Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων
Διεθνές Πανεπιστήμιο της Ελλάδος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Βιβλιογραφική ανασκόπηση μεθόδων και μετρικών για την ανάλυση της μετάδοσης αλλαγών

Αρ. Μητρώου: 05/2918

Του φοιτητή

Αριστείδη Κουτομύλη

Επιβλέπων καθηγητής

Ιγνάτιος Δεληγιάννης

Θεσσαλονίκη 2021

ΠΕΡΙΛΗΨΗ

Κατά τη συντήρηση λογισμικού, το λογισμικό υπόκειται σε αλλαγές εξαιτίας της επιδιόρθωσης σφαλμάτων, της αλλαγής απαιτήσεων, την προσθήκη λειτουργιών κλπ. Με βάση την βιβλιογραφία, η διατήρηση χαμηλού κόστους συντήρησης είναι σημαντική, υποδηλώνοντας ότι το κόστος αυτής της φάσης είναι περίπου 50% - 75% του συνολικού κόστους ανάπτυξης λογισμικού. Το κόστος συντήρησης μπορεί να αυξηθεί περαιτέρω. Ενα λογισμικό χαρακτηρίζεται από: (α) μεταβλητότητα (change proneness), δηλαδή μεγάλη πιθανότητα αλλαγής μιας κλάσης ή μεθόδου, λόγω εσωτερικών λόγων (π.χ., επιδιόρθωση σφαλμάτων ή αλλαγή των απαιτήσεων), και (β) αστάθεια (instability), δηλαδή, πιθανότητα αλλαγής μιας κλάσης ή μεθόδου λόγω αλλαγών σε άλλα τμήματα του συστήματος. Η διαδικασία που ερευνά την μεταβλητότητα και την αστάθεια ονομάζεται ανάλυση μετάδοσης αλλαγών (change impact analysis). Ο σκοπός της πτυχιακής είναι να πραγματοποιηθεί μια ανασκόπηση της βιβλιογραφίας με σκοπό την εύρεση και ανάλυση όλων των μεθόδων και των μετρικών που έχουν προταθεί, χρησιμοποιηθεί, και αξιολογηθεί ώστε να αναλυθεί η μετάδοση των αλλαγών από ένα τμήμα του συστήματος σε ένα άλλο.

Λέξεις – κλειδιά: ανάλυση μετάδοσης αλλαγών, μετρικές, αστάθεια λογισμικού, μεταβλητότητα.

ΕΥΧΑΡΙΣΤΙΕΣ

Στο σημείο αυτό, θα ήθελα να ευχαριστήσω όλους όσους συνέβαλαν στην εκπόνηση της πτυχιακής μου εργασίας. Οφείλω να εκφράσω τις θερμές μου ευχαριστίες, προς τον επιβλέποντα της εργασίας, Καθηγητή Απόστολο Αμπατζογλου, για την καθοδήγησή του, και την πολύτιμη βοήθεια που προσέφερε σε κάθε στάδιο εκπόνησης της διατριβής μου. Επίσης, την κα Δρ. Αρβανίτου Ελβίρα-Μαρία, για την καθοδήγηση της σχετικά με το τεχνικό μέρος της εργασίας. Χωρίς τη συμπαράσταση και συνεχή βοήθειά τους, η ολοκλήρωση αυτής της εργασίας δεν θα ήταν δυνατή.

Τέλος, ευχαριστώ θερμά την οικογένεια, τους φίλους μου και τη σύντροφο μου, για την κατανόηση και συμπαράσταση που έδειξαν ολόκληρη την περίοδο εκπόνησης της εργασίας αυτής.

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο πινάκων

Ευρετήριο διαγραμμάτων

ΠΡΟΛΟΓΟΣ

Η ανάλυση μετάδοσης αλλαγών είναι η διαδικασία προσδιορισμού του συνόλου των τεχνουργημάτων λογισμικού που ενδέχεται να αλλάξουν όταν (ή αφού) ένα αντικείμενο έχει αλλάξει. Η ανάλυση αντικτύπου είναι ζωτικής σημασίας για λήψη τεκμηριωμένων αποφάσεων μεταξύ διαφορετικών εναλλακτικών λύσεων για την εφαρμογή μιας αλλαγής, για τον προγραμματισμό δραστηριοτήτων δοκιμών παλινδρόμησης, για την πρόβλεψη μελλοντικών εργασιών συντήρησης, και για τον επαναπροσδιορισμό / επανασχεδιασμό τμημάτων ενός συστήματος ώστε να είναι πιο ανθεκτικό στις αλλαγές.

Οι υπάρχουσες προσεγγίσεις για ανάλυση αντικτύπου βασίζονται σε στατική ανάλυση (έλεγχος, ροή δεδομένων και εξαρτήσεις), δυναμική ανάλυση και σε ανάλυση κειμένου των αιτημάτων αλλαγής και του κώδικα που συντηρείται . Πρόσφατα, η διαθεσιμότητα δεδομένων από συστήματα εκδόσεων και συστήματα εντοπισμού σφαλμάτων άνοιξε το δρόμο προς μια νέα προοπτική ανάλυσης λογισμικού, συγκεκριμένα την ιστορική ανάλυση, η οποία προσθέτει συμπληρωματικές πληροφορίες σε ό,τι διατίθεται με στατική και δυναμική ανάλυση, δηλαδή πληροφορίες σχετικά με αλλαγές που συμβαίνουν στα

τεχνουργήματα λογισμικού, για τα αντικείμενα που αλλάζουν μαζί, για το ποιος έκανε μια αλλαγή και γιατί έγινε η αλλαγή.

Οι περισσότερες έρευνες σχετικά με την ανάλυση μετάδοσης αλλαγών έχουν επικεντρωθεί στον πηγαίο κώδικα, μελετώντας τόσο τις επιδράσεις της αλλαγής μεταξύ αντικειμένων πηγαίου κώδικα όσο και σε άλλους τύπους τεχνουργημάτων (π.χ. δοκιμή εκτέλεσης μετά από μια αλλαγή). Ένας άλλος τομέας που έχει λάβει σημαντική προσοχή είναι η ανάλυση επιπτώσεων για τις απαιτήσεις, ειδικά κατά τη διαχείριση των απαιτήσεων και η ιχνηλασιμότητα. Η ανάλυση μετάδοσης αλλαγών για προδιαγραφές αρχιτεκτονικής, στοιχεία λογισμικού ή περιπτώσεις δοκιμών έχει επίσης συχνά διερευνηθεί.

Οι τεχνικές ανάλυσης μετάδοσης αλλαγών μπορούν να κατηγοριοποιηθούν σε γενικές γραμμές ως εξής:

1) στατική ανάλυση,
2) ανάλυση δυναμικών πληροφοριών
3) αποθετήρια λογισμικού,
4) μέτρηση σύζευξης
5) συνδυαστικές προσεγγίσεις

1)Η *ανάλυση στατικών εξαρτήσεων* συνήθως κάνει ανάλυση προσβασιμότητας σε γραφική αναπαράσταση ενός προγράμματος που προέρχεται από τον πηγαίο κώδικα (Breech, 2006), όπως γραφήματα κλήσεων, γραφήματα ροής ελέγχου ή γραφήματα εξάρτησης.

2) Η Ανάλυση Δυναμικών Πληροφοριών χρησιμοποιεί δεδομένα που δημιουργούνται από την εκτέλεση των προγραμμάτων και όχι μόνο τον πηγαίο κώδικα (Li B, 2013). Υπάρχουν διαφορετικοί τύποι δεδομένων εκτέλεσης. Για παράδειγμα, η Law και η Rothermel χρησιμοποιούν ίχνη εκτέλεσης των κλήσεων λειτουργίας (Law J, 2003).

3) Μέσω των αποθετηρίων λογισμικού αναλύεται όχι μόνο ο πηγαίος κώδικας και τα δεδομένα εκτέλεσης, αλλά και τα αρχεία καταγραφής των συστημάτων ελέγχου έκδοσης λογισμικού. Πολλά είδη αρχείων καταγραφής μπορούν να χρησιμοποιηθούν Για παράδειγμα, ο Zimmermann (Zimmermann T, 2005) εφάρμοσε την εξόρυξη δεδομένων σε αλλαγές που έγιναν ιστορικά για να δεί ποιες μέθοδοι συνήθως αλλάζουν μαζί.

4) Η μέτρηση σύζευξης στοχεύει στον υπολογισμό του βαθμού εξάρτησης μεταξύ δυο στοιχείων του προγράμματος, όπως πχ οι μέθοδοι. Αυτός ο βαθμός εξάρτησης υποδηλώνει την πιθανότητα να επηρεαστεί το ένα στοιχείο του προγράμματος από το άλλο. Η πλειονότητα των μετρήσεων σύζευξης βασίζεται σε αλληλεπιδράσεις μεταξύ δύο στοιχείων.

5) Οι συνδυασμένες προσεγγίσεις ενσωματώνουν πολλαπλά είδη αναλύσεων για μεγαλύτερη ακρίβεια. Για παράδειγμα, ο Breech (Breech, 2006) συνδύασε στατικές και δυναμικές προσεγγίσεις για μεγαλύτερη ακρίβεια.

1. ΕΙΣΑΓΩΓΗ

1.1 Ο ρόλος των μετρικών

“You cannot control what you cannot measure” [Tom De Marco]

Η κοινωνία εξαρτάται όλο και περισσότερο από πολύπλοκα συστήματα που βασίζονται σε υπολογιστές και λογισμικό. Διεισδύουν σε πολλές πτυχές της καθημερινής μας ζωής, όπως οι μεταφορές, η ενέργεια και η υγειονομική περίθαλψη και η δυσλειτουργία τους μπορεί να έχει σημαντικές και ίσως αρνητικές συνέπειες. Με την εξέλιξη των επιχειρηματικών αναγκών, οι απαιτήσεις των συστημάτων λογισμικού αλλάζουν συνεχώς και νέες απαιτήσεις εμφανίζονται συχνά. Η ενσωμάτωση των νέων ή τροποποιημένων απαιτήσεων με τις υπάρχουσες θέτει την ανάγκη προσαρμογής του κώδικα του συστήματος λογισμικού προκειμένου να ικανοποιηθεί το νέο σύνολο απαιτήσεων.

Κάθε προϊόν επηρεάζεται συνήθως από μεγάλο αριθμό παραγόντων, όπως, λειτουργικές απαιτήσεις, χαρακτηριστικά ποιότητας ή ζητήματα βελτίωσης της διαδικασίας λογισμικού. Δεδομένου ότι ο χρόνος, η χρηματοδότηση και οι πόροι είναι περιορισμένοι, σπάνια είναι δυνατό ή ακόμη και επιθυμητό να αντιμετωπιστούν πλήρως όλοι οι παράγοντες. Επομένως, το επίπεδο προσοχής σε έναν συγκεκριμένο παράγοντα θα πρέπει να αποφασιστεί ανάλογα με τη σημασία του (π.χ. επιχειρηματική αξία), το κόστος, τον κίνδυνο, την αστάθεια, τις εξαρτήσεις μεταξύ των παραγόντων και άλλα τέτοια κριτήρια. Αυτού του είδους οι αποφάσεις λαμβάνονται από ενδιαφερόμενους παράγοντες προϊόντων: χρήστες, πελάτες, διαχειριστές, χορηγοί, προγραμματιστές και άλλα άτομα που σχετίζονται με το προϊόν. Προκειμένου να ληφθούν αποφάσεις που εξαρτώνται από πολλούς παράγοντες γρήγορα, προτείνεται να δοθεί η κατάλληλη προτεραιότητα στον εκάστοτε παράγοντα.

Οι αλλαγές που προτείνονται, ο αντίκτυπος αυτών των αλλαγών σε άλλες απαιτήσεις, τα στοιχεία σχεδίασης και ο πηγαίος κώδικας πρέπει να αναλυθούν προκειμένου να προσδιοριστούν τμήματα του συστήματος λογισμικού που θα αλλάξουν. Ο προσδιορισμός του αντίκτυπου των αλλαγών σε άλλα αντικείμενα ανάπτυξης ονομάζεται ανάλυση μετάδοσης αλλαγών (change impact analysis).

