύνης ότι τα λογιστικά φύλλα που περιγράφηκαν ήταν σωστά.

Ο Hall , 1996, ερεύνησε 106 υπεύθυνους για την ανάπτυξη λογιστικών φύλλων στην Αυστραλία. Το μέσο μέγεθος στα επιλεγμένοι λογιστικά φύλλα ήταν 216 KB, και επειδή τα περισσότερα λογιστικά φύλλα ήταν αναπτυγμένα χρησιμοποιώντας το Lotus 1-2-3, αυτό ήταν ιδιαίτερο μεγάλο μέγεθος. Μεταξύ των συμπερασμάτων ήταν τα εξής:

1. Τα λογιστικά φύλλα ήταν σημαντικά. Μόνο 7% των λογιστικών φύλλων ήταν χαμηλής σημασία, 39% ήταν εξαιρετικά σημαντικά. 27% τροποποιούσαν υπάρχοντα εταιρικά στοιχεία , 49% δημιουργούσαν νέα εταιρικά στοιχεία. 67% οργανώθηκαν σε κανονική βάση, 16% περιστασιακά, και μόνο 17% μία φορά ή μερικές φορές. Μόνο 17% τα αποτελέσματα χρησιμοποιούνταν μόνο από τον υπεύθυνο για την ανάπτυξη, ενώ 23% είχε τα αποτελέσματα που χρησιμοποιούνταν από πολλά τμήματα και 29% χρησιμοποίησαν τα αποτελέσματα έξω από την εταιρία.

2. Τα λογιστικά φύλλα ήταν σύνθετα. 45% χρησιμοποιούσαν μακροεντολές. 36% είχε συνδέσεις με άλλα λογιστικά φύλλα. 21% είχε συνδέσεις με τις βάσεις δεδομένων. 47% χρησιμοποίησε IF συναρτήσεις .66% χρησιμοποιούσε απόλυτη παραπομπή.

3. Η ανάπτυξη χρησιμοποίησε μόνο τα περιορισμένα μέτρα προστασίας. Μόνο κατά το ήμισυ είχε σχέδια ενοτήτων. Μόνο 49% χρησιμοποιούσε προστασία κελιών. Μόνο 46% χρησιμοποιούσε διασταύρωση.

4. Η δοκιμή στοιχείων ήταν περιορισμένη. 71% έλεγχε τους τύπους με τα στοιχεία, αλλά μόνο 50% έκανε έτσι με τα αποτελέσματα που καθορίστηκαν μπροστά από το χρόνο, μόνο 42% χρησιμοποίησε τα στοιχεία δοκιμής με τα λάθη, και μόνο 33% χρησιμοποίησε τα στοιχεία δοκιμής στα όρια της κανονικής σειράς.

5. Ο έλεγχος ήταν περιορισμένος. Μόνο 17 % ελέγχθηκαν από άλλο υπεύθυνο, 5 % από εσωτερικό ελεγκτή και 6 % από εξωτερικό ελεγκτή

6. 18% είπε ότι το πρόγραμμά τους ήταν επείγον.

Κατά τρόπο ενδιαφέροντα, ο Hall ρώτησε τους συμμετέχοντες εάν χρησιμοποίησαν διάφορους ελέγχους και εάν έπρεπε να έχουν χρησιμοποιήσει αυτούς τους ελέγχους. Για σχεδόν όλους τους ελέγχους, οι περισσότεροι υπεύθυνοι για την ανάπτυξη αναγνώρισαν ότι έπρεπε να το έχουν χρησιμοποιήσει από ότι πραγματικά το χρησιμοποίησαν.

Γενικώς οι πεπειραμένοι υπεύθυνοι για την ανάπτυξη ήταν πιθανότερο να χρησιμοποιήσουν τους περισσότερους ελέγχους από ότι οι άπειροι υπεύθυνοι για την ανάπτυξη, αλλά η διαφορά ήταν μόνο μέτρια.

Συνολικά, αυτές οι μελέτες δείχνουν ότι πολλοί λογιστικά φύλλα είναι μεγάλα, σύνθετα, σημαντικά , και έχουν επιπτώσεις σε πολλούς ανθρώπους. Ακόμα η ανάπτυξη τείνει να είναι αρκετά άτυπη, και ακόμα και τετριμμένοι ο έλεγχοι όπως η προστασία κελιών δεν χρησιμοποιούνται στις περισσότερες περιπτώσεις .Στον προγραμματισμό, έλεγχο κωδικοποίησης και η δοκιμή στοιχείων απαιτείται για να μειώσει τα ποσοστά λάθους αφότου μια ενότητα αναπτύσσεται . Ακόμα η επιθεώρηση κώδικα είναι πολύ σπάνια, και ενώ η δοκιμή στοιχείων γίνεται, αυτό στερείται τέτοιες ευκαμψίας όπως χρήση εξωτερικών στοιχείων. Γενικά, ανάπτυξη λογιστικών φύλλων από τελικούς χρήστες φαίνεται να μοιάζει με την πρακτική προγραμματισμού στη δεκαετία του '50 και τη δεκαετία του '60

4. Μέθοδοι επαλήθευσης και διόρθωσης λαθών στα λογιστικά φύλλα

Σε αυτή την ενότητα θα ασχοληθούμε με προσεγγίσεις που αναφέρονται στη βελτίωση της ποιότητας των λογιστικών φύλλων. Οι δύο προσεγγίσεις στις οποίες θα αναφερθούμε ασχολούνται με τις διαφορετικές κατηγορίες σφαλμάτων όπως προαναφέραμε κατά την ταξινόμησή τους.

Κατά αρχάς θα δούμε *την οπτική αναπαράσταση του προτύπου*. Αυτός ο νοερός σχεδιασμός δίνει στον προγραμματιστή του λογιστικού φύλλου, και αντίστοιχα στον χρήστη του, περισσότερη διορατικότητα ως προς τη δόμηση του λογιστικού φύλλου, με αναμενόμενο αποτέλεσμα την επιτάχυνση της εμπειρικής διαδικασίας κατά τη διάρκεια της δημιουργίας του λογιστικού φύλλου, και την δυνατότητα να κατανοεί και αποκωδικοποιεί λογιστικά φύλλα που χρησιμοποιεί. Η δεύτερη προσέγγιση είναι αυτή των *περιοδικών ελέγχων* του λογιστικού φύλλου, όπου οι δυσκολίες που προέρχονται από έλλειψη προσδιορισμού του λογιστικού φύλλου αντιμετωπίζονται με την χρήση περιοδικών μαθηματικών ως βασικού εργαλείου.

4.1 Οπτική αναπαράσταση του προτύπου

Το γεγονός ότι τα πρότυπα των λογιστικών φύλλων[3] είναι «βαθιά κρυμμένα στους τύπους» κάνει εμφανώς πολύ δύσκολη την κατανόηση και την ανασυγκρότηση του προτύπου του λογιστικού φύλλου.

Το κρυφό πρότυπο πρέπει να ανασυγκροτηθεί, ώστε ο δημιουργός του ή ο ελεγκτής του να έχει την δυνατότητα να δει πέρα από τους τύπους στη θεμελιώδη λογική και δόμησή του. Για επιτύχουμε αυτό πρέπει να λάβουμε υπόψη και την ροή δεδομένων του λογιστικού φύλλου (όπως προτείνουν οι Boaz Ronen, Michael Palley, και Henry Lucas) αλλά και τις στατικές πλευρές, όπως τις λογικές και φυσικές περιοχές. Η δημιουργία αυτής της αναπαράστασης του προτύπου του λογιστικού φύλλου πρέπει να γίνεται αυτόματα, χωρίς καμία ή λιγοστή παρέμβαση από τον προγραμματιστή. Όταν το πρότυπο δημιουργηθεί μπορεί να χρησιμοποιηθεί για τη νοερή σχεδίαση και για την αυτόματη σύγκριση προγραμμάτων λογιστικών φύλλων.

Η οπτική αυτή αναπαράσταση πρέπει να παρέχει διαφορετικές αναλύσεις, από χαμηλή έως την υψηλότερη δυνατή ανάλυση, ώστε ο χρήστης και αντίστοιχα ο προγραμματιστής να έχει τη δυνατότητα να δει το πρόγραμμα των λογιστικών φύλλων στα επίπεδα τής φυσικής και λογικής περιοχής και τη ροή δεδομένων μεταξύ αυτών των περιοχών. Σε ένα περαιτέρω βήμα πρέπει να έχει τη δυνατότητα να εστιάσει τη προσοχή του σε ορισμένες περιοχές και να έχει μια πιο λεπτομερή εικόνα της ανάλυσης των τύπων και των κελιών αναφοράς.

Σχεδιάζουμε να εφαρμόσουμε την γραφική σχεδίαση του προτύπου κατά τρόπο ώστε η γραφική αναπαράσταση να βασίζεται στη

[3] Συνοπτική αναπαράσταση του προγράμματος του λογιστικού φύλλου

ροή των δεδομένων του λογιστικού φύλλου, και επίσης αναπαριστά τις φυσικές και λογικές περιοχές. Πρέπει ο χρήστης να έχει τη δυνατότητα να πλοηγείτε με άνεση εντός του προτύπου, όπως προτείνει ο Storey. Η αναπαράσταση πρέπει να επιτρέπει εστίαση σε συγκεκριμένα σημεία της γραφικής αναπαράστασης, χωρίς να χάνεται η γενική εικόνα της έννοιας, χρησιμοποιώντας ένα ευρύτερο ορατό πεδίο.

Το οπτικό πρότυπο που σχεδιάζουμε πρέπει να λειτουργεί ως εργαλείο για τρεις σκοπούς:

1. επιτάχυνση της εμπειρικής διαδικασίας για την ανεύρεση λύσεων σε πραγματικά προβλήματα. Προϋποθέτουμε ότι η κατανόηση του προβλήματος υποστηρίζεται από την γραφική αναπαράσταση του προτύπου του λογιστικού φύλλου.
2. κατανόηση προγραμμάτων λογιστικών φύλλων που έχουν δημιουργηθεί από άλλο προγραμματιστή.
3. καθιστά ικανή τη σύγκριση προγραμμάτων λογιστικών φύλλων στο επίπεδο του προτύπου του λογιστικού φύλλου. Αυτή η σύγκριση πρέπει να αφαιρεί τις τιμές και να λαμβάνει υπόψη μόνο τις ιδιότητες του προτύπου, όπως η ροή των δεδομένων, φυσικές και λογικές περιοχές .

Το οπτικό αυτό πρότυπο θα παρουσιάζει τις φυσικές περιοχές , δίνοντας μία ορατή εικόνα στον χρήστη, για το αν υπάρχουν κελιά διαφορετικής μορφής ή διαφορετικού εννοιολογικού περιεχομένου στη περιοχή. Μία φυσική ή λογική περιοχή μπορεί να αναπαρασταθεί ως ένα πλαίσιο, και οι διακοπές ως σειρές διαφορετικού χρώματος. Αυτή η αναπαράσταση θα βοηθήσει στον έλεγχο των σφαλμάτων *αναφοράς σε κελί με λάθος μορφή τιμής*.

Πρέπει επίσης να ελεγχθεί, αν υπάρχουν γειτονικά κελιά στη φυσική περιοχή, τα οποία έχουν την ίδια μορφή όπως τα κελιά του περιοχής.[4] Αυτό υπαινίσσεται πιθανά σφάλματα *εσφαλμένου*

[4] Εάν υπάρχουν κελιά διαφορετικής μορφής στο πεδίο, η σωστή μορφή μπορεί να διευθετηθεί εφαρμόζοντας στο πεδίο συνάρτηση ομαδοποίησης.

προσδιορισμού φυσικής περιοχής , το οποίο μπορεί να αναπαρασταθεί με το σχεδιασμό των απαραιτήτων ορίων σε διαφορετικό χρώμα.

Το πρόβλημα των *εμπλεκόμενων φυσικών περιοχών* μπορεί να λυθεί ξεχωρίζοντας τις περιοχές που συμπίπτουν με γραφική αναπαράσταση. Όμως, η ανεύρεση περιοχών που συμπίπτουν, δεν είναι ένα ασήμαντο πρόβλημα και χρειάζεται περαιτέρω μελέτη επί του θέματος.

Με τον προσδιορισμό και τον ορατό σχεδιασμό των λογικών περιοχών, μία θεωρία που δεν εκφράζεται ορατά στον χρήστη και αντίστοιχα στον προγραμματιστή μοντέρνων συστημάτων λογιστικών φύλλων, πολλά από τα προβλήματα που παρουσιάσαμε στην ενότητα 4.2 ήδη επιλύονται. Λογικές περιοχές συχνά είναι και γειτονεύοντες, παρόλο που αυτό δεν είναι απαραίτητο. Αν είναι σποραδικά διακεκομμένες από μερικά κελιά τότε αυτό υπαινίσσεται πιθανό σφάλμα *αντικατάστασης τύπου από σταθερή αξία*. Η αναπαράσταση είναι παρόμοια με την αναπαράσταση του σφάλματος *αναφοράς σε κελί με λάθος μορφή τιμής*.

4.2 Περιοδικός Έλεγχος (μεσοδιαστήματος)

Μετά τη δημιουργία ενός προγράμματος λογιστικών φύλλων για μια ειδική εφαρμογή, είναι φυσικό να ελεγχθεί η ορθότητά του. Δημιουργούμε λογιστικά φύλλα κυρίως για αριθμητικούς υπολογισμούς. Τι περιμένουμε να είναι σωστό; Συνήθως, έχουμε μία διαίσθηση του φάσματος των λογικών τιμών για κάθε κελί.

Η δημιουργία λογιστικών φύλλων βασίζεται σε κελιά στα οποία πρέπει να εισαχθούν τιμές και τύποι για υπολογισμό. Για την ορθότητα του προγράμματος των λογιστικών φύλλων, κάθε εισαγόμενη τιμή και κάθε τύπος πρέπει να είναι σωστά. Στην πραγματικότητα, τα περισσότερα λάθη στα λογιστικά φύλλα γίνονται κατά τον προσδιορισμό των τύπων. Για να επαληθεύσουμε την ορθότητα της τιμής ενός τύπου σε ένα κελί, ελέγχουμε εάν ο υπολογισμός είναι στα πλαίσια του αναμενόμενου αποτελέσματος. Ωστόσο, η αναμενόμενη

συμπεριφορά ενός προγράμματος λογιστικών φύλλων δεν είναι ρητά προσδιορισμένη.

Ο κύριος σκοπός ελέγχου ενός προγράμματος είναι η ανεύρεση σφαλμάτων στο πρόγραμμα. Για την επίτευξη αυτού του σκοπού χρειαζόμαστε συστηματικά σχεδιασμένες περιπτώσεις ελέγχου (χρησιμοποιώντας την κατάλληλη στρατηγική ελέγχου) οι οποίες αποκαλύπτουν λάθη στο πρόγραμμα. Εκτελώντας το πρόγραμμα με τις περιπτώσεις ελέγχου και συγκρίνοντας το αποτέλεσμα με το αναμενόμενο προϊόν, όπως αυτό έχει προδιαγραφεί ή όπως αυτό παράγεται από έναν χρησμοδοτικό έλεγχο[5], η παρουσία ενός σφάλματος μπορεί να ανιχνευθεί.

Η δημιουργία ενός ισχυρού χρησμοδότη, ωστόσο, προϋποθέτει την ύπαρξη προδιαγραφών. Εδώ όμως, δεν έχουμε ούτε τις απαιτούμενες προδιαγραφές, ούτε οι δημιουργοί λογιστικών φύλλων έχουν την υπομονή και την πραγματογνωμοσύνη να εκτελέσουν μία εκτενή διαδικασία περιπτώσεων ελέγχου. Για αυτό, μηχανισμοί πρέπει να επινοηθούν παράλληλοι σε ισχύ με αυτούς ενός ελεγκτικού χρησμοδότη ενώ δεν επιβαρύνουν την επιμέλεια και τη γνώση του δημιουργού τους σε περίπλοκες εξαρτήσεις. Για αυτό πρέπει να αναγνωρίσουμε ότι οι «ελεγκτές» προγραμμάτων λογιστικών φύλλων είναι οι τελικοί χρήστες οι οποίοι δεν γνωρίζουν τις θεωρίες ελέγχου και δεν αναμένεται από τους ίδιους να κάνουν έλεγχο με την παραδοσιακή του έννοια. Μάλλον, οι χρήστες προγραμμάτων λογιστικών φύλλων είναι εξαρτημένοι σε υψηλό βαθμό στη βοήθεια που παρέχει το σύστημα. Δεδομένου ότι η δομή του ελέγχου περιορίζεται στο περιεχόμενο του κελιού (και γενικώς χρησιμοποιείται μάλλον σπάνια σε σύγκριση με τα αλγοριθμικά προγράμματα) μας επιτρέπει να χρησιμοποιούμε περιοδική αριθμητική ως διαμεσολαβητή των υπηρεσιών ενός ισχυρού ελεγκτικού χρησμοδότη.

Βάση του στόχου του υπολογισμού και βλέποντας τις εισαγόμενες τιμές αναφερομένων κελιών σε ένα τύπο, ο χρήστης, παίρνοντας τον ρόλο του ανθρώπινου χρησμοδότη, προσδιορίζει το φάσμα του υπολογισμού του τύπου σε περιοδική μορφή με επιτρεπόμενες/αναμενόμενες τιμές.

[5] Ένας μηχανισμός που προβλέπει την αναμενόμενη συμπεριφορά ενός προγράμματος βάση των προδιαγραφών του.

Κάθε πραγματική τιμή που λαμβάνει κάθε κελί είναι μια διακριτική τιμή, είτε αυτή εισαχθεί από τον χρήστη, είτε είναι αποτέλεσμα υπολογισμού του προγράμματος του λογιστικού φύλλου. Για κάθε από αυτά τα κελιά, ένα φάσμα επιτρεπόμενων τιμών πρέπει να προσδιορίζεται. Αυτή η διαδικασία είναι πιο εύκολη από τη δημιουργία ελεγκτικών περιπτώσεων (μία πολύ πολύπλοκη διαδικασία ειδικά για χρήστες) που επιβάλλεται σε προστακτικά προγράμματα.

Ο χρήστης προσδιορίζει τα διαστήματα για αυτά τα εισαγόμενα κελιά τα οποία μπορεί να περιέχουν διαφορετικές τιμές. Τα κελιά που δεν περιέχουν διαφορετικές τιμές αντιπροσωπεύονται με διάστημα μηδενικού μήκους. Για αυτό, ένα κελί τύπου που υπόκειται σε έλεγχο, υπάρχουν δύο τιμές να υπολογισθούν και να συγκριθούν: μία τιμή που υπολογίζει το πρόγραμμα του λογιστικού φύλλου (d) βασισμένη σε τιμές των κελιών που αναφέρονται στον τύπο και ένα συνδετικό διάστημα (Β) υπολογιζόμενο από περιοδικό πρόγραμμα βασισμένο σε περιοδική αριθμητική και χρησιμοποιώντας περιοδικές τιμές των αναφερόμενων κελιών. Το περιοδικό πρόγραμμα είναι αντίστοιχο ενός προγράμματος λογιστικών φύλλων όπου οι τιμές των κελιών εκφράζονται ως διαστήματα και ο υπολογισμός γίνεται με την εφαρμογή περιοδικής αριθμητικής.