Το 1994 οι Chidamber & Kemerer (Chidamber SR, 1994) πρότειναν την πρώτη ομάδα αντικειμενοστρεφών μετρικών. Αρκετές παραλλαγές προέκυψαν από τότε με τις πιο δημοφιλείς κατηγορίες να διαμορφώνονται ως εξής:

- Πολυπλοκότητας
- Κληρονομικότητας
- Μεγέθους
- Σύζευξης
- Συνοχής

Έχουν διεξαχθεί πολλές μελέτες για να συσχετιστούν οι μετρήσεις λογισμικού με εξωτερικά ποιοτικά χαρακτηριστικά, όπως η δυνατότητα συντήρησης και η δυνατότητα ελέγχου. Ομοίως, έχουν διεξαχθεί μελέτες σε μετρικές μεταβλητότητας και αστάθειας. Ορίζουμε τη **σταθερότητα** ως την ευκολία με την οποία ένα στοιχείο λογισμικού μπορεί να εξελιχθεί διατηρώντας παράλληλα το σχεδιασμό του όσο το δυνατόν περισσότερο. Η **μεταβλητότητα** μιας κλάσης είναι η δεύτερη εξαρτημένη μεταβλητή της παρούσας εργασίας. Ορίζεται ως η πιθανότητα να υπάρξει αλλαγή σε μια τάξη μετά την κυκλοφορία μιας έκδοσης του προϊόντος.

Λόγω της σημασίας της ανάλυσης μετάδοσης αλλαγών, έχει γίνει προσπάθεια στην ανάπτυξη μετρικών (μεταβλητών) και για τη μέτρηση της αστάθειας και της μεταβλητότητας του λογισμικού και για την εμπειρική αξιολόγηση της χρησιμότητάς τους για την ανάπτυξη λογισμικού στην πράξη. Συχνά μια μετρική εμπίπτει σε περισσότερες από μια κατηγορία καθώς δύναται να αξιολογήσει πολύπλευρα ένα λογισμικό, όπως θα φανεί και στη συνέχεια. Στην εργασία αυτή επιχειρείται να καταγραφούν όλες οι μέθοδοι και οι μετρικές που έχουν προταθεί, χρησιμοποιηθεί, και αξιολογηθεί ώστε να αναλυθεί η μετάδοση των αλλαγών από ένα τμήμα του συστήματος σε ένα άλλο και έχουν αναφερθεί στη σχετική βιβλιογραφία.

Η μέτρηση και η αξιολόγηση της σταθερότητας των διαδικασιών συντήρησης είναι σημαντική λόγω της αναγνωρισμένης σχέσης μεταξύ της ποιότητας της διαδικασίας και της ποιότητας του προϊόντος. Μια διαδικασία συντήρησης μπορεί γρήγορα να γίνει ασταθής επειδή η ίδια η εγκατάσταση λογισμικού αλλάζει το περιβάλλον. Οι αλλαγές που δημιουργούνται από τους χρήστες και το περιβάλλον και η συνακόλουθη ανάγκη προσαρμογής του λογισμικού στις αλλαγές είναι απρόβλεπτες. Τα προγράμματα πρέπει να είναι προσαρμόσιμα στις αλλαγές και η προκύπτουσα διαδικασία αλλαγής πρέπει να προγραμματίζεται και να ελέγχεται.

Τα δομικά χαρακτηριστικά των modules του λογισμικού, όπως αποτυπώνονται από διάφορες μετρικές, έχουν συσχετιστεί συνήθως με προβλήματα, όπως αλλαγή ή εμφάνιση ελαττωμάτων και η προσπάθεια συντήρησης. Ορισμένες εμπειρικές μελέτες υποστήριξαν αυτές τις απόψεις, οπως ο El Emam, που σημείωσε ότι υπήρχε συνεχής σχέση μεταξύ μεγέθους και ελαττωμάτων (K. El Emam, 2002). Οι αναλύσεις των Briand et al. (L.C. Briand, 2000) εδειξαν ότι οι μετρικές σύζευξης και

κληρονομικότητας σχετίζονται στενά με την πιθανότητα εμφάνισης ελαττωμάτων σε μια κλαση. Οι Cartwright και Shepperd διαπίστωσαν ότι οι κλασεις που συμμετέχουν στις ιεραρχίες κληρονομικότητας είχαν μεγαλύτερη πιθανότητα να περιέχουν σφαλματα.

Στη δημοσίευση (Li W, 1993), οι Li et al. διαπίστωσαν ότι υπήρχε ισχυρή συσχέτιση μεταξύ των μετρικών και της μεταβλητότητας. Συγκεκριμένα, απέδειξαν ότι το μοντέλο πρόβλεψης τους για τη μεταβλητότητα από τις μετρικές είχε καλή απόδοση. Στο (M, 1998), ο Lindvall έδειξε ότι οι μεγάλες κλάσεις ήταν πιο επιρρεπείς σε αλλαγές. Στο (Arisholm, Briand, & Foyen, 2004) οι Arisholm et al. ανέφεραν ότι οι μετρικές δυναμικής σύζευξης ήταν σημαντικοί δείκτες μεταβλητότητας. Στο (Koru & Tian, 2005) οι Koru και Tian διερεύνησαν εάν οι κλάσεις με τις υψηλότερες τιμές είχαν και τη μεγαλύτερη τάση μεταβλητότητας. Αρχικά, ταξινόμησαν τις μετρικές δομής σε μετρικές μεγέθους, ζεύξης, συνοχής και κληρονομικότητας. Στη συνέχεια, ανέλυσαν εάν οι κλάσεις με πολλές αλλαγές ήταν κλάσεις μεγάλου μεγέθους, υψηλής σύζευξης, χαμηλής συνοχής ή υψηλής κληρονομικότητας. Εν τέλει, διαπίστωσαν ότι οι κλάσεις με πολλές αλλαγές δεν είχαν τις υψηλότερες τιμές μέτρησης. Συνολικά, προηγούμενες μελέτες καταλήγουν στο συμπέρασμα ότι οι μετρικές είναι σε θέση να προβλέψουν την μεταβλητότητα και την αστάθεια.

Οι Basili et al. (Basili VR, 1996) ήταν από τους πρώτους που χρησιμοποίησαν τις μετρικές που πρότειναν οι C&K (Chidamber και Kemerer 1994) για την πρόβλεψη ελαττωματικών κλάσεων. Για να επικυρώσουν το έργο τους, οι συγγραφείς συνέλεξαν δεδομένα σχετικά με την ανάπτυξη οκτώ παρόμοιων συστημάτων διαχείρισης πληροφοριών μικρού μεγέθους (180 κλάσεις συνολικά). Και τα οκτώ συστήματα αναπτύχθηκαν χρησιμοποιώντας τη γλώσσα προγραμματισμού C ++. Τα αποτελέσματα έδειξαν ότι πέντε από τις έξι μετρικές φαίνεται να είναι χρήσιμες για την πρόβλεψη ελαττωματικών κλάσεων.

Οι Moser και Pedrycz (Moser, 2008) πρότειναν 17 μετρικές σε επίπεδο αρχείου, που κυμαίνονται από τον αριθμό των refactorings, των συγγραφέων, των διορθώσεων των bugs. Χρησιμοποιώντας τρεις διαφορετικούς δυαδικούς ταξινομητές, διαπίστωσαν ότι οι μετρήσεις τους ήταν σημαντικά καλύτερες από αυτές των Zimmermann et al. στο σύνολο δεδομένων του Eclipse.

1.2 Αναζήτηση άρθρων

Στην παρούσα εργασία παρουσιάζεται μια συστηματική βιβλιογραφική ανασκόπηση σχετικά με τα παραπάνω.

Η αναζήτηση άρθρων πραγματοποιήθηκε σε 5 βιβλιοθήκες:

- IEEE Digital Library

- ACM Digital Library
- Springer
- Wiley
- ScienceDirect

Η κάθε βιβλιοθήκη φιλοξενεί συγκεκριμένα περιοδικά και συνέδρια:

- IEEE Digital Library:
  - Transactions on Software Engineering (TSE)
  - International Conference on Software Engineering (ICSE)
  - International Symposium on Empirical Software Engineering and Measurement (ESEM)
  - International Conference on Automated Software Engineering (ASE)
  - International Conference on Software Processes (ICSP)
  - IEEE Software (SW)
- ACM Digital Library:
  - Transactions on Software Engineering and Methodology (TOSEM)
  - International Symposium on the Foundations of Software Engineering (FSE)
- Springer:
  - Empirical Software Engineering (ESE)
- Wiley:
  - Software: Practice and Experience (SPE)
- ScienceDirect:
  - Information and Software Technology (IST)
  - Journal of Systems and Software (JSS)

Το search string που χρησιμοποιήθηκε στην αναζήτηση, είναι το εξής:

"Change impact analysis" OR (("change proneness" OR "changeability" OR "instability") AND ("metric" OR "method" OR "measurement"))

Η αναζήτηση σε αρχική φάση αφορούσε μόνο τον τίτλο και την περίληψη του εκάστοτε άρθρου. Ορισμένα από τα παραπάνω συνέδρια δε φιλοξενούνται κάθε χρόνο από την ίδια βιβλιοθήκη. Υπάρχει πιθανότητα δηλαδή τον ένα χρόνο να φιλοξενούνται στην IEEE και τον άλλο χρόνο στην ACM. Συνεπώς η αναζήτηση για ορισμένα συνέδρια έγινε σε περισσότερες από μια βιβλιοθήκη.

Μετα την ολοκλήρωση της αναζήτησης προέκυψαν τα παρακάτω αποτελέσματα ανά συνέδριο:

Συνέδριο/ Περιοδικό	**Πλήθος**
International Conference on Automated Software Engineering	2
Empirical Software Engineering	23
International Symposium on Empirical Software Engineering and Measurement	6
International Symposium on the Foundations of Software Engineering	1
International Conference on Software Engineering	6
Information and Software Technology	23
Journal of Systems and Software	17
Software: Practice and Experience	2
IEEE Software	1
Transactions on Software Engineering	7

Από τα 88 παραπάνω άρθρα, και έπειτα από προσεκτική αξιολόγηση του καθενός ξεχωριστά, διατηρήθηκαν στο τελικό dataset τα 74. Δεν διατηρήθηκαν εντός όσα δε χρησιμοποίησαν μετρικές για να υπολογίσουν την αστάθεια ή τη μεταβλητότητα. Δε διατηρήθηκαν ούτε οι βιβλιογραφικές εργασίες.

Για τα 88 αρχικά άρθρα συλλέχθηκαν οι ακόλουθες δώδεκα διαφορετικές μεταβλητές:

- Τίτλος της δημοσίευσης
- Συγγραφείς της δημοσίευσης
- Έτος έκδοσης της δημοσίευσης
- Δημοσίευση σε συνέδριο ή περιοδικό
- Επωνυμία συνέδριου/ περιοδικού
- Κίνητρο της δημοσίευσης (καταγραφή οφέλους ή αξιολόγηση του φαινομένου)
- Φάση υλοποίησης (απαιτήσεις, σχεδιασμός, υλοποίηση, έλεγχος ή συνδυασμός αυτών)
- Τεχνούργημα που χρησιμοποιείται (διάγραμμα κλάσης, πηγαίος κώδικας κλπ.)
- Ποιοτικό χαρακτηριστικό που αξιολογείται
- Λίστα σχετικών μετρικών
- Ύπαρξη εργαλείου για μέτρηση μετρικών (υπολογισμός μετρικής από εργαλείο που ανέπτυξαν ή χρησιμοποίησαν οι ερευνητές)
- Εμπειρική μέθοδος ερευνάς (μελέτη περίπτωσης, έρευνα, πείραμα κλπ.)