Για να συμπεραθεί η παρουσία σφάλματος σε ένα κελί τύπου, οι τρεις τιμές d, Ε, και Β που έχουν δημιουργηθεί από διαφορετικές πηγές πρέπει να συγκριθούν. Δύο περιπτώσεις προκύπτουν από τη σύγκρισή αυτή.

Περίπτωση 1η : $d \in E$ και $E \leq B$

Καθότι η υπολογισμένη περιοδική τιμή ενός τύπου περιορίζεται από ελάχιστη και μέγιστη τιμή του πιθανού υπολογισμού (αυτό είναι από προσδιορισμό περιοδική αριθμητική), η αναμενόμενη περίοδος πρέπει να τίθεται εντός των ορίων της υπολογισμένης περιόδου. Επίσης, η τιμή η οποία υπολογίσθηκε από το πρόγραμμα του λογιστικού φύλλου πρέπει να τίθεται εντός του αναμενόμενου μεγέθους του υπολογισμού. Συνεπώς, σε αυτή τη περίπτωση, μπορούμε να αναφέρουμε ότι δεν υπάρχει σύμπτωμα σφάλματος.

Περίπτωση 2η : d ∉ E ή E E⊄ B

Σε αυτή τη περίπτωση, υπάρχει ένδειξη συμπτώματος σφάλματος. Το σφάλμα μπορεί να έγκειται στον τύπο, ή στην αντίληψη του χρήστη ως προς τα αναμενόμενα αποτελέσματα. Ασφαλώς, ο έλεγχος διεξάγεται βάση της προϋπόθεσης ότι υπάρχει ορθή συμπεριφορά του προγράμματος έναντι της οποίας συγκρίνουμε το ακριβές αποτέλεσμα. Δεν μπορούμε όμως πάντα να βασιζόμαστε ότι η αναμενόμενη συμπεριφορά είναι ορθή.

Στη περίπτωση του **d ∉ E**, λόγω κάποιας λανθασμένης αναφοράς κελιών στον τύπο ή κάποιο άλλο σφάλμα, το πραγματικό αποτέλεσμα έχει αλλάξει από το αναμενόμενο. Στη δεύτερη περίπτωση **E ⊄ B**, τα λάθη επηρεάζουν τη συνδετική περίοδο που έχει υπολογισθεί για τον τύπο και δημιουργούν μία λανθασμένη ευθυγράμμιση μεταξύ του **E** και του **B**.

Αυτή η προσέγγιση κυρίως προορίζεται για το προσδιορισμό σφαλμάτων λόγω λάθος αναφοράς και λάθος προσδιορισμού περιοχής. Αυτά τα λάθη είναι αποτέλεσμα εσφαλμένου προσδιορισμού ή εσφαλμένης επιλογής ομάδας κελιών για τον σκοπό του υπολογισμού. Γενικά, μπορούμε να αναφερθούμε ότι αυτά τα σφάλματα οφείλονται σε έλλειψη προσδιορισμού σχεδίου για την επίτευξη ενός υπολογιστικού σκοπού. Λάθη αναφοράς και λάθη προσδιορισμού περιοχής δημιουργούν, κατά πάσα πιθανότητα, εσφαλμένη ευθυγράμμιση μεταξύ των υπολογιζόμενων τιμών από το πρόγραμμα λογιστικών φύλλων, το περιοδικό πρόγραμμα, και την αναμενόμενη περίοδο που έχει προσδιορίσει ο χρήστης. Επίσης, άλλα λάθη μπορεί να δημιουργήσουν μία ασυμφωνία στις τιμές **E** και **B** και μπορούν να ανιχνευθούν κατά τη διαδικασία. Όταν η ύπαρξη ενός προβλήματος σε ένα τύπο αναγνωρισθεί, η προέλευση του σφάλματος μπορεί να ανιχνευτεί με τη χρήση της σχέσης εξάρτησης δεδομένων μεταξύ κελιών που έχει δημιουργηθεί μέσω του τύπου.

Πρέπει να αναφερθεί ότι ο περιοδικός έλεγχος έχει διπλό ρόλο. Αναγνωρίζει σφάλματα σε στιγμιότυπα λογιστικών φύλλων, όταν οι πραγματικές τιμές **d** είναι εκτός της επιτρεπόμενης περιοχής. Επίσης, η σύγκριση μεταξύ **E** και **B** είναι μάλλον ένας έλεγχος

συνάφειας του αριθμητικού προτύπου του χρήστη. Αυτός ο έλεγχος μπορεί να είναι αρκετά ισχυρός και σε γενικότερο επίπεδο και όχι μόνο στο επίπεδο του συγκεκριμένου στιγμιότυπου του λογιστικού φύλλου.

4.3 Ρευστή Ορατή Αναπαράσταση των Δομών των Λογιστικών Φύλλων

Εδώ παρουσιάζουμε μια ομάδα τεχνικών που καθιστούν έμμεσες δομές των λογιστικών φύλλων ορατές και ευπρόσιτες, επιτρέποντας στους χρήστες να αλληλεπιδρούν άμεσα με την ορατή δομή της ροής δεδομένων, αντί να έχουν πρόσβαση στη ροή δεδομένων έμμεσα μέσω τύπων που περιέχουν κείμενο. Χρησιμοποιούμε γραφικές παραλλαγές (χρώμα, σκίαση, περίγραμμα, κλπ), κινούμενα σχέδια, και απλή διαδραστικότητα για την ορατοποίηση υποκείμενης δομής της ροής των δεδομένων, παράλληλα ελαχιστοποιώντας την ακαταστασία από περιττά στοιχεία στην οθόνη.

Οι πρώτες τρεις τεχνικές γραφικά μεγεθύνουν τα κελιά τύπων για να βοηθήσουν τους χρήστες να κατανοήσουν τις δομές ροής δεδομένων, χωρίς την επίπονη διαδικασία του ελέγχου κάθε κελιού για να αναγνωσθεί ο τύπος του. Η βασική ιδέα είναι να αποδώσουμε το γράφημα της ροής δεδομένων άμεσα στην οθόνη χωρίς να παρεμποδίσει αδικαιολόγητα με τη φυσική εμφάνιση του λογιστικού φύλλου. Η **παροδική τοπική όψη** επιτρέπει στους χρήστες να δουν τις δομές της ροής δεδομένων που σχετίζονται με ένα συγκεκριμένο κελί με απλό τρόπο. Η **στατική καθολική όψη** ορατοποιεί ολόκληρο το γράφημα της ροής δεδομένων ενός λογιστικού φύλλου επικαλύπτοντας την πινακοειδή διάταξη με ένα στατικό γράφημα. Η **κινούμενη καθολική εξήγηση** αυτόματα δημιουργεί και παρουσιάζει μία κινούμενη παράσταση της ροής δεδομένων μέσω των κελιών του λογιστικού φύλλου για να περιγράψει δομές που είναι δύσκολο να εμφανισθούν με τη στατική επίστρωση. Η **σημασιολογική πλοήγηση** επιτρέπει στους χρήστες να διασχίζουν τη δομή του λογιστικού φύλλου διαδραστικά βάση λογικής σύνδεσης με τις αναφορές των κελιών. Η **ορατή επεξεργασία** επιτρέπει στους χρήστες να δημιουργήσουν αποδοτικά γραφήματα ροής δεδομένων με τη χρήση, για παράδειγμα, άμεσου χειρισμού και προγραμματισμού. Ενώ η

σημασιολογική πλοήγηση και η ορατή επεξεργασία δεν είναι οι ίδιες ορατές αναπαραστάσεις, ωστόσο συμβάλλουν πολύ στην ορατή κατανόηση του γραφήματος ροής δεδομένων, μεγεθύνοτας τις τρεις πρώτες τεχνικές.

Αυτές τις τεχνικές τις αποκαλούμε *ρευστές* διότι έχουν διαδραστικό ύφος, και για την ορατή τους αναπαράσταση: πολλές αλληλεπιδράσεις γίνονται ομαλά με την απλή κίνηση του ποντικιού, η οποία εκφράζει το ενδιαφέρον του χρήστη, και η ορατοποίηση της δομής ροής δεδομένων ενσωματώνεται αρμονικά με την αρχική πινακοειδή διάταξη του λογιστικού φύλλου. Η προσέγγισή μας είναι να μεγεθύνουμε την υπάρχουσα πινακοειδή διάταξη και όχι να την αντικαταστήσουμε με μία διαφορετική ορατή αναπαράσταση. Έτσι ο χρήστης κατανοεί τους τύπους μέσα στα πλαίσια της κανονικής διάταξης του λογιστικού φύλλου.

Έχουμε εφαρμόσει αυτές τις διαδραστικές τεχνικές σε ένα πρότυπο πρόγραμμα λογιστικού φύλλου, με τη χρήση Pad++[6] και Python[7] που εκτελείται σε πλατφόρμες Unix. Αν και το πρότυπο σύστημα έχει περιορισμένες αποδόσεις, τα κινούμενα σχέδια γίνονται αργόστροφα σε λογιστικά φύλλα που περιέχουν περισσότερα από 20 X 20 κελιά, παρατηρήσαμε ότι οι ρευστές μας τεχνικές ορατοποίησης διευκολύνουν ουσιαστικά την κατανόηση των κρυφών δομών του λογιστικού φύλλου.

Το υπόλοιπο της εργασίας έχει οργανωθεί ως εξής. Πρώτον παρουσιάζουμε την τελευταία τεχνολογία στον τομέα της ορατοποίησης των λογιστικών φύλλων. Συνεχίζουμε με την αναλυτική παρουσίαση των τεχνικών ορατής αναπαράστασης που προτείνουμε. Και τελικά αναφέρουμε μερικές άλλες σχετικές εργασίες επί του θέματος και μία περίληψη της εργασίας.

4.3.1 Τεχνολογία

Λαμβάνοντας υπόψη τα προβλήματα που σχετίζονται με τους κρυμμένους τύπους, δεν είναι καθόλου παράξενο ότι πολλές προσπάθειες έχουν γίνει για να τους κάνουν πιο ορατούς. Παραδείγματος χάριν, η Microsoft Excel 97™, αναμφισβήτητα η πιο εμπλουτισμένη εφαρμογή λογιστικών φύλλων που υπάρχει, περιέχει

[6] Ένα σύστημα γραφικής διασύνδεσης που επιτρέπει εστίαση (zoom)
[7] Τύπος γλώσσας προγραμματισμού αντικειμενοστρεφής

δύο τεχνικές που παρέχουν περιορισμένη ορατή αναπαράσταση του γραφήματος ροής δεδομένων ενός συγκεκριμένου κελιού. Η πρώτη τεχνική, αποκαλούμενη « Ευρετής Περιοχής» (Range Finder), λειτουργεί με την επιλογή ενός κελιού που περιέχει έναν τύπο και επιλέγοντας το πλαίσιο του τύπου. Τότε οι διευθύνσεις που περιέχει ο τύπος χρωματίζονται και οι αντίστοιχες περιοχές τους στο λογιστικό φύλλο υπογραμμίζονται με χρωματιστά ορθογώνια, τα οποία μπορούν να μετακινηθούν και να ρυθμισθούν ώστε να γίνει η επεξεργασία του τύπου με άμεσο τρόπο χειρισμού.

Η δεύτερη τεχνική, αποκαλούμενη «Λογιστική Πληρότητα» (Auditing), σχεδιάζει βέλη από το επιλεγμένο κελί και προς τους προγόνους και απογόνους του. Επιπλέον, ο χρήστης μπορεί να πάει άμεσα σε ένα πρόγονο ή απόγονο με διπλό χτύπημα στην αιχμή του βέλους.

Αυτές οι τεχνικές προσπαθούν να παρουσιάσουν τις κρυμμένες δομές της ροής δεδομένων, αλλά πρέπει να ενεργοποιηθούν μέσω του μενού και κουμπιών των γραμμών εργαλείων, και περιορίζονται στην παρουσίαση ενός μεμονωμένου κελιού. Δεν υπάρχει η δυνατότητα να παρουσιάσουν ολόκληρη τη δομή του λογιστικού φύλλου. Ο χρήστης πρέπει να επιλέγει μεμονωμένα κάθε κελί για να δει τα χρωματιστά ορθογώνια του Ευρετή Περιοχής, σύνθετα λογιστικά φύλλα δημιουργούν μπερδεμένα βέλη, καθιστώντας δύσκολη τη κατανόηση των σχέσεων μεταξύ των κελιών.

Όπως και αυτά τα τμήματα του Excel, η δική μας εργασία εστιάζεται στην αναπαράσταση της δομής της ροής των δεδομένων. Αντίθετα με το Excel, εμείς θέλουμε να δείξουμε τοπικές και καθολικές όψεις τμημάτων της δομής. Επίσης, βελτιώνουμε την ορατοποίηση με βάση τις χωρικές σχέσεις της δομής ροής δεδομένων, χρησιμοποιούμε και τις δύο παροδικές τεχνικές (βλέπε την επόμενη ενότητα) και κινούμενα σχέδια για να παρουσιάσουμε με ρευστότητα τη δομή ροής δεδομένων.

4.3.2 Παροδική τοπική όψη

Η τεχνική της παροδικής τοπικής όψης επιτρέπει στον χρήστη να δει ένα τμήμα του γραφήματος ροής δεδομένων συνδεδεμένο με ένα κελί, το κελί που επεξεργάζεται ο χρήστης. Το σύστημα ορατοποιεί και τα εισαγόμενα κελιά (δηλαδή, αυτά που επηρεάζουν το τρέχον κελί, π.χ. εμπεριέχονται στον τύπο του) και τα εξαγόμενα κελιά

(δηλαδή, αυτά που επηρεάζονται από το τρέχον κελί, π.χ. εμπεριέχονται στον τύπο του). Το σύστημα κάνει ορατή διάκριση μεταξύ αυτών των δύο ειδών κελιών με τη χρήση χρώματος, πάχους γραμμών, ή άλλων γραφικών χαρακτηριστικών. Η εφαρμογή μας ομαδοποιεί εισαγόμενα κελιά σε χωρικά γειτονικές περιοχές και περιφράσσει κάθε περιοχή με ένα ορθογώνιο. Το ορθογώνιο είναι συνδεδεμένο με το τρέχον κελί με μία λεπτή γραμμή, ενώ τα εξαγόμενα κελιά είναι σκιασμένα. Η ομαδοποίηση των εισαγόμενων κελιών κάνει χρήση και της λογικής δομής του λογιστικού φύλλου (είναι εισαγόμενα στο τρέχον κελί) και της φυσικής δομής του λογιστικού φύλλου (είναι γειτονικά στο χώρο).

Ο λόγος που αποκαλούμε αυτή τη τεχνική *παροδική* βασίζεται στον τρόπο που ο χρήστης προσδιορίζει το τρέχον κελί. Σε συμβατικές εφαρμογές λογιστικών φύλλων, ο χρήστης πρέπει να κινήσει τον δρομέα στο κελί που τον ενδιαφέρει και να το επιλέξει για να δει τον τύπο του.

Στο δικό μας σύστημα ο χρήστης επιλέγει το τρέχον κελί απλώς με τη κίνηση του δρομέα του ποντικιού επάνω στο κελί. Όταν ο δρομέας φθάνει στο κελί, το γράφημα ροής δεδομένων του συγκεκριμένου κελιού βαθμιαία εμφανίζεται στην οθόνη, και βαθμιαία σβήνεται όταν ο δρομέας απομακρύνεται από το κελί. Έτσι, ο χρήστης έχει τη δυνατότητα να εξερευνήσει το γράφημα ροής δεδομένων της δομής του λογιστικού φύλλου με την απλή κίνηση του δρομέα στην επιφάνεια του λογιστικού φύλλου.

Αυτή η διαδραστικότητα της «κίνησης του ποντικιού» βρίσκεται και σε άλλα συστήματα. Ένα αξιόλογο παράδειγμα είναι το "ToolTips," της Microsoft τα οποία είναι μικρά παράθυρα που περιέχουν επεξηγηματικά κείμενα. Τα "ToolTips," εμφανίζονται όταν ο δρομέας παραμένει πάνω σε ένα στοιχείο. Ωστόσο, τα "ToolTips," είναι ξεχωριστά από τα στοιχεία που επεξηγούν και τα επικαλύπτουν, ενώ η δική μας παρουσίαση του γραφήματος της ροής δεδομένων είναι ενσωματωμένη στην αρχική όψη του λογιστικού φύλλου, χωρίς να καλύπτει το λογιστικό φύλλο αλλά και χωρίς να είναι ενοχλητική. Η βαθμιαία παρουσίαση/σβήσιμο έχει σημαντικό ρόλο στην ενσωμάτωση - εμφανίζει μεγαλύτερες ορατές δομές όταν ο χρήστης κινεί τον δρομέα στην επιφάνεια του λογιστικού φύλλου. Χωρίς αυτό το εφέ του ξεθωριάσματος και τον προσεκτικό συγχρονισμό, θα είχαμε ένα συνεχές αναβόσβημα των πληροφοριών ροής δεδομένων. Ειδικά, όταν το γράφημα της ροής δεδομένων είναι εκτεταμένο, θα ήταν πολύ ενοχλητικό να βλέπει κανείς μεγάλα γραφήματα να εμφανίζονται και να εξαφανίζονται συχνά.

Αυτή η μορφή της παροδικής αλληλεπίδρασης αναστέλλεται κατά τη διάρκεια μερικών διαδικασιών, όπως η επεξεργασία, ώστε η κίνηση του δρομέα να μην επιφέρει αθέμιτες ορατοποιήσεις.

4.3.3 Στατική καθολική όψη

Η παροδική τοπική όψη έχει πολλά προτερήματα. Δεν απαιτεί ειδικές λειτουργίες, έτσι ώστε η προσοχή του χρήστη να παραμένει στο περιεχόμενο του λογιστικού φύλλου όπως απλά κινεί τον δρομέα για να προκαλέσει τοπικές ορατές αναπαραστάσεις. Δεν φορτώνει την οθόνη με περιττά στοιχεία διότι περιορίζεται στο γράφημα ροής δεδομένων το οποίο συνδέεται με ένα κελί. Επίσης, το εφέ του ξεθωριάσματος βοηθά το χρήστη να έχει μια καθολική όψη της δομής του λογιστικού φύλλου με έναν ευχάριστο, μη-ενοχλητικό τρόπο.

Εντούτοις, υπάρχουν επίσης πολλές περιπτώσεις όπου ο χρήστης μπορεί να θέλει να δει ολόκληρη τη δομή ταυτόχρονα. Για παράδειγμα, ο χρήστης θέλει γρήγορα να επιθεωρήσει ολόκληρη τη δομή όταν του δοθεί ένα λογιστικό φύλλο που δημιουργήθηκε από άλλο χρηστή.