Η υπόλοιπη εργασία οργανώνεται ως εξής: Στα κεφάλαια 2-5 παρατίθενται σύντομες παρουσιάσεις όσων δημοσιεύσεων μελετήθηκαν. Αναφέρονται

επιγραμματικά οι παραπάνω μεταβλητές που καταγράφηκαν για καθεμιά από αυτά. Στο κεφάλαιο 2 αναφέρονται όσες σχετίζονται με την αστάθεια λογισμικού, στο κεφάλαιο 3 με τη μεταβλητότητα, στο κεφάλαιο 4 με την ποσότητα αλλαγής και στο κεφάλαιο 5 με τη δυνατότητα αλλαγής. Το κεφάλαιο 6 καταλήγει σε κάποια συμπεράσματα, ενώ στο κεφάλαιο 7 αναλύονται ορισμένοι από τους κίνδυνους εγκυρότητας.

2. Αστάθεια λογισμικού

Περίπου τα μισά από το σύνολο των άρθρων που μελετήθηκαν στη διάρκεια της τρέχουσας εργασίας αφορούσαν την **αστάθεια** λογισμικού. Από τα συνολικά 87 άρθρα που περιλαμβάνονταν στην αρχική συλλογή δεδομένων τα 40 είχαν συσχέτιση με την αστάθεια λογισμικού.

Ακολουθούν οι συνοπτικές περίγραφες των εν λόγω άρθρων.

2.1 Συλλογή άρθρων

Η γραμμή παραγωγής λογισμικού στοχεύει στη βελτίωση μιας μεγάλης οικογένειας συστημάτων λογισμικού (Software product line). Τα κόστη της εξέλιξης αυτών των συστημάτων πιθανώς να ανεβάσουν και τα κόστη αυτής της γραμμής συνολικά. Το 2011 οι (Tizzei, Dias, Rubira, Garcia, & Lee, 2011) αξιολογήσαν τις αρνητικές και τις θετικές επιδράσεις δυο διαφορετικών προσεγγίσεων στον τομέα της αρχιτεκτονικής της γραμμής παραγωγής λογισμικού. Η πρώτη προσέγγιση είναι η προσανατολισμένη στη σύνταξη και τη συντακτική ανάλυση και η δεύτερη μη προσανατολισμένη στη σύνταξη. Η ποιοτική και ποσοτική ανάλυση τους έγινε στην περίπτωση χρήσης του COSMOS (component model). Υπολογίζοντας τον αριθμό των modules που αλλάχθηκαν στη διάρκεια της ζωής του τελευταίου κατέληξαν στο συμπέρασμα ότι ένας συνδυασμός των δυο προσεγγίσεων αποτελεί την καλύτερη λύση. Η βασική συνεισφορά αυτού του άρθρου είναι η καινοτόμος ανάλυση των πλεονεκτημάτων και των μειονεκτημάτων της ενσωμάτωσης των components στο σχεδιασμό μια γραμμής παραγωγής λογισμικού. Ο συνδυασμός της χρήσης components και aspects οδήγησε, στην περίπτωση χρήσης που εξετάστηκε, σε υψηλή συνοχή του πηγαίου κώδικα.

Την προηγουμένη δεκαετία τα ψηφιακά αποθετήρια λογισμικού (Mining Software repositories) έδωσαν μια νέα κατεύθυνση στην έρευνα θεωρητικά αλλά και πρακτικά. Σε μια προσπάθεια βελτίωσης της αποτελεσματικότητας του λογισμικού οι (Li, Sun, & Keung, 2013) εξετάζουν το θέμα των ψηφιακών αποθετηρίων

λογισμικού. Το θέμα περιλαμβάνει από αρχεία επικοινωνιών μέχρι αποθετήρια bugs. Δεν είχε ερευνηθεί ενδελεχώς εκείνη τη χρονική στιγμή το ποιες συγκεκριμένες πληροφορίες ήταν χρήσιμες από ένα τέτοιο αποθετήριο ώστε να υποστηριχθούν κατάλληλα οι διαδικασίες συντήρησης. Με στόχο να βελτιώσουν τις διαδικασίες συντήρησης του λογισμικού προτείνουν τη χρήση του εργαλείου MSR4SM το οποίο αξιολογείται πάνω σε δυο βασικές διαδικασίες συντήρησης μεταξύ των οποίων και η ανάλυση μετάδοσης αλλαγών. Το εργαλείο έχει τη δυνατότητα να ξεχωρίσει μια χρήσιμη από μια αχρείαστη πληροφορία. Διευκολύνει τη συντήρηση του λογισμικού εξάγοντας χρήσιμες πληροφορίες από καθένα από τα παραπάνω ψηφιακά αποθετήρια. Ερευνήθηκαν οι περιπτώσεις χρήσης ορισμένων εφαρμογών ανοιχτού λογισμικού (jEdit, ArgoUML andKOffice) και αποδείχτηκε ότι μπορεί να βελτιώσει την αποτελεσματικότητα των παραδοσιακών ψηφιακών αποθετηρίων.

Το μακρινό 2000 οι (Li, Etzkorn, Davis, & Talburt, 2000) προσπάθησαν να μετρήσουν την αστάθεια του αντικειμενοστρεφούς λογισμικού στη φάση της σχεδίασης του, δηλαδή πριν δημιουργηθούν. Ο βασικός σκοπός τους ήταν να μπορέσουν να προβλέψουν την αστάθεια λογισμικού ή την τάση για αλλαγή μέσα από υπάρχουσες μετρικές. Για αυτό το σκοπό χρησιμοποίησαν 3 εξελικτικές μετρικές: I. System Design Instability, ii. Class Implementation Instability, iii. System Implementation Instability. Οι μετρικές, σύμφωνα με αυτό το paper, μπορούν να παρέχουν πληροφορίες που θα προσαρμόσουν κατάλληλα το χρονοδιάγραμμα του project. Η πρώτη και η τρίτη παρέχουν ενδείξεις για την εξέλιξη του project. Χρησιμοποιούνται δεκάδες μετρικές άλλων χαρακτηριστικών ποιότητας, όπως πχ το coupling οι οποίες έχουν προταθεί από τους Chidamber και Kemerer, για να μετρηθούν εν τελεί και αλλαγές στη φάση της υλοποίησης του λογισμικού. Ολοκλήρωσαν μια μελέτη της αστάθειας του σχεδιασμού οπού εξετάστηκε πως η ενσωμάτωση μιας κλάσης μπορεί να επηρεάσει το γενικότερο σχεδιασμό. Ωστόσο, καταλήγουν στο ότι δεν μπορούν όλες οι μετρικές να βοηθήσουν στην ανάλυση μετάδοσης αλλαγών. Έβγαλαν το συμπέρασμα ότι κάποια χαρακτηριστικά του σχεδιασμού είναι άμεσα εξαρτώμενα με την υλοποίηση ενώ αλλά όχι.

Πολλές τεχνικές έχουν προταθεί στη βιβλιογραφία για να προβλεφθούν οι αλλαγές σε επίπεδο πηγαίου κώδικα. Μεταξύ άλλων έχει καταγραφεί ότι οι αλλαγές που έχουν ήδη συμβεί τείνουν να ξανασυμβαίνουν, δηλαδή η ιστορία επαναλαμβάνεται. Υπάρχουν λοιπόν τεχνικές που επιβάλουν την προσεκτική καταγραφή ιστορικότητας και εξέλιξης των εκδόσεων του λογισμικού. Σε μια πιο πρόσφατη ερευνά του 2015 οι (Abdeen, Bali,

Sahraoui, & Dufour, 2015) εξέτασαν την αστάθεια λογισμικού σε επίπεδο *ιστορικότητας* των κλάσεων. Προηγούμενες έρευνες είχαν δείξει ότι συνδυάζοντας σύγχρονα με προγενέστερα δεδομένα μπορεί να βελτιωθεί η ακρίβεια της ανάλυσης μετάδοσης δεδομένων. Προτείναν μια υβριδική προσέγγιση που προβλέπει τις επιπτώσεις των αλλαγών από υπάρχουσες καταγεγραμμένες ιστορικότητες. Εξέτασαν μεθοδικά το συσχετισμό των γραφημάτων των αλλαγών και τους συσχετισμούς ανάμεσα στους διαφορετικούς παράγοντες που τις επηρεάζουν. Εφόσον οι predictors εκπαιδευτούν σωστά μπορούν να προβλέψουν την αστάθεια σε οποιαδήποτε μετέπειτα έκδοση του λογισμικού. Εξέτασαν επιτυχώς τέσσερα δημοφιλή συστήματα java. Αποδείχθηκε ότι για την ακρίβεια της πρόβλεψης της μετάδοσης των αλλαγών αρκεί η πληροφορία των δομικών συσχετίσεων σε σχέση με την τρέχουσα κατάσταση του εκάστοτε λογισμικού που αναλύεται. Η προσέγγιση τους παρήγαγε ακριβείς προβλέψεις όχι μόνο για τις παρελθούσες αλλά και για τις νεότερες κλάσεις .

Ακολουθώντας την εξέλιξη των αναγκών της αγοράς οι απαιτήσεις αλλάζουν συχνά αλλά δημιουργούνται και νέες συνεχώς. Όταν δημιουργείται μια νέα απαίτηση αυτομάτως ένας μηχανικός καλείται να αναλύσει αυτή την απαίτηση ώστε να υλοποιηθεί η αλλαγή. Τις επιδράσεις αυτών των αλλαγών θέλησαν να αναλύσουν οι (Goknil, Kurtev, van den Berg, & Spijkerman, 2014) το 2014 που εστίασαν στη φάση των απαιτήσεων. Σε συνέχεια της προηγουμένης σχετικής τους καταγραφής, θέλησαν να τυποποιήσουν τις συσχετίσεις των απαιτήσεων και των αλλαγών. Παρείχαν μια ταξινόμηση των αλλαγών των απαιτήσεων. Οι περισσότερες προσεγγίσεις και εργαλεία δεν εστιάζουν στον προσδιορισμό και την επιβεβαίωση των συσχετίσεων των απαιτήσεων. Ο προσδιορισμός που παρουσιάζεται παρέχει μια πιο ακριβή ανάλυση μετάδοσης αλλαγών. Το εργαλείο που χρησιμοποίησαν ήταν μια επέκταση του δικού τους TEO for Requirements Inferencing and Consistency Checking (TRIC). Η προσέγγιση τους βοηθά στην εξάλειψη ορισμένων αρνητικών επιδράσεων στη διάδοση των αλλαγών ενώ παράλληλα ενεργοποιούνται εναλλακτικές που μειώνουν τις αλλαγές των απαιτήσεων που απαιτούνται. Το βασικό της πλεονέκτημα είναι ότι δίνονται εναλλακτικές επιλογές στον μηχανικό των απαιτήσεων ο οποίος με τη σειρά του μπορεί να αναλύσει την πορεία από το στάδιο των απαιτήσεων σε αυτό της αρχιτεκτονικής.

Την περασμένη δεκαετία η βιομηχανία λογισμικού αναγνώρισε τη στρατηγική σημασία των γραμμών παραγωγής. Πλέον η προσαρμογή στις νέες αλλαγές είναι κομμάτι αυτής της βιομηχανίας και δεν είναι εφικτό να τις

αποφύγει κάνεις. Συνεπώς την τελευταία δεκαετία έχουν αναπτυχθεί και πολλαπλασιαστεί μέθοδοι που ενσωματώνονται στον όρο Ευέλικτη Ανάπτυξη Λογισμικού. Μέσα από την περίπτωση χρήσης των έξυπνων δικτυών ξεκίνησαν την ερευνά τους οι (Díaz, Pérez, & Garbajosa, 2014) . Καθώς η γραμμή παραγωγής λογισμικού απαιτεί την εκ των προτέρων σχεδιασμό της αρχιτεκτονικής έρχεται σε αντίθεση με την ανάγκη ευελιξίας και ευκινησίας που υπάρχει στο σύγχρονο περιβάλλον. Παρόλα αυτά είναι απαραίτητη. Έτσι στο άρθρο τους παρουσιάζουν μια Ευέλικτη Αρχιτεκτονική Γραμμή Παραγωγής (Agile Product-Line Architecting) βασισμένη σε ένα αλγόριθμο που κατευθύνει τη λήψη αποφάσεων στις διαδικασίες της φάσης της αρχιτεκτονικής ενός λογισμικού. Η ποσοτική ανάλυση έγινε χρησιμοποιώντας κυρίως στατιστικά στοιχεία ενώ η ποιοτική ανάλυση μέσω συνεντεύξεων. Ο μηχανισμός που παρέχεται από την Ευέλικτη Αρχιτεκτονική υποστηρίζει: α) τον προσδιορισμό των παρεχόμενων αρχιτεκτονικών χαρακτηριστικών β) την τεκμηρίωση των αποφάσεων σχεδιασμού γ) την ιχνηλασιμότητα των χαρακτηριστικών και δ) την απόκτηση γνώσης για την επίδραση των αλλαγών ώστε να προκύπτουν σωστές αποφάσεις. Τα αποτελέσματα προέκυψαν μέσα από τρία διαφορετικά project της εταιρείας που και συνεργάστηκε και απέδειξαν ότι – στον τομέα των έξυπνων δικτυών τουλάχιστον- η εφαρμογή της Ευέλικτης Αρχιτεκτονικής Γραμμής Παραγωγής μπορεί να προσφέρει αυτοματοποιήσεις στην ανάλυση μετάδοσης αλλαγών. Τέλος, η περίπτωση χρήσης που μελετήθηκε αποκάλυψε ότι η αρχιτεκτονική που βασίζεται στις παραπάνω γνώσεις οδηγεί σε λιγότερο επίφοβες αλλαγές, κάτι που είναι σημαντικό στον κόσμο της ευελιξίας.

Τις τελευταίες δεκαετίες υπάρχει μια αξιοσημείωτη αύξηση των πολύ μεγάλων αντικειμενοστρεφών συστημάτων λογισμικού που αποτελούνται από χιλιάδες κλάσεις . Είναι σύνηθες πλέον οι συσχετιζόμενες κλάσεις να ομαδοποιούνται σε πακέτα. Αυτά, αν είναι καλά σχεδιασμένα, μπορούν να μειώσουν την πολυπλοκότητα του συστήματος και να το κάνουν ευκολότερα κατανοητό και επεκτάσιμο. Συνεπώς, για να μειωθεί το κόστος της διασφάλισης ποιότητας είναι απαραίτητο να ερευνηθεί η ποιότητα της οργάνωσης των πακέτων. Το 2014 οι Zhao, Y., Yang, Y., Lu, H., Zhou, Y., Song, Q., & Xu, B. (Zhao, et al., 2015) εκτίμησαν την επίδραση της αστάθειας του λογισμικού πάνω στην επήρεια σε σφάλματα. Για το σκοπό αυτό προτείναν ένα νέο πακέτο μετρικών που αφορά αντικειμενοστρεφή συστήματα. Ενώ οι μέχρι τότε μετρικές επικεντρώνονταν κυρίως στο μέγεθος, την επεκτασιμότητα και τις προοπτικές αστάθειας, το νέο πακέτο βασίζεται στην επικοινωνία μεταξύ των μονάδων, την εύθραυστη σχεδίαση

και τη ρύπανση των plugins. Βασισμένοι σε έξι διαφορετικά συστήματα ανοιχτού λογισμικού (java) έδειξαν ότι το νέο πακέτο παρέχει μια διαφορετική προοπτική σε σχέση με τις παραδοσιακές μετρικές ενώ με τον συνδυασμό νέων και παραδοσιακών μετρικών βελτιώνονται οι προβλέψεις της επήρειας σε σφάλματα. Τα αποτελέσματα έδειξαν ότι: α)οι μετρικές του Sarkar δίνουν νέες, συμπληρωματικές προοπτικές για συγκρινόμενες με τις παραδοσιακές β) οι μετρικές του Sarkar δε σχετίζονται με την επήρεια σε λάθη και γ) μπορούν να βελτιώσουν την πρόβλεψη επήρειας σε λάθη αν συνδυαστούν με παραδοσιακές μετρικές. Στο πρώτο από τα δυο σενάρια που έτρεξαν η βελτίωση ήταν της τάξης του 33% στην πρόβλεψη των προβληματικών πακέτων ενώ στο δεύτερο σενάριο η βελτίωση έφτασε πάνω από 42% αποδεικνύοντας για μια ακόμη φορά ποσό σημαντική είναι μια τέτοια πρόβλεψη, ειδικά όταν οι πόροι για έλεγχο και δοκιμές είναι περιορισμένοι.

Από το συνολικό κύκλο ζωής ενός λογισμικού το 60-90% διοχετεύεται στη συντήρηση του, κάτι που την καθιστά την πιο χρονοβόρα διαδικασία. Καθώς λοιπόν η συντήρηση του λογισμικού συνεχώς αυξάνει το χρόνο που απαιτείται, η πρόβλεψη αυτού του χρόνου μπορεί να έχει κρίσιμο ρολό στη διάρκεια της εξέλιξης του. Το ISO/IEC 9126 διαχωρίζει τη συντήρηση αυτή σε δυο τομείς: στην προσπάθεια αλλαγής και στην αστάθεια του λογισμικού. Ενώ η προσπάθεια αλλαγής υποδεικνύει την ικανότητα του λογισμικού να ανταπεξέρχεται στις αλλαγές, η αστάθεια μπορεί να καταγραφεί αφού μια αλλαγή έχει ολοκληρωθεί. Οι Conejero, J. M., Figueiredo, E., Garcia, A., Hernández, J., & Jurado, E (Conejero, Figueiredo, Garcia, Hernández, & Jurado, 2012) υπολόγισαν τη συσχέτιση μεταξύ της προσπάθειας και της αστάθειας βασισμένοι σε μετρικές που επιτρέπουν την ποσοτικοποίηση της αντιγραφής του πηγαίου κώδικα και συγκεκριμένων εξαρτήσεων. Η εμπειρική μελέτη που διεξήχθη επιβεβαίωσε ότι μια αλλαγή σε ένα από τα βασικά κομμάτια του λογισμικού είναι πιο δύσκολο να πραγματοποιηθεί και όσο εξελίσσεται το λογισμικό τόσο πιο ασταθές γίνεται. Το κύριο ερώτημα που απασχόλησε τους ερευνητές είναι αν η συντήρηση επηρεάζεται αρνητικά από τις ιδιότητες των κυρίων τμημάτων του λογισμικού στις πρώιμες φάσεις όπως αυτή των απαιτήσεων. Για να απαντηθεί το ερώτημα παρατήρησαν διαφορά ποιοτικά χαρακτηριστικά σε τρεις διαφορετικές γραμμές παραγωγής: MobileMedia, HealthWatcher και SmartHome systems.

Η ανάλυση μετάδοσης αλλαγών (Software Change Impact Analysis (CIA)) είναι μια απαραίτητη τεχνική στη μηχανική λογισμικού για να μπορέσεις

κάνεις να προσδιορίσει τις δυνητικές επιρροές μιας αλλαγής ή για να αποφασίσει σε ποιες οντότητες να εφαρμοστεί μια αλλαγή. Τα αποτελέσματα της μερικές φορές είναι αμφιλεγόμενα. Οι κύριοι λόγοι είναι ότι οι συντηρητές του λογισμικού δε γνωρίζουν από ποια οντότητα του συνοδού να ξεκινήσουν την ανάλυση τους αλλά και το ότι οι περισσότερες τεχνικές υπολογίζουν την επιρροή μιας αλλαγής ως άθροισμα επιρροών ετεροκλήτων στοιχείων. Συχνά όμως έχει σημασία και η συσχέτιση αυτών των οντοτήτων στο τελικό άθροισμα. Την ανάγκη να διευθετηθεί αυτό το ζήτημα θέλησαν να καλύψουν οι Li, B., Sun, X., & Keung, J. (Li, Sun, & Keung, 2013) σχεδιάζοντας μια καινοτόμο προσέγγιση της ανάλυσης μετάδοσης αλλαγών. Η προσέγγιση αυτή, που ονομάζεται FCA–CIA (formal concept analysis (FCA)) βασίζεται σε στατική ανάλυση του πηγαίου κώδικα για να πράξει μια ενδιάμεση αναπαράσταση του προγράμματος. Η αναπαράσταση αυτή ονομάζεται Lattice of Class and Method Dependence (Locums). Η LoCMD οργανώνει τις μεθόδους σε ιεραρχική σειρά. Βασισμένη σε αυτή την οργάνωση η FCA–CIA υπολογίζει μια ταξινομημένη λίστα των επιρροών θεωρώντας τις αλλαγές ως ένα σύνολο. Η ταξινόμηση γίνεται σύμφωνα με μια νέα μετρική που προτείνεται: τον παράγοντα επιρροής (impact factor). Η εφαρμογή της παραπάνω τεχνικής έγινε με χρήση Java στο περιβάλλον του Eclipse.

Μια ιδιαίτερα συνήθης πρακτική στον προγραμματισμό είναι η χρήση βιβλιοθηκών. Ενώ τα πλεονεκτήματα της χρήσης αυτής είναι προφανή, είναι φανερό ότι εμπεριέχει το ρίσκο της κρησαρισμάτων λόγω των αλλαγών των APIs που εξελίσσονται ανεξάρτητα. Διαχρονικά η συμβατότητα μεταξύ του προγράμματος και των βιβλιοθηκών που χρησιμοποιεί ελέγχεται στο compile ολοκλήρου του συστήματος. Παρόλα αυτά η μεμονωμένη αναβάθμιση δείχνει να ξεπερνιέται. Έτσι, το 2014, οι Jezek, K., Dietrich, J., & Brada, P. (Jezek, Dietrich, & Brada, 2015) μελέτησαν το παραπάνω ζήτημα στον πραγματικό κόσμο. Ασχολήθηκαν με δυο όψεις αυτού: τη συμβατότητα των αλλαγών των APIs όσο εξελίσσονται οι βιβλιοθήκες και την επίδραση που αυτές έχουν στα προγράμματα που χρησιμοποιούν αυτές τις βιβλιοθήκες. Ένα σύνολο 109 java εφαρμογών ανοιχτού λογισμικού και 564 διαφορετικών εκδόσεων των εφαρμογών χρησιμοποιήθηκαν. Ερευνήθηκαν δυο είδη εξαρτήσεων βιβλιοθηκών: α) ρητές εξαρτήσεις με τις ενσωματωμένες βιβλιοθήκες και β) εξαρτήσεις που δημιουργούνται τη στιγμή του build μιας εφαρμογής. Το εργαλείο που χρησιμοποιήθηκε για την ανάλυση των APIs είναι το JaCC, το οποίο βασίζεται στη δημοφιλή βιβλιοθήκη ASM, με σκοπό να ερευνηθεί η συμβατότητα μεταξύ βιβλιοθηκών και προγραμμάτων. Κατέληξαν, στο ότι τα περισσότερα

προγράμματα που ερευνήθηκαν, ότι τα APIs είναι ασταθή καθώς οι ασύμβατες αλλαγές είναι συχνές. Παραδόξως, τα προβλήματα συμβατότητας ήταν περισσότερα στα project που χρησιμοποιούν αυτοματοποιημένη ανάλυση εξαρτήσεων. Συνεπώς η αστάθεια των APIs δημιουργεί όντως προβλήματα στα προγράμματα που τα χρησιμοποιούν. Επομένως χρειάζονται καλυτέρα εργαλεία κ μέθοδοι για να διασφαλίσουν την ανάπτυξη των βιβλιοθηκών.