Εφαρμόσαμε την **στατική καθολική όψη** ακριβώς για αυτή τη συνολική δομική πληροφόρηση. Με αυτή τη μορφή, όλες οι σχέσεις της ροής δεδομένων εμφανίζονται στην οθόνη - δηλαδή κάθε εισαγόμενο και εξαγόμενο κελί για κάθε κελί του λογιστικού φύλλου.

Όπως και στις παροδικές τοπικές όψεις, τα κελιά παρουσιάζονται ως ομάδες, το οποίο χρησιμεύει για την ελλάτωση της αταξίας και επίσης συνοψίζει τη ροή δεδομένων.

Ακόμη και με την τεχνική ομαδοποίησης, η επικάλυψη πολλών τμημάτων του γραφήματος είναι αναπόφευκτη, και καθιστά δύσκολη τη παρατήρηση της λεπτομερούς ροής δεδομένων για συγκεκριμένα κελιά. Αντί' αυτού, η στατική καθολική όψη είναι χρήσιμη για μία γενική επισκόπηση ολόκληρης της δομής. Στη δική μας εφαρμογή, ορίζουμε διαφορετικά χρώματα για κάθετες, οριζόντιες και μονό-κελικές ομάδες δομών. Κατά συνέπεια, ο χρήστης μπορεί γρήγορα να διακρίνει τη δομή του λογιστικού φύλλου. Η καθολική όψη είναι

επίσης πολύτιμη κατά την επεξεργασία της δομής της ροής δεδομένων.

4.3.4Κινούμενη καθολική εξήγηση

Η στατική καθολική όψη λειτουργεί καλά για να απεικονίσει την γενική δομή του λογιστικού φύλλου, ιδιαίτερα όταν το γράφημα ροής δεδομένων έχει ένα κανονικό σχέδιο. Εντούτοις, για λογιστικά φύλλα όπου το κάθε κελί περιλαμβάνεται σε πολλούς τύπους, ή όπου το γράφημα επικαλύπτει με συγκεχυμένο τρόπο, η στατική καθολική όψη μπορεί να μην φανεί πολύ χρήσιμη. Επίσης, η στατική καθολική όψη δεν παρουσιάζει αποδοτικά τη γενική κατεύθυνση της ροής δεδομένων.

Για να αντιμετωπίσουμε αυτά τα ζητήματα, χρησιμοποιούμε κινούμενα σχέδια για να ξετυλίξουμε την ιστορία της δομής του λογιστικού φύλλου. Η ροή των τιμών των κελιών ξεκινά από τα αρχικά κελιά (που περιέχουν μόνο δεδομένα), ενώνονται και χωρίζουν σε ενδιάμεσα κελιά, και τελειώνει στα τελικά κελιά (συνήθως, που περιέχουν το αποτέλεσμα του υπολογισμού). Αυτή την ακολουθία της αποκαλούμε αφηγηματική έκφραση του γραφήματος ροής δεδομένων, η **κινούμενη καθολική εξήγηση** παρουσιάζει την ιστορία σαν μία σειρά από κινούμενα σχέδια. Αρχικά εξετάζει τους τύπους για την ανάλυση των εξαρτήσεων των κελιών, μετά προσδιορίζει μία διάταξη για το γράφημα, και τελικά εμφανίζει τη ροή δεδομένων ως μία σειρά κινουμένων σχεδίων όπου οι κόμβοι στην αρχή της ροής δεδομένων ζωντανεύουν πρώτοι, και ακολουθούν άλλα κινούμενα σχέδια διαδοχικά για κάθε κελί. Ως αποτέλεσμα, ο χρήστης βλέπει μία ορατή αναπαράσταση της πορείας των υπολογισμών εντός του γραφήματος.

Με την επιλογή της κινούμενης διάταξης της εξήγησης, χρησιμοποιούμε πληροφορίες και από το υποκείμενο γράφημα ροής δεδομένων και από την πινακειδή διάταξη της δομής. Οι εξαρτήσεις των κελιών είναι η βάση της διάταξης αυτής της κίνησης, αλλά μπορεί να υπάρχουν διάφορες πιθανότητες για ταυτόχρονη κίνηση διαφορετικών τμημάτων του γραφήματος. Επίσης, χρησιμοποιούμε πληροφορίες σχετικές με τη γραφική διάταξη του λογιστικού φύλλου για να αποφασίσουμε ποια από τα σχέδια θα κινήσουμε ταυτόχρονα. Ακριβώς όπως, οι παροδικές τοπικές και οι στατικές καθολικές όψεις χρησιμοποιούν χωρικές σχέσεις για την ομαδοποίηση των κελιών ώστε να αποδώσουν μία αποτελεσματικότερη ορατή

αναπαράσταση, έτσι και η κινούμενη καθολική εξήγηση χρησιμοποιεί τις χωρικές σχέσεις για να ομαδοποίηση τα κινούμενα σχέδια για να εκτελέσουν μία αποτελεσματικότερη ορατή αναπαράσταση. Ο σκοπός είναι η παραγωγή μιας γενικής κίνησης σχεδίων που οργανώνει αποτελεσματικά τη ροή δεδομένων σε κατανοητά τμήματα. Ροές με κανονική χωρική δομή η κίνησή του είναι ταυτόχρονη. Ο τρέχον αλγόριθμος του προγράμματος προσπαθεί να εντοπίσει τέτοιες παράλληλες ροές δεδομένων και αν τις ομαδοποιήσει.

4.3.5 Σημασιολογική πλοήγηση

Η λειτουργία «Λογιστικής Πληρότητας» (Auditing) του Excel επιτρέπει στον χρήστη να κινείται σε χωρικά ασύνδετα, αλλά λογικά συνδεδεμένα κελιά με διπλό χτύπημα στην αιχμή του βέλους, τη τεχνική αυτή την αποκαλούμε *σημασιολογική πλοήγηση*. Στην σημασιολογική πλοήγηση, οι προτροπές για την πλοήγησης δίνονται από τις σημασιολογικές σχέσεις των κελιών και όχι από την επιφανειακή χωρική τους συνέχεια. Βελτιώσαμε αυτή τη διασύνδεση της σημασιολογικής πλοήγησης για να επιτύχουμε ευκολότερη πρόσβαση σε κρυμμένες δομές με την εισαγωγή λειτουργιών του πληκτρολογίου και εκφραστικά κινούμενα σχέδια.

Επιλέξαμε τη χρήση του πληκτρολογίου για τη σημασιολογική πλοήγηση, διότι έχει τη δυνατότητα να βελτιώσει τον κύκλο « επεξεργασία κελιού/πλοήγηση σε νέο κελί». Η εφαρμογή έχει ως εξής:

Α) ένα βέλος αναπτύσσεται όταν πιέζουμε το πλήκτρο «control», με κατεύθυνση ένα από τα σημασιολογικά συνδεδεμένα κελιά (εισαγόμενα ή εξαγόμενα), το οποίο κελί το αποκαλούμε κελί προορισμού.

Β) ο χρήστης μπορεί να αλλάξει την κατεύθυνση το βέλους με τη χρήση των πλήκτρων αριστερά/δεξιά.← →

Γ) ο δείκτης του τρέχοντος κελιού μετακινείται στο κελί προορισμού όταν πιέσουμε το πλήκτρο πάνω .↑

Δ) ο χρήστης έτσι συνεχίζει τη πλοήγηση από το κελί προορισμού.

Η πλοήγηση είναι δυνατή με τη χρήση των πλήκτρων που φέρουν τα βέλη, ενώ πιέζουμε το πλήκτρο «control». Όταν αφήνουμε το πλήκτρο «control» ο δρομέας επιστρέφει στη κανονική του λειτουργία για κανονική χωρική πλοήγηση. Κάθε οπτική επίδραση

παρουσιάζεται με κινούμενο τρόπο για να βοηθήσει τον χρήστη να κατανοήσει την πλοήγηση. Η αιχμή του βέλους κινείται συνεχώς και ο δείκτης του τρέχοντος κελιού μετακινείται πολύ γρήγορα (πετάει) και ομαλά. Αυτή η σημασιολογική πλοήγηση όχι μόνο μειώνει τον αριθμό πληκτρολογήσεων, αλλά επίσης προωθεί καλύτερη κατανόηση της δομής ροής των δεδομένων. Εύκολη πρόσβαση σε κελιά με λογική σύνδεση κάνει τον χρήστη να αισθάνεται ότι τα κελιά βρίσκονται κοντά το ένα με το άλλο, ακόμη και όταν είναι χωρικά απομακρυσμένα. Με άλλα λόγια, η σημασιολογική πλοήγηση ορατοποιεί την κρυμμένη τοπολογία του γραφήματος ροής δεδομένων μέσω της χαρακτηριστικής της αλληλεπίδρασης.

4.3.6 Ορατή επεξεργασία

Στις προηγούμενες ενότητες αναφερθήκαμε στις τεχνικές ορατής αναπαράστασης της υποκείμενης δομής της ροής δεδομένων ενός λογιστικού φύλλου. Σε αυτή την ενότητα, θα παρουσιάσουμε τεχνικές ορατής επεξεργασίας της δομής της ροής δεδομένων με τους υποκείμενους κρυμμένους τύπους τους. Αυτή η λειτουργική ορατή επεξεργασία έχει εκπληκτικά αποτελέσματα στη δημιουργία δομών ροής δεδομένων οι οποίες είναι χωρικά τακτικές και απλές, αλλά σύνθετες και περίπλοκες για να προσδιορισθούν με κείμενο. **Άμεση επεξεργασία του γραφήματος της ροής δεδομένων** επιτρέπει στον χρήστη να επεξεργαστεί τις δομές της ροής δεδομένων με έναν παρεμφερή τρόπο αυτού των αντικειμενοστρεφών επεξεργαστών σχεδίου, και αλληλεπιδρόμενη γραφική εισαγωγή επιτρέπει στον χρήστη να δομήσει αλληλεπιδραστικά κανονικούς τύπους για μία σειρά κελιών.