Οι εξαρτήσεις μεταξύ ανεξάρτητων απαιτήσεων παίζουν σπουδαίο ρολό στις δραστηριότητες της ανάπτυξης λογισμικού όπως πχ σχεδιασμός και αρχιτεκτονική αλλά ανάλυση μετάδοσης αλλαγών. Παρόλο που πολλά είδη εξαρτήσεων των απαιτήσεων είχαν προταθεί στη βιβλιογραφία, υπήρχε μέχρι και το 2013 σχετικά μικρή εμπειρία στην εφαρμογή αυτών των εξαρτήσεων στην πραγματική ζωή. Τότε ήταν που οι Zhang, H., Li, J., Zhu, L., Jeffery, R., Liu, Y., Wang, Q., & Li, M. (Zhang, et al., 2014) αποφάσισαν να κατανοήσουν αν οι υπάρχοντες τύποι εξαρτήσεων είναι χρήσιμοι στην πράξη και να προτείνουν βελτιώσεις αλλά και να δημιουργήσουν τους ορισμούς αυτών. Διάμεσου της μελέτης της περίπτωσης της οργάνωσης Lending Industry XML Initiative (LIXI) αξιολογήθηκε η χρησιμότητα και η δυνατότητα εφαρμογής δυο γενικών μοντέλων εξαρτήσεων (P-model and D-model) που καλύπτουν 25 είδη εξαρτήσεων. Η μελέτη αφορά τον πραγματικό κόσμο της βιομηχανίας με συμμετοχή τριών συμμετεχόντων που προσφέρουν διαφορετικές προοπτικές. Η αρχική ανάλυση έδειξε ότι υπάρχουν αλληλεπικαλυπτόμενες και/ή ασαφείς τύποι εξαρτήσεων στα υπάρχοντα μοντέλα. Στην πράξη οι συμμετέχοντες εξέφραζαν διαφορετική άποψη για τη μετάδοση αλλαγών ανάλογα με το υπόβαθρο και τις εμπειρίες τους. Προκειμένου να βελτιωθεί αυτή η κατάσταση, προτάθηκε ένα νέο μοντέλο εξαρτήσεων το οποίο ταξινομεί τις εξαρτήσεις σε εσωτερικές και πρόσθετες και προτείνει εννιά τύπους εξαρτήσεων με τους ακριβείς ορισμούς τους. Οι τύποι εξαρτήσεων είναι σημαντικοί στη διάδοση των αλλαγών και την αστάθεια του λογισμικού όχι μόνο επειδή μπορούν να χρησιμοποιηθούν στα φυσικά εγχειρίδια απαιτήσεων αλλά και στα μοντέλα απαιτήσεων και στις γλώσσες που χρησιμοποιούνται σε αυτά.

Η ανάπτυξη και συντήρηση πολύπλοκων συστημάτων λογισμικού είναι απαιτητικά καθήκοντα. Η αυτονομία των συστατικών στοιχείων (modularity) παίζει σπουδαίο ρολό σε αυτά τα καθήκοντα αφού επιτρέπει το σπάσιμο των εκάστοτε καθηκόντων σε μικρότερα ανεξάρτητα μεταξύ τους κομμάτια που είναι ευκολότερο να διαχειριστεί κάνεις. Συνεπώς, οι προγραμματιστές χρησιμοποιούν διάφορους μηχανισμούς σπασίματος σε συστατικά στοιχεία

ώστε να είναι ευκολότερο να επαναχρησιμοποιήσουν, να εξελίξουν και να αναπτύξουν πιθανώς παράλληλα τα ανεξάρτητα συστατικά στοιχεία. Έτσι, το 2012, ο d'Amorim, F., & Borba, P. (d'Amorim & Borba, 2012) ανέλυσαν ποσοτικά την επιρροή που έχουν αυτοί οι μηχανισμοί. Εφαρμόζουν μια συγκεκριμένη τεχνική πάνω σε δυο συγκεκριμένες περιπτώσεις πληροφοριακών συστημάτων. Εξάγουν παραδοσιακές αλλά και σύγχρονες μετρικές που περιλαμβάνουν τη συνοχή(cohesion) αλλά και τη σύζευξη(coupling) για να καταφέρουν να αναλύσουν το Modularity από την άποψη ποιοτικών χαρακτηριστικών όπως η μεταβλητότητα και η συνδεσιμότητα (pluggability). Οι εν λόγω μετρικές είναι οι εξής:

- Αριθμός αρχείων πηγαίου κώδικα (Number of source files)
- Αριθμός κλάσεων που προστέθηκαν/αλλάχθηκαν (Number of added /changed classes/aspects)
- Αριθμός γραμμών κώδικα που προστέθηκαν/αλλάχθηκαν (Number of added lines of code (LOC))

Μετρώντας τα παραπάνω καταλήγουν ότι είναι συχνά δύσκολο, αν όχι ακατόρθωτο μερικές φορές, να σχεδιαστεί ένα σύστημα, οπού θα ανεξαρτητοποιηθούν τα συστατικά του χωρίς να επηρεαστούν αρνητικά η συντήρηση και η εξέλιξη του. Εξαρτάται από πληθώρα παραγόντων όπως οι τύποι των αλλαγών, η αρχιτεκτονική αλλά ακόμα και οι στόχοι των προγραμματιστών.

Ο βαθμός σύζευξης μεταξύ των συστατικών του λογισμικού (coupling) αποτελεί θεμελιώδη ιδιότητα των συστημάτων λογισμικού που συνδέεται στενά με την ποιότητα του σχεδιασμού του λογισμικού ενώ έχει μεγάλη επίπτωση στην κατανόηση του προγράμματος. Γενικότερα, ένα ιδανικό αντικειμενοστρεφές σύστημα χαρακτηρίζεται από χαμηλό coupling και υψηλή συνοχή (cohesion). Ωστόσο στη διάρκεια του κύκλου ζωής του λογισμικού η δομή ενός συστήματος συχνά αλλάζει επηρεάζοντας την εξέλιξη του. Για να εξασφαλιστεί λοιπόν μια αποδοτική διαδικασία συντήρησης που δε θα είναι κοστοβορα είναι απαραίτητο οι προγραμματιστές να υπολογίζουν με ακρίβεια την επίδραση των αλλαγών. Η μέθοδος που εξέτασαν το 2019 οι Istvan Gergely Czibula, Gabriela Czibula, Diana-Lucia Miholca και Zsuzsanna Onet-Marian (Czibula, Czibula, Miholca, & Onet-Marian, 2019) είναι η εκτίμηση του coupling. Εισήγαγαν μια καινοτόμο μέθοδο μέτρησης που περιέχει τόσο δομικά όσο και εννοιολογικα χαρακτηριστικά που βρίσκονται στα συστατικά του λογισμικού. Η πρόταση τους συνδυάζει τις πληροφορίες που προκύπτουν από τον γραπτό πηγαίο

κώδικα με τις δομικές σχέσεις μεταξύ των συστατικών του λογισμικού. Η νέα μετρική που προτείνουν ονομάζεται ACE (aggregated coupling of a software entity). Διεξήχθησαν πειράματα, μέσα από μελέτη τριών διαφορετικών περιπτώσεων (Commons DbUtils, RL framework, Apache Commons Collections), στα οποία συγκρίθηκαν πέντε παραδοσιακές μετρικές του coupling με την προτεινομένη μετρική ώστε να δοθεί έμφαση στην απόδοση που έχει η νέα μετρική στην ανάλυση μετάδοσης αλλαγών. Τα αποτελέσματα έδειξαν ότι η τελευταία μπορεί να βελτιωθεί αφού η μέτρηση του ACE επιτρέπει να εξακριβώσουμε αν μια αλλαγή μπορεί να απαιτήσει και τροποποίηση άλλων τεχνουργημάτων εκτός του πηγαίου κώδικα (κάτι που δεν επιτρέπουν οι παραδοσιακές μετρικές coupling).

Η παρακολούθηση project και η προσαρμογή του πλάνου είναι δυο σημαντικές δραστηριότητες της μηχανικής λογισμικού. Ένας παραγοντας-κλειδι για την αποτελεσματική διαχείριση ενός project είναι η έγκυρη και ακριβής διαδικασία του σχεδιασμού από την πρώτη στιγμή. Επειδή τα περισσότερα project παρεκκλίνουν από το αρχικό πλάνο είναι χρήσιμο να μπορεί να μετρηθεί ανά πασά στιγμή η ακριβής πρόοδος ώστε να προσαρμοστεί αντίστοιχα το συνολικό πλάνο. Η Αστάθεια Σχεδιασμού Συστήματος (System Design Instability (SDI)) είναι μια μετρική που προτάθηκε πίσω στο 2005 από τους Alshayeb, M., & Li, W. (Alshayeb & Li, 2005) ώστε να υποδηλώνει την πρόοδο ενός αντικειμενοστρεφούς project. Η μετρική παρέχει πληροφορίες σχετικά με την εξέλιξη του έργου ώστε οι διαχειριστές να προβούν σε πιθανή προσαρμογή. Στόχος ήταν ο έλεγχος αν το SDI μπορεί να συμβάλει στα παραπάνω και να μελετηθεί η εξέλιξή του σχεδιασμού ενός συστήματος μέσα στη διαδικασία ευέλικτου λογισμικού. Σε ημερήσια βάση έκαναν συλλογή δεδομένων από δυο συγκεκριμένα αντικειμενοστρεφή συστήματα τα οποία στη συνέχεια ανέλυαν για να καταλήξουν στο γεγονός ότι στα πρώιμα και στα τελευταία στάδια των project υπήρξαν δυο σταθερές αυξήσεις του SDI. Επίσης, ανακάλυψαν ότι το SDI σχετίζεται θετικά με την προσπάθεια για αλλαγή στο σχεδιασμό και αρνητικά με τις προσπάθειες για διόρθωση λαθών. Μια δεύτερη μετρική που εξέτασαν είναι ο όγκος των κλάσεων. Διαπίστωσαν ότι στο τελευταίο τέταρτο υπήρχε αξιοσημείωτη ανάπτυξη του όγκου των κλάσεων και στις δυο περιπτώσεις που εξετάστηκαν. Τα αποτελέσματα συγκρίθηκαν με προγενέστερη μελέτη με στόχο την επικύρωση τους και βρέθηκε ότι η παραδοσιακή ανάπτυξη λογισμικού και το Extreme programming ακολουθούν ίδιο μοτίβο.

Οι περισσότερες μετρικές μπορούν να εφαρμοστούν στο στάδιο της υλοποίησης δηλαδή αφού έχει ξεκινήσει να γράφεται κώδικας ή λίγο πιο πριν. Αναγνωρίζεται ότι λάθη ή κακές επιλογές στη φάση του σχεδιασμού συχνά προκαλούν δύσκολα προβλήματα αργότερα. Ο σωστός σχεδιασμός περιλαμβάνει τη σύγκριση επιλογών και εύρεση εναλλακτικών ώστε να επιλεχθεί η καλύτερη εν τελεί. Ωστόσο η επιλογή είναι δύσκολη καθώς δεν υπάρχει άμεση ανατροφοδότηση ή δεν υπάρχει γνώση των συνέπειων. Επιπρόσθετα, τα συστήματα λογισμικού, και ειδικά τα αντικειμενοστρεφή, εξελίσσονται συνεχώς. Παρόλο που στην αρχή φαίνεται όλα να λειτουργούν αψογά οι απαιτήσεις στην πορεία αλλάζουν δημιουργώντας ανάγκη για αλλαγές. Ευτυχώς η δυνατότητα συντήρησης του λογισμικού έχει απασχολήσει αρκετούς ερευνητές που έχουν προτείνει πληθώρα μετρικών. Τα αποτελέσματα τους έδειξαν ότι η σύζευξη έχει επιρροή στη συντηρησιμοτητα. Αυτές οι μελέτες ωθήσαν τους (Almugrin, Albattah, & Melton, 2016) να ασχοληθούν με τη σημασία του κόστους του μεταγενέστερου ελέγχου και συντήρησης αλλά και με τις μετρικές της σύζευξης, οι οποίες μπορούν να προβλέψουν τη συντηρησιμοτητα. Εμπνευσμένοι από τις αρχές του Martin και τη σχετική του εργασία του 2003 προτείναν μια νέα μετρική που αφορά την αστάθεια που ονομάζεται I (A) και σχετίζεται με τις εξαρτήσεις μεταξύ των αντικειμένων. Για τον υπολογισμό αυτής της μετρικής χρησιμοποίησαν ένα εξατομικευμένο εργαλείο μέσω του οποίου ανέλυσαν τη σύγκριση αναμεσά στις μετρικές του Martin και τις δίκες τους. Μέσα από την πειραματική μελέτη επτά συστημάτων ανοιχτού κώδικα (Camel 2.0.0, Tomcat 7.0.6, JHotDraw 7.5.1, JEdit 4.5.0, Hadoop 2.2.0, Synapse 2.1.0 and Ant 1.92) συμπέραναν ότι οι νέες μετρικές που προτείναν παράγουν καλές προβλέψεις για τη συντηρησιμοτητα.