Άμεση επεξεργασία γραφημάτων ροής δεδομένων

Άμεση επεξεργασία γραφημάτων ροής δεδομένων είναι μία απλή εφαρμογή ορατής αναπαράστασης του γραφήματος ροής δεδομένων. Επιτρέπει στο χρήστη να μετακινεί, να διαβαθμίζει, και να διαγράφει τα ορατοποιημένα γραφήματα με τη χρήση απλών άμεσων τεχνικών χειρισμού. Αυτές οι λειτουργίες επεξεργασίας αναγκάζουν

τους κειμενικούς τύπους να ενημερωθούν αναλόγως. Είναι ιδιαίτερα χρήσιμο να επεξεργαζόμαστε πολλαπλά γραφήματα ροής δεδομένων ταυτόχρονα (το Excel επιτρέπει γραφική επεξεργασία, αλλά για μόνο μία περιοχή κάθε φορά).

Όταν η στατική καθολική όψη είναι εν λειτουργία, ή όταν ο χρήστης εισέρχεται σε κατάσταση επεξεργασίας επιλέγοντας ένα κελί στην κατάσταση παροδικής ορατοποίησης, τότε τα γραφήματα ροής δεδομένων γίνονται ορατά. Ο χρήστης μπορεί να επιλέξει ένα γράφημα χτυπώντας το, να το μετακινήσει σύροντας την επιλογή του και να την διαβαθμίσει σύροντας τις λαβές που έχει στις γωνίες του. Πολλαπλά γραφήματα μπορεί να επιλεχθούν χτυπώντας τα και παράλληλα πιέζοντας το πλήκτρο «shift», έτσι ταυτόχρονα μετακινώντας ή διαβαθμίζοντας όλα τα επιλεγμένα γραφήματα.

Αλληλεπιδρώμενες γραφικές εισαγωγές

Η ορατή αναπαράσταση των γραφημάτων ροής δεδομένων καθιστά δυνατή την κατανόηση της δομής ως ένα σύνολο τακτικών σχεδίων. Σε αυτή την ενότητα θα περιγράψουμε την προσέγγισή μας για να κατασκευάσουμε αυτά τα τακτικά σχέδια με τη χρήση της εισαγωγής. Σε τρέχοντα λογιστικά φύλλα ο χρήστης πρέπει να εισάγει το σύμβολο $ για να προσδιορίσει απόλυτες αναφορές, και μετά να εκτελέσει σωστά λειτουργίες γεμίσματος για να δημιουργήσει την επιθυμητή δομή των σχεδίων ροής δεδομένων. Το σύμβολο $ υποδεικνύει ότι η επόμενη παράμετρος δεν αλλάζει κατά τη διαδικασία του γεμίσματος, ενώ οι άλλες παράμετροι αλλάζουν σύμφωνα με τη σχετική τους θέση ως προς τα γεμισμένα κελιά. Για παράδειγμα, αν ο τύπος του B2 είναι =A2*B$1 και ο χρήστης γέμισε την περιοχή B2-D4 με το ίδιο τύπο, τότε ο τύπος του D4 γίνεται =$A4*D$1.

Αυτή η μεθόδευση συχνά αποτυγχάνει, ακόμη και σε απλές περιπτώσεις είναι δύσκολο να δημιουργηθούν και να κατανοηθούν τύποι με τα σύμβολα $. Ο Hendry πρότεινε μία τεχνική προγραμματισμού με παραδείγματα για να επιλύσει αυτό το πρόβλημα. Η προσέγγισή του απαιτεί από τον χρήστη να πληκτρολογήσει τους πρώτους δύο τύπους, να προσδιορίσει την περιοχή για γέμισμα, και να εκτελέσει μία διαταγή από το μενού για να αρχίσει το γέμισμα. Έδειξε ότι δύο παραδείγματα είναι επαρκή για την έκφραση κοινών δομών λογιστικών φύλλων. Τύποι για τα γεμισμένα κελιά μπορούν αυτόματα να υπονοηθούν από τη σχετική θέση των κελιών και τη

διαφορά τιμών των παραμέτρων μεταξύ των δύο παραδειγμάτων. Αν και δεν χρησιμοποιεί αυτόν τον όρο, έχει ουσιαστικά προτείνει το γέμισμα των κελιών με τη χρήση της *εισαγωγής* στη δομή των δύο πρώτων τύπων.

Η δική μας τεχνική **αλληλεπιδρωμένης γραφικής εισαγωγής** επεκτείνει την προσέγγισή του με την χρήση γραφικών παραδειγμάτων αντί κειμενικών τύπων για την εισαγωγή. Το σύστημα επιτρέπει στον χρήστη να επεξεργαστεί γραφικά τα δύο παραδείγματα, και μετά να ορατοποιήσει το αποτέλεσμα της λειτουργίας γεμίσματος.

Η τρέχουσα εφαρμογή μας λειτουργεί ως εξής:

1. πληκτρολογούμε ένα τύπο στο κελί που ξεκινά την περιοχή του γεμίσματος,

2. επιλέγουμε την περιοχή και εκτελούμε διαταγή γεμίσματος,

3. το σύστημα παρουσιάζει μία γραφική πρόταση του δεύτερου σταδίου της εισαγωγής, το οποίο μπορούμε να χειριστούμε γραφικά στη σωστή του θέση,

4. πιέζουμε το κουμπί επιβεβαίωσης, «confirm», για να εκτελεστεί η λειτουργία του γεμίσματος.

Η βασική ιδέα βρίσκεται στο 3: όπου το σύστημα προτρέπει τον χρήστη να προσδιορίσει γραφικά το δεύτερο παράδειγμα. Ως συνέπεια αυτής της αλληλεπιδραστικής προσέγγισης, ο χρήστης δεν χρειάζεται πλέον να προετοιμάζει προσεκτικά κειμενικούς τύπους πριν την λειτουργία του γεμίσματος. Το μόνο που πρέπει να κάνει ο χρήστης είναι να απαντήσει στο αίτημα του συστήματος με επιβεβαίωση, και φυσικά μετακινώντας τα γραφικά κουτιά στη σωστή τους θέση. Επιπλέον, η γραφική αναπαράσταση των δύο πρώτων παραδειγμάτων και τον επόμενων γεμισμένων κελιών επίσης βοηθά στην κατανόηση της δομής του γραφήματος ροής δεδομένων.

4.3.7 Σχετικές εργασίες

Η επιτυχία του λογιστικού φύλλου ως εύχρηστο υπολογιστικό περιβάλλον, έχει οδηγήσει στη δημιουργία αρκετών προγραμμάτων για τελικούς χρήστες βασισμένα σε διασυνδέσεις της μορφής του λογιστικού φύλλου. Το σύστημα ACE επέκτεινε τη βασική ιδέα των λογιστικών φύλλων, με μία πινακοειδή διάταξη εμπλουτισμένη με

σημεία κειμένου, και την εφάρμοσε για την ανάπτυξη αλληλεπιδραστικών γραφικών εφαρμογών. Το σύστημα NoPumpG εφαρμόζει την ιδέα της αυτόματης συντήρησης προδιαγεγραμμένων σχέσεων μεταξύ κελιών του λογιστικού φύλλου για τον έλεγχο γραφικών αναπαραστάσεων. Το C32 χρησιμοποιεί τη διασύνδεση του λογιστικού φύλλου για τη δόμηση περιοριστικών πλαισίων σε κουτιά εργαλείων για χρήστες. Το σύστημα Forms/3 είναι μία γλώσσα ορατού προγραμματισμού γενικής χρήσης βασισμένη στο παράδειγμα του λογιστικού φύλλου. Οι Yang κ.α. προτείνουν ένα σχέδιο με συγκριτικές μετρήσεις επιδόσεων για τον οπτικό προγραμματισμό

Συμπεριλαμβανομένης της στατικής απεικόνισης ροής δεδομένων.

Τα συστήματα Toolglass και Magic Lenses χρησιμοποιούν φακούς χωρικά αφιερωμένους για την ορατοποίηση και αλληλεπίδραση με τα υποκείμενα στρώματα πληροφοριών. Σε αντίθεση με τη ορατή αναπαράσταση του Magic Lenses, η ρευστή ορατοποίηση ελέγχει όλη την περιοχή με συγχρονισμένο τρόπο, και παρέχει μία καλυτέρα ενσωματωμένη διασύνδεση.

Διασυνδέσεις εστίασης (Zooming), και τεχνικές διαστρέβλωσης εστίασης και συγκρότησης είναι προσπάθειες παρουσίασης μεγάλων περιοχών πληροφοριών εντός του περιορισμένου χώρου της οθόνης. Προσπαθούν να παρουσιάσουν ένα εστιακό τμήμα της περιοχής πληροφοριών και να διατηρήσουν την περιβάλλουσα συγκρότηση. Για παράδειγμα, η διαδικασία για την ορατοποίηση των λογιστικών φύλλων χρησιμοποιεί εστίαση και συγκρότηση για να ορατοποιήσει μεγάλες πινακοειδείς διατάξεις. Η διαφορά έγκειται στο ότι η δική μας εστίαση είναι στην ορατοποίηση των κρυμμένων, πίσω από την πινακοειδή διάταξη, δομών της ροής των δεδομένων, και όχι η ίδια η διάταξη.

Πολλά σύγχρονα συστήματα διασύνδεσης ενσωματώνουν κινούμενα σχέδια. Τα κινούμενα σχέδια προσδίδουν ενότητα, συνέχεια, και πραγματική ύπαρξη στα οπτικά αντικείμενα. Βοηθούν το χρήστη να κατανοήσει οπτικά γεγονότα με την αποφυγή απότομων αλλαγών στην οθόνη. Είναι αναπόφευκτα εργαλεία για την αναπαράσταση σειράς γεγονότων με ένα διαισθητικό τρόπο (π.χ. αλγοριθμικά κινούμενα σχέδια). Σε αυτή την εργασία, προτείναμε άλλη μία χρήσιμη εφαρμογή των κινούμενων σχεδίων: την ορατή αναπαράσταση αόρατων πληροφοριών. Είναι δύσκολο να παρουσιάσουμε σε στατική μορφή αναπαράστασης τις κρυμμένες δομές της ροής δεδομένων. Τα κινούμενα σχέδια, όμως, μας δίνουν αυτή τη δυνατότητα.

Η τεχνική μας της γραφικής εισαγωγής μπορεί να θεωρηθεί ως εφαρμογή του προγράμματος επίδειξη / παράδειγμα. Το Metamouse ανιχνεύει επαναλήψεις στις γραφικές επεξεργασίες του χρήστη και προτείνει την επόμενη λειτουργία. Το Eager αναλύει την ακολουθία μιας λειτουργίας HyperCard, και προτείνει το επόμενο σύνολο λειτουργιών. Τα συστήματα Chimera και IMAGE βρίσκουν κατάλληλους γεωμετρικούς περιορισμούς από γραφικά παραδείγματα.

Τα λογιστικά φύλλα περιέχουν κρυμμένα γραφήματα ροής δεδομένων εκτός από τα επιφανειακές πινακοειδείς διατάξεις τους. Παρουσιάσαμε ένα σύνολο τεχνικών καθιστούν την δομή της ροής δεδομένων ορατή και προσιτή, διατηρώντας όμως, την αρχική μορφή του λογιστικού φύλλου. Ο στόχος αυτών των τεχνικών είναι να μεταδοθεί μία καλύτερη κατανόηση της δομής ροής δεδομένων με τον χρήστη να έχει τη δυνατότητα ορατής αλληλεπίδρασης με τις κρυμμένες δομές.

Εφαρμόσαμε αυτές τις τεχνικές στο πρότυπο λογιστικού φύλλου που προγραμματίσαμε, και παρατηρήσαμε ότι αυξάνουν σημαντικά την κατανόηση των λογιστικών φύλλων. Βασικές αρχές στη προσέγγισή μας είναι προσεκτικά σχεδιασμένες γραφικές ενσωματώσεις των πληροφοριών της ροής δεδομένων με τη διάταξη του λογιστικού φύλλου, παρουσίαση των γραφικών αλλαγών με κινούμενα σχέδια, και απλή αλληλεπίδραση. Αποδείξαμε την αποδοτικότητα της βαθμιαίας παρουσίασης / σβήσιμο των κινουμένων σχεδίων - ο χρήστης ενημερώνεται για την ύπαρξη του κρυμμένου γραφήματος της ροής δεδομένων με τρόπο ανεπαίσθητο.

Μελλοντικές εργασίες θα εστιάζονται στο θέμα της ενσωμάτωσης με πιο ρεαλιστικά παραδείγματα λογιστικών φύλλων. Όπως, ήδη έχουμε αναφέρει, το πρότυπό μας λογιστικού φύλλου συντάχθηκε με τη χρήση των συστημάτων Pad++ και Python εκτελούμενα σε πλατφόρμα Unix. Αν και Pad++ παρέχει ισχυρά πρωτόγονα για διαβάθμιση και κινούμενα σχέδια και για την γρήγορη εξερεύνηση νέων ιδεών, όμως επιβαρύνει την αποδοτικότητα. Ως αποτέλεσμα, ομαλή παρουσίαση των κινούμενων σχεδίων είναι δυνατή μόνο για λογιστικά φύλλα με 400 κελιά ή λιγότερα.

Η Margaret Burnett συνεχίζει να αναπτύσσει μία μεθοδολογία για ορατό έλεγχο η οποία θα συμπληρώνει το ορατό περιβάλλον προγραμματισμού που χρησιμοποιούν τα λογιστικά φύλλα.

Ο Takeo Igarashi παρουσίασε μία σειρά από τεχνικές για την αντιμετώπιση των κρυμμένων γραφημάτων ροής δεδομένων και επιφανειακές πινακοειδής διάταξης λογιστικών φύλλων. Έχουν σχεδιασθεί για να βοηθήσουν τον χρήστη να κατανοήσει καλύτερα τη δομή ροής δεδομένων και να έχει τη δυνατότητα ορατής αλληλεπίδρασης με τις σκοτεινές, ασαφείς δομές. Επίσης αναφέρει ότι πρέπει τα διαγράμματα αυτά να ενσωματωθούν σε ένα πιο ρεαλιστικό πρόγραμμα για λογιστικά φύλλα. Εμείς αυτό το θέμα το καλύψαμε επιτρέποντα στα διαγράμματα να δημιουργηθούν ελεύθερα από την εφαρμογή.

Ο Jorma Sajaniemi παρουσίασε ένα θεωρητικό πρότυπο λογιστικών φύλλων μαζί με τις περιγραφές διάφορων μηχανισμών κριτικού ελέγχου λογιστικών φύλλων οι οποίοι χρησιμοποιούν το πρότυπο. Τα εργαλεία που εμείς παρουσιάσαμε έχουν την δυνατότητα να υποστηρίζουν αυτούς τούς μηχανισμούς.

Για να κλιμακώσουμε τις εμπειρίες μας σε μεγαλύτερα λογιστικά φύλλα έχουμε να αντιμετωπίσουμε τρία ζητήματα: αποδοτικότητα των κινούμενων σχεδίων, αποδοτικότητα των αλγορίθμων, και ορατοποίηση και πλοήγηση σε μεγαλύτερες δομές λογιστικών φύλλων. Είμαστε σίγουροι ότι μία κατά παραγγελία εφαρμογή θα υποστήριζε άνετα κινούμενα σχέδια για μεγαλύτερα λογιστικά φύλλα. Οι αλγόριθμοι που υποστηρίζουν τις διάφορες τεχνικές μας βασίζονται σε παραδοσιακούς υπολογισμούς ροής δεδομένων, οι οποίοι είναι υπολογιστικά πειθήνιοι για μεγαλύτερα λογιστικά φύλλα. Ορατοποίηση και πλοήγηση σε μεγαλύτερα λογιστικά φύλλα, όπου κελιά αναφέρονται σε απομακρυσμένα κελιά εκτός των ορίων της οθόνης, είναι ένα πιο ουσιαστικό ζήτημα. Αναμένουμε ότι με τεχνικές όπως η αναπαράσταση με κινούμενα σχέδια και οι σημασιολογικές πλοηγήσεις η χρησιμότητα των οποίων σε μεγαλύτερα λογιστικά φύλλα θα αποδειχθεί ιδιαίτερα χρήσιμη. Σε αυτές τις περιπτώσεις μπορεί να απαιτηθεί η χρήση μηχανισμών αυτόματης κάμερας, όπως η απεστίαση για ευρύτερο πλαίσιο, φιλτράρισμα ενός λογικά σχετιζόμενου κελιού, και εστίαση για καλύτερη ανάλυση.

Η Ρευστή ορατή αναπαράσταση των λογιστικών φύλλων είναι μια ιδιαίτερη εφαρμογή της γενικότερης ομάδας τεχνικών με ρευστές διασυνδέσεις , σκοπός των οποίων είναι να παρέχουν απλή, συμφραζόμενη και κινούμενη πρόσβαση προς ένα δευτερεύον υπόστρωμα περιεχομένου ενώ διατηρεί την εμφάνιση του αρχικού υλικού. Ρευστές τεχνικές επιτρέπουν στον χρήστη να μετατοπίζει την προσοχή του από το αρχικό στο δευτερεύον περιεχόμενο, καθώς το σύστημα ρευστά αλλάζει την απεικόνιση του και επιδεικνύει κρυμμένο δευτερεύον περιεχόμενο στο πλαίσιο του συσχετισμένου του

αρχικού περιεχομένου. Άλλοι τομείς στους οποίους έχουμε εξερευνήσει τη ρευστή διασύνδεση του χρήστη είναι η σύνδεση Υπερκειμένου (hypertext) και επισημειώσεις. Άλλες πιθανές εφαρμογές ρευστής ορατοποίησης είναι οι χάρτες, τα διαγράμματα του CAD, και άλλων γραφικών τομέων με πλούσιες υποκείμενες δομές. Η μέχρι τώρα εμπειρία μας προτείνει ότι οι ρευστές τεχνικές αποτελούν ένα ισχυρό βοήθημα για τους χρήστες σε μία μεγάλη ποικιλία εφαρμογών που απαιτούν κατανόηση και αλληλεπίδραση με τα δεδομένα.

Βιβλιογραφία

[1] R. Abraham and M. Erwig. Inferring Templates from Spreadsheets. In *28th IEEE Int. Conf. on Software Engineering*, pages 182–191, 2006.

[2] R. Abraham and M. Erwig. AutoTest: A Tool for Automatic Test Case Generation in Spreadsheets. In IEEE Int. Symp. on Visual Languages and Human-Centric Computing, pages 43–50, 2006.

[3] R. Abraham and M. Erwig. UCheck: A Spreadsheet Unit Checker for End Users. Journal of Visual Languages and Computing, 18(1):71–95, 2007.