Η σύνθεση του πηγαίου κώδικα απαιτεί την ένωση δυο ή περισσοτέρων συστατικών σε ένα πρόγραμμα. Υφίστανται ήδη μηχανισμοί σύνθεσης κώδικα που βοηθάν τους προγραμματιστές να υπολογίζουν την πολυπλοκότητα του προγράμματος και να διευκολύνουν την εξέλιξη του. Δυστυχώς όμως ο ρόλος της σύνθεσης του κώδικα δεν είχε κατανοηθεί πλήρως μέχρι το 2012. Υπήρχε ένα κενό εξαιτίας της έλλειψης ενός πλαισίου που θα ποσοτικοποιουσε τη σύνθεση του κώδικα. Οι υπάρχουσες μετρικές εστίαζαν στις ιδιότητες των συστατικών(modules) του προγράμματος. Επομένως, οι προγραμματιστές δεν ήταν εφικτό να αναλύσουν την επιρροή των ιδιοτήτων της σύνθεσης του κώδικα στην εξελιξιμοτητα του προγράμματος. Αυτό το κενό ήρθαν να καλύψουν οι Dantas, F., Garcia, A., & Whittle, J. (Dantas, Garcia, & Whittle, 2012)

παρουσιάζοντας ένα πλαίσιο το οποίο αποτελείται από ορολογίες και μια σουίτα μετρικών. Μέσα από τη μελέτη 22 διαφορετικών εκδόσεων τεσσάρων συγκεκριμένων project αποκαλύφθηκε ότι το προτεινόμενο πλαίσιο φέρει σαφείς ενδείξεις για την αστάθεια του λογισμικού. Ορίστηκαν τέσσερις μετρικές που χαρακτηρίζουν τη σύνθεση του κώδικα: Local Impact (LoI), Global Impact (GoI), Composition Volatility (CoV) και Depth of Dependency Chain (DDC). Οι δυο πρώτες αφορούν το πεδίο εφαρμογής του κώδικα ενώ οι δυο δεύτερες την αστάθεια. Οι ιδιότητες της σύνθεσης φάνηκε να υπολογίζουν καλυτέρα την αστάθεια από ότι συμβατικές ιδιότητες όπως η σύζευξη.

Η ανάλυση μετάδοσης αλλαγών σχετίζεται με την εξακρίβωση των συνέπειων, που προκαλούνται από τις τροποποιήσεις σε ένα μέρος του συστήματος, σε άλλο μέρος του συστήματος, γνωστές και ως αλυσιδωτές αντιδράσεις (ripple effect). Η μελέτη των αντιδράσεων/ επιδράσεων αυτών μπορεί να έχει προτερήματα πριν από την πραγματική εφαρμογή μιας αλλαγής αλλά και αφού μια αλλαγή έχει υλοποιηθεί. Πάρα τα προτερήματα αυτά λίγες είναι οι μετρικές που υπολογίζουν κατά ποσό μια κλάση είναι επιρρεπής στο φαινόμενο των αλυσιδωτών αντιδράσεων. Αυτό το κενό ήρθαν το 2015 να καλύψουν οι Arvanitou, E.-M., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (Arvanitou, Ampatzoglou, Chatzigeorgiou, & Avgeriou, 2015) οι οποίοι ελέγχοντας τους ορισμους των πιο δημοφιλών μετρικών σύζευξης, παρατήρησαν ότι καμία δεν ήλεγχε ταυτόχρονα τον αριθμό των εξαρτήσεων μια κλάσης και το συντελεστή διάδοσης. Έτσι, προτείναν μια νέα μετρική σύζευξης που ονόμασαν Μέτρηση Αλυσιδωτών Αντιδράσεων (Ripple Effect Measure (REM)) που βρίσκει εφαρμογή στη φάση του σχεδιασμού. Η επικύρωση της μετρικής έγινε με 2 τρόπους:

- Θεωρητικά, υπολογίζοντας μαθηματικά τις βασικές ιδιότητες του λογισμικού που είχαν προταθεί από τον Briand
- Πρακτικά, επιβεβαιώνοντας την εγκυρότητα της μετρικής συγκρίνοντας την με τις υπάρχουσες μετρικές σύζευξης

Για το 2ο σκέλος χρειάστηκε μια διαδικασία εμπειρικής μελέτης δυο λογισμικών ανοικτού κώδικα, το Jflex και το Jmol. Ο κύριος λόγος που χρησιμοποιήθηκαν τα συγκεκριμένα λογισμικά ήταν η πρόθεση επαναχρησιμοποίησης υπάρχοντος συνόλου δεδομένων (dataset) που έχει αναπτυχθεί και χρησιμοποιηθεί για ερευνά με παρομοίους σκοπούς. Η REM μετρήθηκε με τη χρήση υπάρχοντος εργαλείου που ονομάζεται

ClassInstability το οποίο τροποποιήθηκε ελαφρώς. Τα αποτελέσματα έδειξαν ότι η μετρική μπορεί να αξιολογήσει έγκυρα την πιθανότητα μια κλάση να αλλάξει λόγω των αλυσιδωτών αντιδράσεων, σε σύγκριση με τις υπάρχουσες μετρικές.

Οι προγραμματιστές αλλάζουν τον πηγαίο κώδικα για να προσθέσουν μια νέα λειτουργικότητα είτε για να διορθώσουν bugs. Αυτές οι αλλαγές επιφέρουν άμεση επίδραση στην ποιότητα ή στη σταθερότητα. Παρόλα αυτά αυτές οι επιδράσεις πιθανώς θα έχουν αντίκτυπο μακροπρόθεσμα. Ο στόχος του Herzig, K. S. (Herzig, 2010) ήταν να ερευνήσει τις μακροπρόθεσμες επιδράσεις ανιχνεύοντας συσχετίσεις αναμεσά στις αλλαγές του κώδικα και μετρώντας την επιρροή τους στην ποιότητα του λογισμικού. Χρησιμοποίησε απλές παραστάσεις εξαρτήσεων (transaction dependency graphs (TDGs)) οι οποίες βασίζονται στις συσχετίσεις αναμεσά σε σύνολα ταυτόχρονα εφαρμοσμένων αλλαγών του πηγαίου κώδικα, τις λεγόμενες συναλλαγές (transactions). Η σχέση αναμεσά στον αριθμό των μοναδικών συναλλαγών και των αριθμών των συναλλαγών που πραγματοποιήθηκαν πρόσφατα καθορίζει την επιρροή μιας απλής συναλλαγής αργότερα. Έτσι, ορίζεται η μετρική της Επιρροής που βασίζεται στις παραπάνω τιμές. Με τη χρήση ενός απλού παραδείγματος στην εισαγωγή καλείται ο αναγνώστης να σκεφτεί ένα παράδειγμα οπού μια ομάδα προγραμματιστών πρέπει να αυξήσει την απόδοση της εφαρμογής δημοσιεύοντας έναν μέχρι πρότινος κρυφό αλγόριθμο.

Με την πίεση για κατασκευή επαναχρησιμοποιήσιμου λογισμικού συνεχώς να αυξάνεται, υπάρχει η ανάγκη συνεχούς ερευνάς για μηχανισμούς που θα βελτιώσουν τη σύνδεση των ανεξάρτητων μονάδων. Ένας συνεχώς αυξανόμενος αριθμός τεχνικών προγραμματισμού έχει προκύψει για να υποστηριχθεί αυτού του είδους η σύνθεση. Παρόλο που αυτές οι τεχνικές ποικίλουν ως προς τη μορφή και τους συμβολισμούς τους, όλες προωθούν μια στροφή στον τρόπο που κατασκευάζονται τα προγράμματα. Ωστόσο, αν και γνωρίζουμε την πολυπλοκότητα της σύνθεσης του κώδικα, δεν είναι αρκετά κατανοητές οι επιδράσεις των τεχνικών στην ποιότητα του λογισμικού. Έτσι, το 2011, το Opel Research Group αποφάσισε να συγκρίνει την επιρροή των τεχνικών ταυτόχρονα στην επαναχρησιμοποίηση και τη συντήρηση του λογισμικού. Για να το πέτυχει αυτό, όρισε ένα σύνολο από μετρικές που προσανατολίζονται στη σύνθεση και σύγκρινε την αποτελεσματικότητα του με αυτήν ενός συνόλου παραδοσιακών μετρικών που καθορίζουν κατά ποσό τα στοιχεία είναι ξεχωριστά. Η ανάλυση αφορά τη φάση του σχεδιασμού και έγινε μέσω διαγράμματος κλάσεων. Αναμένεται

να συνεισφέρει παρέχοντας αποδείξεις για τα πλεονεκτήματα και τα μειονεκτήματα των μηχανισμών σύνθεσης στα πλαίσια της αστάθειας λογισμικού και της επαναχρησιμοποίησης αλλά και να παρέχει ένα σύνολο μετρικών που θα ποσοτικοποίησή τις ιδιότητες της σύνθεσης θεωρητικά και πρακτικά.

Με στόχο την ελαχιστοποίηση των μετέπειτα ελέγχων και του κόστους της αναβάθμισης της τεχνολογίας για τα συστήματα ασφάλειας είναι απαραίτητη η πλήρης κατανόηση και ανάλυση των συσχετίσεων στη φάση της αρχιτεκτονικής. Οι μη διαχωρίσιμες συσχετίσεις δημιουργούν υπερβάσεις του κόστους ενώ ρίχνουν την ποιότητα στα συστήματα. Παρόλα αυτά, η ανάλυση των αρχιτεκτονικών συσχετίσεων στην πράξη συνήθως γίνεται αναδρομικά χρησιμοποιώντας τη δομή του πηγαίου κώδικα. Τέτοιου είδους ανάλυση δεν εντοπίζει σημαντικές συσχετίσεις που θα μπορούσαν να έχουν έρθει στην επιφάνεια νωρίτερα. Η περιγραφή της αρχιτεκτονικής λογισμικού και οι ξεκάθαρες απαιτήσεις μπορούν να βελτιώσουν τη διαδικασία της ανάλυσης, αν και η ποιότητα τους, η συνοχή τους και το περιεχόμενο τους συχνά ποικίλουν. Το 2014 Nord, R. L., Ozkaya, I., Sangwan, R. S., & Koontz, R. J. (Robert L. Nord, 2014) ανέλυσαν μια από τις πιο δημοφιλείς μετρικές συσχέτισης, τη σταθερότητα. Εφάρμοσαν μια τεχνική οπτικοποίησης πάνω σε ένα βιομηχανικό λογισμικό, στο οποίο ένας από τους συγγράφεις είχε συμβάλει ως αρχιτέκτονας λογισμικού, με στόχο τη μείωση των ελέγχων (testing). Αναλύουν τις δομικές συσχετίσεις ποσοτικά χρησιμοποιώντας εργαλεία. Μέσω της χρήσης διαγραμμάτων στοιχείων (component diagrams) συγκρίνουν τη σταθερότητα του συστήματος πριν και Μετα την εφαρμογή των βελτιώσεων. Συμπεραίνεται εν τελεί πως η ανάλυση των συσχετίσεων των δομικών στοιχείων δεν επαρκεί για ένα σύστημα του οποίου η αρχιτεκτονική έχει ξαναδημιουργηθεί. Ομοίως τα υπάρχοντα εργαλεία ανάλυσης συσχετίσεων υπολείπονται όσον αφορά στα ποιοτικά χαρακτηριστικά της αρχιτεκτονικής.