[4] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A Type System for Statically Detecting Spreadsheet Errors. In *18th IEEE Int. Conf. on Automated Software Engineering*, pages 174–183, 2003.

[5] T. Antoniu, P. A. Steckler, S. Krishnamurthi, E. Neuwirth, and M. Felleisen. Validating the Unit Correctness of Spreadsheet Programs. In *26th IEEE Int. Conf. on Software Engineering*, pages 439–448, 2004.

[6] Ed Asher, "Glitch scrambles city election tally - computer error changes none of the outcomes," *Albuquerque Tribune*,

November 8, 2003, http://web.abqtrib.com/archives/news03/110803_news_canvass.shtml.

[7] Yirsaw Ayalew, Markus Clermont, and Roland Mittermeir (2000). *Detecting Errors in Spreadsheets* . In Spreadsheet Risks,Audit and Development Methods, (1):51-62

[8] Baker, J.E., and Sugden, S.J. (2003). Spreadsheets in Education: The First 25 Years. Spreadsheets in Education 1(1): 18—43.

[9] Brethour, P. (2003). Human error costs TransAlta $24-million on contract bids. Globe and Mail, Toronto, Wednesday, Jun. 4, 2003 (internet edition from http://www.bpm.ca/TransAlta.htm).

[10] L Beckwith, M Burnett, and C Cook. Reasoning about many-to-many requirement relationships in spreadsheets. In HCC'02 [HCC02], pages 149–157.

[11] Polly Brown and John Gould. An experimental study of people creating spreadsheets. *Transactions on Office Information Systems*, 5(3):258-272, July 1987.

[12] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein and S. Yang (2001) „Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm", Journal of Functional Programming 11 (2), pages 155-206.

[13] M. M. Burnett, A. Sheretov, B. Ren, and G. Rothermel. Testing Homogeneous Spreadsheet Grids with the "What You See Is What You Test" Methodology. IEEE Transactions on Software Engineering, 29(6):576-594, 2002.

[14] Ray Butler, "The Role of Spreadsheets in the Allied Irish Banks / Allfirst Currency Trading Fraud," *Proceedings of the 2nd Annual EuSpRIG Symposium*, Amsterdam, Netherlands, July 2001, www.gre.ac.uk/-~cd02/eusprig/2001/AIB_Spreadsheets.htm.

[15] R. Casimir. Real programmers don't use spreadsheets. *ACM SIGPLAN Notices*, 27(6):10-16, June 1992.

[16] Markus Clermont (2002) "A Scalable Approach to Spreadsheet Visualization", PhD-Dissertation (University of Klagenfurt).

[17] Drew Cullen, "Spreadsheet snafu costs firm $24m," *The Register*, June 19, 2003, www.theregister.co.uk/content/67/31298.html.

[18] J. S. Davis. Tools for spreadsheet auditing. *International Journal of Human-Computer Studies*, 45(4):429-442, 1996.

[19] S. Ditlea. Spreadsheets Can be Hazardous to Your Health. *Personal Computing*, 11(1):60-69, 1987.

[20] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Gencel—A Program Generator for Correct Spreadsheets. *Journal of Functional Programming*, 16(3):293-325, May 2006.

[21] M. Fisher, G. Rothermel, D. Brown, M. Cao, C. Cook, and B. Burnett. Integrating Automated Test Generation into the WYSIWYT Spreadsheet Testing Methodology. *ACM Trans. on Software Engineering and Methodology*, 2006. To appear.

[22] M. Fisher II and G. Rothermel. The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanism. In 1st Workshop on End-User Software Engineering, pages 47-51, 2005.

[23] Joe Francoeur (2002), "Algorithms using Java for Spreadsheet Dependent Cell Recomputation", Technical Paper at The MITRE Corporation, http://arxiv.org/abs/cs.DS/0301036, 16.30 p.m. 2004/4/8

[24] Kathy Godfrey, "Computing error at Fidelity's Magellan Fund," *The Risks Digest*, Volume 16, Issue 72, January 6, 1995, catless.ncl.ac.uk/Risks/16.72.html.

[25] D.G. Hendry and T.R.G. Green. Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model. *International Journal of Human-Computer Studies*, 40(6):1033-1065, 1994

[26] Takeo Igarashi, Jock Mackinlay, Bay-Wei Chang and Polle Zellweger (1998) „Fluid Visualization of spreadsheet structures", Proceedings of the 1998 IEEE Symposium on Visual Languages, pages 118-125

[27] Thomas Isakowitz, Shimon Shocken, and Henry C. Lucas. Toward a Logical/Physical Theory of Spreadsheet Modeling. *ACM Transactions on Information Systems*, 13(1):1-37, 1995.

[28] Roland Mittermeir, Markus Clermon, Christian Hanin (2002) „A spreadsheet Auditing Tool Evaluated in an Industrial Context", Proceedings of the 3rd Annual Symposium of the EuSpRIG, pages 35-46

[29] R. Mittermeir and M. Clermont. Finding High-Level Structures in Spreadsheet Programs. In 9th Working Conference on Reverse Engineering, pages 221-232, 2002.

[30] Nash, J.C., Smith, N., and Adler, A. (2003). Audit and change analysis of spreadsheets. Proceedings of the 2003 EUSPRIG Conference (European Spreadsheet Interest Group), David Chadwick and DavidWard, editors81–88. Also School of Management, University of Ottawa, Working Paper 03-23.

[31] David Nixon and Mike O'Hara, "Spreadsheet Auditing Software," *Proceedings of the 2nd Annual EuSpRIG Symposium*, Amsterdam, Netherlands, July 2001, www.gre.ac.uk/%7Ecd02/EUSPRIG/2001/Nixon_-2001.htm.

[32] Raymond R. Panko. What we know about spreadsheet errors. *Journal of End User Computing: Special issue on Scaling Up End User Development*, 10(2):15-21, Spring 1998.

[33] Panko, Raymond R., "Applying Code Inspection to Spreadsheet Testing," *Journal of Management Information Systems*, (16:2), Fall 1999, 159-176.

[34] Simon Peyton-Jones, Alan Blackwell and Margaret Burnett, "A User-Centred Approach to Functions in Excel", http://research.microsoft.com/~simonpj/Papers/excel, 14.30 p.m. 2004/4/6

[35] K. Rothermel, C. Cook, M. Burnett, J. Schonfeld, T. Green, and G. Rothermel. WYSIWYT testing in the spreadsheet paradigm: An empirical evaluation. In Proceedings of the 22nd International Conference onSoftware Engineering, Limerick, Ireland, June 2000.

[36] Kamalasen Rajalingham, David R. Chadwick and Brian Knight, "Classification of Spreadsheet Errors," *Proceedings of the 2nd Annual EuSpRIG Symposium*, Amsterdam, Netherlands, July 2001, www.mcs.vuw.ac.-nz/~db/references/rajalingham-00b.pdf

[37] Kamalasen Rajalingham, David Chadwick, Brian Knight, and Dilwyn Edwards. Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development. In *Proceedings of the 33rd Hawaii International Conference on System Sciences 2000*, volume 33. IEEE, 2000.

[38] J. Reichwein, G. Rothermel, and M. Burnett. Slicing spreadsheets: An integrated methodology for spreadsheet testing and debugging. In *Proceedings of the 2nd Conference on domain-specific languages*, volume 2, pages 25-38. ACM, 2000.

[39] Ronen, B., Palley, M. A., & Lucas, H. (1989). Spreadsheet Analysis and Design. *Communications of the ACM, 32(1)*, 84-92.

[40] G. Rothermel, M. M. Burnett, L. Li, C. DuPuis, and A. Sheretov. A Methodology for Testing Spreadsheets. *ACM Transactions on Software Engineering and Methodology*, pages 110-147, 2001.

[41] J. Ruthruff, E. Creswick, M. M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main. End-User Software Visualizations for Fault Localization. In ACM Symp. on Software Visualization, pages 123-132, 2003.

[42] G. Rothermel, L. Li, C. DuPuis, and M. Burnett. What you see is what you test: A methodology for testing form-based visual programs. In *ICSE 1998 Proceedings*, volume 20, pages 198—207. IEEE, April 1998.

[43] Jorma Sajaniemi, (1998), "Modelling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization", Technical Report A-1998-5, University of Joensuu

[44] T. S. H. Teo and M. Tan. Quantitative and Qualitative Errors in Spreadsheet Development. Proceedings of the Thirtieth Hawaii International Conference on System Sciences, 3:149-156, 1997.

[45] Markku Tukiainen (2001), "Comparing Two Spreadsheet Calculation Paradigms: An Empirical Study with Novice

Users", Interacting with Computers (IWC) , 13 (4), p. 427-446

[46] Nicolas P. Wilde, 1993. *A WYSIWYC (What You See Is What You Compute) Spreadsheet* . In Proceedings of the 1993 Symposium on Visual Languages, pages 72-76.

[47] Alan G. Yoder, David L. Cohn (1994), "Observations on Spreadsheet languages, Intension and Dataflow", (TR 94-22), http://www.cse.nd.edu/research/tech_reports/1994.html, 16.30 p.m. 2004/4/8

[48] Alan G. Yoder, David L. Cohn (2002), "Domain-specific and General-Purpose Aspects of Spreadsheet Languages", http://www-sal.cs.uiuc.edu/~kamin/dsl/papers/yoder.ps 16.30 p.m. 2004/4/8

[49] A. G. Yoder and D. L. Cohn. Real Spreadsheets for Real Programmers. In Int. Conf. on Computer Languages, pages 20-30, 1994.

[50] Karen Hodnigg. A Pragmatic Approach to Spreadsheet Training Based Upon the "Projection-Scren" Moder. Klagenfurt University, 2003