Το μεγαλύτερο κομμάτι της συνολικής προσπάθειας στη βιομηχανία ανάπτυξης λογισμικού είναι αφιερωμένο στην εξέλιξη του. Τα αντικειμενοστρεφή συστήματα δεν αποτελούν εξαίρεση. Η πρόβλεψη της αστάθειας του λογισμικού στη φάση του σχεδιασμού μπορεί να μειώσει αυτά τα κόστη συντήρησης. Η πλειοψηφία των τεχνικών που χρησιμοποιούνται για να χτιστούν μοντέλα πρόβλεψης βασίζονται σε μοτίβα σε ένα δείγμα δεδομένων. Ωστόσο, η έλλειψη αντιπροσωπευτικών δεδομένων δυσχεραίνει τα ασφαλή συμπεράσματα. Οι Grosser, D., Sahraoui, H. A., & Valtchev, P. (David Grosser, 2002) αποφάσισαν να εξετάσουν την αστάθεια βασισμένοι

σε άλλες μετρικές. Συγκρίναν δυο διαφορετικές εκδόσεις δυο διαφορετικών συστημάτων από οπού προέκυψαν 691 περιπτώσεις. Προέκυψαν 22 μετρικές που ομαδοποιήθηκαν σε 4 κατηγορίες: σύζευξη, συνοχή, μέγεθος, ιεραρχία. Οι μετρικές ήταν οι ακόλουθες: COH,LCOMB,COM,COMI,OCMAIC,CUBF,CUB,OMAEC,NOC,NOP,NON,NOCON,DIT,MDS,CHM,NOM,NPA,NPPM,WMC,MCC,DEPCC,WMCLOC. Για την μέτρηση των παραπάνω χρησιμοποιήθηκε η στρατηγική που είναι γνωστή και ως case-based reasoning(CBR) σε συνδυασμό με διάγραμμα κλάσεων. Τα αποτελέσματα έδειξαν ότι η σταθερότητα μιας κλάσης μπορεί να προβλεφθεί με εμπιστοσύνη ως και 75% με την παραπάνω στρατηγική.

Μια σημαντική δραστηριότητα στον τομέα της μηχανικής συστημάτων είναι η ανάλυση του τρόπου που μια αλλαγή στις απαιτήσεις θα επιδράσει στο σχεδιασμό του συστήματος. Η χειροκίνητη ανάλυση μπορεί να αποβεί κοστοβορα, ειδικά σε πολύπλοκα συστήματα. Για αυτό οι Nejati, Shiva, Sabetzadeh, Mehrdad, Arora, Chetan, Briand, Lionel C. and Mandoux, Felix (Nejati, Sabetzadeh, Arora, Briand, & Mandoux, 2016) προτείναν μια νέα προσέγγιση ώστε να εντοπίζεται με αυτοματοποιημένο τρόπο η επίδραση που θα έχει η αλλαγή των απαιτήσεων στη φάση του σχεδιασμού όταν οι απαιτήσεις εκφράζονται χρησιμοποιώντας μοντέλα. Βασίστηκαν στη Systems Modeling Language (SysML) λόγω της συνεχώς αυξανομένης χρήσης της στη βιομηχανία. Η προσέγγιση χρησιμοποίησε την πραγματική περίπτωση χρήσης ενός cam phaser της εταιρείας Delphi Automotive που περιλαμβάνει μηχανικά, ηλεκτρονικά αλλά και στοιχεία λογισμικού. Χωρίζεται σε δυο βήματα. Δοθείσης μιας αλλαγής, αρχικά εφάρμοσαν ένα στατικό αλγόριθμο για να εξάγουν ένα σύνολο στοιχείων που θα επηρεαστούν. Στη συνέχεια τα ταξινομήσαν σύμφωνα με την ποσοτική μέτρηση που προέβλεπε ποσό πιθανό είναι να επηρεαστεί ένα στοιχείο. Δεκαέξι πραγματικές αλλαγές απαιτήσεων εξετάστηκαν και έδειξαν ότι οι μηχανικοί, χρησιμοποιώντας την προσέγγιση τους, χρειάζεται να εξετάσουν μόλις το 4,8% του συνολικού σχεδιασμού για να αποφασίσουν ακριβώς ποια στοιχεία θα επηρεαστούν εν τελεί. Επιπλέον, τα αποτελέσματα βελτιώνονται ακόμα περισσότερο αν ληφθούν υπόψιν τα διαγράμματα απαιτήσεων (requirements diagram) και τα διαγράμματα δραστηριοτήτων (activity diagram).

Ο έλεγχος που βασίζεται σε μοντέλα βοηθά τους μηχανικούς να αυτοματοποιούν τις διαδικασίες ελέγχου ώστε να μειώνουν τα κόστη τους. Όταν ένα μοντέλο αλλάζει εξαιτίας της λογικής του εξέλιξης είναι σημαντικό να διατηρηθούν ενημερωμένες και οι σουίτες δοκιμής παλινδρόμησης

(regression testing). Το να δημιουργηθεί εξ αρχής ολόκληρη η σουίτα για ένα νέο μοντέλο, αν και μη αποδοτικό, ακόμα προτιμάται συχνά στη βιομηχανία, συμπεριλαμβανομένης της ιδίας της Microsoft. Όταν καταλήξει κάνεις σε μια σουίτα ελέγχων ο πιο σημαντικός σκοπός είναι να επιτευχθεί η κάλυψη των απαιτήσεων. Με σκοπό να προλάβουν τις εξελίξεις των απαιτήσεων οι Jiang, B., Tse, T. H., Grieskamp, W., Kicillof, N., Cao, Y., Li, X., & Chan, W. K (Chan, 2011) ανέπτυξαν μια τεχνική για να εξακριβώσουν τις επαναχρησιμοποιήσιμες περιπτώσεις χρήσης από την πρωτότυπη σουίτα. Επίσης ανέπτυξαν μια βελτίωση αυτής της τεχνικής που θα καλύπτει και τις νέες περιπτώσεις χρήσης για να καλύψουν και τα κομμάτια εκείνα που θα έχουν αλλαγές. Τα πειράματα τους διεξήχθησαν πάνω σε τέσσερα μεγάλα project δοκίμων και απέδειξαν ότι μπορούν να εντοπίσουν με μεγάλη επιτυχία υψηλό ποσοστό επαναχρησιμοποιήσιμων περιπτώσεων χρήσης. Συγκρινόμενη με την περίπτωση της καθολικής δημιουργίας ολόκληρης της σουίτας, η τεχνική τους μειώνει αισθητά τον χρόνο που χρειάζεται για τις δοκιμές παλινδρόμησης. Διατηρεί επίσης τη σταθερότητα στην κάλυψη των απαιτήσεων. Εν τελεί, η περαιτέρω ανάλυση, αποκάλυψε μια χρήσιμη πρακτική μοντελοποίησης που επιτρέπει στους μηχανικούς να τροποποιούν τα μοντέλα τους με τέτοιο τρόπο που η επαναχρησιμοποίηση περιπτώσεων ελέγχου (test cases) μπορεί να βελτιωθεί.

Η χρησιμότητα των μετρικών ήταν πάντοτέ αδιαμφισβήτητά, ακόμα και στο μακρινό 2001 από έναν φοιτητή, ονόματα Schneidewind, N. F (Schneidewind, 1998) ο οποίος εξέτασε τη σκοπιμότητα της μέτρησης της σταθερότητας και κατέγραψε την ανάγκη ενός μοντέλου πρόβλεψης στα παλαιού τύπου συστήματα( ). Το Πρόγραμμα Διαστημικού Λεωφορείου (Space Shuttle Application) επιλέχθηκε για αξιολόγηση των παραπάνω. Βασισμένος σε άλλες μετρικές που προκύπτουν από τον πηγαίο κώδικα, όπως οι γραμμές κώδικα που έχουν αλλαχθεί, κατέληξε στη Change Metric (CM). Εν τελεί κατέληξε στο ότι η εκτίμηση ενός ξεπερασμένου συστήματος είναι σημαντική καθώς ακόμα χρησιμοποιούνται σε τομείς όπως οι επικοινωνίες.

Καθώς η κοινωνία αυξάνει την ανάγκη της για τα συστήματα λογισμικού αυξάνεται και η ανάγκη για μεγαλύτερη ασφάλεια. Έχοντας διεισδύσει σε κάθε τομέα της καθημερινότητας πρέπει να εξασφαλίσουμε ότι δεν υπάρχει έκθεση σε ρίσκο για τον κόσμο, τις ιδιοκτησίες αλλά και το περιβάλλον. Αυτό περιλαμβάνει τη συλλογή αποδείξεων με τη μορφή των αναλύσεων ασφάλειας, προδιάγραφες συστημάτων και αποτελέσματα ελέγχων. Όλα αυτά εξελίσσονται στη διάρκεια του κύκλου ζωής ενός συστήματος

δημιουργώντας μια ανάγκη για ανάλυση αντικτύπου αλλαγής με τέτοιο τρόπο που να εξασφαλίζεται ότι δε θα μειωθεί η ασφάλεια. Για αυτό το λόγο οι de la Vara, J. L., Borg, M., Wnuk, K., & Moonen, L. (de la Vara, Borg, Wnuk, & Moonen, 2016) αποφασισαν να ρίξουν φως στη διαδικασία της ανάλυσης της μετάδοσης αλλαγών ως προς την ασφάλεια στην πράξη. Η γνώση για αυτή τη δραστηριότητα είναι μικρή παρόλη την ερευνά που έχει γίνει γενικότερα για την ανάλυση μετάδοσης αλλαγών. Διεξήγαγαν μια έρευνα στη βιομηχανία σχετικά με τις συνθήκες υπό τις οποίες η ανάλυση των επιπτώσεων στην ασφάλεια αντιμετωπίζεται, τα εργαλεία που χρησιμοποιούνται και τις προκλήσεις που αντιμετωπίζουν. Μάλιστα, σε αντίθεση με τις περισσότερες έρευνες, δεν εστίασαν σε μια συγκεκριμένη φάση ανάπτυξης του λογισμικού, αλλά μελέτησαν τόσο τις προδιάγραφες της φάσης του σχεδιασμού και των απαιτήσεων όσο και τη φάση της υλοποίησης και του ελέγχου. Οι πιο δυνατοί συσχετισμοί που εντοπίστηκαν αναμεσά στις φάσεις υλοποίησης είναι οι εξής:

- Προδιάγραφες απαιτήσεων και σχεδιασμού
- Προδιάγραφες απαιτήσεων και πηγαίος κώδικας
- Έλεγχος περιπτώσεων και πηγαίος κώδικας
- Αποτελέσματα ανάλυσης ασφάλειας και πηγαίος κώδικας
- Προδιάγραφες συνθήκων λειτουργείας και πηγαίος κώδικας

Τα αποτελέσματα της ερευνάς έδειξαν ότι πρακτικά η ανάλυση μετάδοσης αλλαγών ως προς τις αποδείξεις ασφάλειας (Safety evidence change impact analysis (SECIA)) συμβαίνει σε ποίκιλες καταστάσεις ενώ οι αλλαγές στον κώδικα επηρεάζουν διάφορους τομείς της ασφάλειας. Υποστήριξη από εργαλεία υπάρχει αλλά είναι ανεπαρκής και συχνά η βελτίωση της αποτελεί μια πρόκληση.

Η καταμέτρηση και η αξιολόγηση της σταθερότητας των διαδικασιών συντήρησης είναι σημαντική και χάρη στη σχέση μεταξύ της ποιότητας της διαδικασίας και της ποιότητας του προϊόντος. Η επόμενη ερευνά εστιάζει σε ένα σημαντικό κριτήριο ποιότητας: την αξιοπιστία. Σε μια διαδικασία ανάλυσης της σταθερότητας μιας διαδικασίας συντήρησης, είναι σημαντικό να μην αντιμετωπίζεται ξεχωριστά από την αξιοπιστία. Επιπλέον, πρέπει να ληφθεί υπόψιν ότι η αποτελεσματικότητα της προσπάθειας για έλεγχο (test) αποτελεί επίσης κομμάτι της διαδικασίας και κριτήριο αξιοπιστίας. Η σχέση μεταξύ ποιότητας των παραγομένων και της ωριμότητας της διαδικασίας έχει αναγνωριστεί ως πρωτεύον ζήτημα στη μηχανική λογισμικού καθώς κάθε βελτίωση στη διαδικασία συνεπάγεται και βελτιώσεις στα παραγόμενα

προϊόντα. Ο Schneidewind, N. F. (1999). ({Schneidewind, 1999) αποφάσισε να ερευνήσει την αστάθεια όπως αυτή ορίζεται και αξιολογείται από τις τάσεις και τις αλλαγές στη διάρκεια των releases αλλά και εντός ενός release. Χρησιμοποίησε τη μελέτη του Διαστημικού Λεωφορείου της NASA προκειμένου να εξηγήσει την προσέγγιση του. Μέσω αυτής ανέπτυξε μια νέα μετρική, την Change Metric, της οποίας ο υπολογισμός βασίζεται στις αλλαγές που γίνονται στον πηγαίο κώδικα(Failures/KLOC decrease). Οι θετικές τιμές σημαίνουν σταθερότητα αντίθετα με τις αρνητικές που σημαίνουν αστάθεια. Κατέληξε στο γεγονός ότι είναι δύσκολο να συσχετισθεί μια συγκεκριμένη βελτίωση του παραγομένου με συγκεκριμένη βελτίωση της διαδικασίας. Σε κάθε περίπτωση όμως προτείνει τη χρήση των τιμών της μετρικής ώστε να δοθεί έμφαση σε πιθανά προβλήματα της διαδικασίας συντήρησης.

Η επόμενη δημοσίευση παρουσιάζει μια προσέγγιση που συνδυάζει εννοιολογικές και εξελικτικές τεχνικές ώστε να υποστηριχθεί η ανάλυση μετάδοσης αλλαγών (IA) στον πηγαίο κώδικα. Οι πληροφορίες αντλούνται με μεθόδους ανάκτησης που βασίζονται σε τεχνουργήματα λογισμικού σε μορφή κειμένου (textual Software artifacts) που βρίσκονται σε μια συγκεκριμένη έκδοση λογισμικού όπως πχ σχόλια ή αναγνωριστικά (identifiers) σε ένα κομμάτι του πηγαίου κώδικα. Ο σκοπός των Kagdi, H., Gethers, M., & Poshyvanyk, D. (Kagdi, Gethers, & Poshyvanyk, 2013) ήταν να αναπτύξουν μια νέα και βελτιωμένη τεχνικη IA χρησιμοποιώντας ορισμένες από τις υπάρχουσες λύσεις. Η βασική τους φιλοσοφία ήταν ότι η κυριά πηγή της πληροφορίας είναι ο άνθρωπος (πχ σχόλια, identifiers) και όχι κάποια πιο επίσημη μορφή όπως γραφήματα και στατικές συσχετίσεις. Με αυτή την προοπτική, χρησιμοποίησαν τη μελέτη των αλλαγών πραγματικών συστημάτων ανοιχτού λογισμικού όπως Apache httpd, ArgoUML, iBatis, Koffice και jEdit, πάνω στα οποία ελέγξαν αν η υπόθεση τους έχει τελικά εφαρμογή. Τα αποτελέσματα της μελέτης έδειξαν ότι ο συνδυασμός τεχνικών Ανάκτησης Πληροφορίας αλλά και Αναζήτησης σε Αποθήκες Πληροφορίας (Mining Software Repositories) παρέχει στατιστικά σημαντικές βελτιώσεις στην ακρίβεια της IA σε σχέση με την ανεξάρτητη χρήση τους. Επίσης, συμπέραναν ότι η ανάλυση μεγαλύτερων χρονικών διαστημάτων μπορεί να βελτιώσει τα αποτελέσματα της συνδυαστικής χρήσης τεχνικών.

Η επαναχρησιμοποίηση κομματιών πηγαίου κώδικα είναι κοινή πρακτική στον προγραμματισμό. Τα κοινά αυτά κομμάτια κώδικα ονομάζονται κλώνοι. Οι κλώνοι έχουν αναδειχθεί ως ένα αμφιλεγόμενος ορός στη μηχανική

λογισμικού όσον αφορά στην ερευνά αλλά και στην πράξη. Η επίδραση των κλώνων έχει μεγάλη επίδραση στις προοπτικές συντήρησης λογισμικού. Η σταθερότητα (ή αστάθεια) έχει ερευνηθεί αρκετά ως προς τις επιδράσεις που έχουν οι κλώνοι στη συντήρηση του λογισμικού. Εφόσον ο κλωνοποιημένος κώδικας τείνει να έχει μεγαλύτερη αστάθεια από τον μη-κλωνοποιημένος, είναι αναμενόμενο ο πρώτος να απαιτεί και μεγαλύτερη προσπάθεια / κόστος για τη συντήρηση του. Ωστόσο, δε συμφωνούν όλες οι μελέτες σε αυτό. Ένας πιθανός λόγος πίσω από τα αντιφατικά ευρήματα είναι η διεξαγωγή των διαφορετικών μελετών σε διαφορετικές πτυχές της σταθερότητας καθώς και η χρησιμοποίηση διαφορετικών εργαλείων ανίχνευσης κλώνων σε διαφορετικά συστήματα αλλά και με διαφορετικά πειράματα. Εμβαθύνοντας σε τέτοιου είδους ζητήματα οι Mondal, M., Rahman, M. S., Roy, C. K., & Schneider, K. A. (Mondal, Rahman, Roy, & Schneider, 2018) διεξήγαγαν πειράματα οπού:

1. Εφάρμοσαν και διερευνήσαν επτά διαφορετικές μεθοδολογίες καθεμιά από τις οποίες εξερευνούσε άλλη πτυχή της αστάθειας
2. Χρησιμοποίησαν δυο διαφορετικά εργαλεία εντοπισμού κλώνων (NiCad και CCFinderX)
3. Εξέτασαν την αστάθεια τριών τύπων κλώνων (Type-1, Type-2, Type-3)

Τα αποτελέσματα αυτής αλλά και προηγουμένων δημοσιεύσεων έχουν καταλήξεις σε οκτώ μετρικές:

1. Συχνότητα τροποποιήσεων (Modification frequency) πηγαίου κώδικα
2. Πιθανότητα τροποποιήσεων (Modification probability) πηγαίου κώδικα
3. Μέσες τελευταίες ημερομηνίες αλλαγών (Average last change dates) πηγαίου κώδικα
4. Μέση ηλικία (Average age) πηγαίου κώδικα
5. Επίδραση (Impact) αλλαγών στον κώδικα
6. Πιθανότητα (Likelihood) αλλαγών στον κώδικα
7. Μέση αστάθεια ανά κλωνοποιημένη μέθοδο (Average Instability per cloned method) εξαιτίας του πηγαίου κώδικα
8. Διασπορά αλλαγής (Change dispersion) πηγαίου κώδικα

Αν και διαπιστώθηκε ότι οι κλώνοι είναι γενικά πιο ασταθείς, τα στατιστικά αποτελέσματα έδειξαν ότι η διαφορά της αστάθειας αναμεσά στον κλωνοποιημένος και μη κώδικα είναι αμελητέα για τις επτά πρώτες μετρικές. Από τις παραπάνω μετρικές μόνο για την τελευταία εντοπίστηκε σημαντική διαφορά στη αστάθεια του κλωνοποιημένος κώδικα.

Δεδομένης της σημασίας της εξέλιξης και της αρχιτεκτονικής για πολλά συστήματα λογισμικού η επόμενη δημοσίευση επικεντρώνεται στην αρχιτεκτονική εξέλιξη, η οποία μπορεί να εξεταστεί από δύο προοπτικές: Αφενός, ο αρχιτεκτονικός σχεδιασμός θα διευκολύνει την εξέλιξη του συστήματος, αφετέρου, η διαδικασία εξέλιξης πρέπει να γνωρίζει την αρχιτεκτονική του συστήματος και να υποστηρίζει τον εννοιολογικό του ρολό. Απώτερος σκοπός των Wermelinger, M., Yu, Y., Lozano, A., & Capiluppi, A. (Wermelinger, Yu, Lozano, & Capiluppi, 2011) στην κατανόηση της πρακτικής της αρχιτεκτονικής εξέλιξης. Παρουσίασαν ένα απλό δομικό μοντέλο με συ σχετιζόμενες ιστορικές μετρικές και απεικονίσεις που θα μπορούσαν να αποτελούν την απαρχή (dashboard) ενός αρχιτέκτονα λογισμικού. Μελέτησαν την περίπτωση χρήσης του Eclipse SDK, λόγω του όγκου, της πολυπλοκότητας και της μακροζωίας του συστήματος. Ο στόχος ήταν διπλός. Από τη μια, μελέτησαν αν υπάρχουν ορισμένα διδάγματα που θα μπορούσαν να αποκομίσουν για την καλύτερη πρακτική της αρχιτεκτονικής εξέλιξης. Από την άλλη, πήραν περισσότερες γνώσεις σχετικά με την εφαρμογή τέτοιων αρχών. Μέσω της ανάλυσης αρχιτεκτονικών στοιχείων, διαπίστωσαν ότι ορισμένες αρχιτεκτονικές αρχές δεν είναι ευκολά μετρήσιμες ή δεν έχει ιδιαίτερο νόημα σε αρχιτεκτονικό επίπεδο ή γενικότερα με την εξέλιξη της αρχιτεκτονικής.

Προηγούμενες μελέτες έχουν καταδείξει τη σχέση μεταξύ σύζευξης και εξωτερικών χαρακτηριστικών ποιότητας λογισμικού, όπως η τάση για σφάλματα, και την εφαρμογή της σύζευξης σε εργασίες συντήρησης λογισμικού, όπως η ανάλυση μετάδοσης αλλαγών. Αυτές οι προηγούμενες μελέτες επικεντρώνονται στη σύζευξη κλάσεων. Ωστόσο παρατηρείται ένα συνεχώς αυξανόμενο ενδιαφέρον για τη μελέτη χαρακτηριστικών που εφαρμόζονται σε διάφορες κλάσεις ταυτόχρονα. Η απορία που προσπάθησαν να λύσουν, Revelle, M., Gethers, M., & Poshyvanyk, D. (Revelle, Gethers, & Poshyvanyk, 2011) είναι αν η μέτρηση της σύζευξης σε επίπεδο χαρακτηριστικών είναι όντως χρήσιμη. Ορίσαν καινούριες μετρικές σύζευξης χαρακτηριστικών (feature coupling metrics) με βάση το δομή και το κείμενο του πηγαίου κώδικα. Από αυτές σχετίζονται άμεσα με την αστάθεια λογισμικού οι εξής:

1. Σύζευξη δομικών χαρακτηριστικών (Structural Feature Coupling)
2. Σύζευξη δομικών χαρακτηριστικών που περιλαμβάνει τις καλούμενες και τις καλούσες μεθόδους ενός χαρακτηριστικού (Structural Feature Coupling')
3. Σύζευξη χαρακτηριστικών κειμένου κώδικα (Textual Feature Coupling)

4. Μέγιστη Σύζευξη χαρακτηριστικών κειμένου κώδικα (Maximum Textual Feature Coupling)
5. Υβριδική Σύζευξη χαρακτηριστικών (Hybrid Feature Coupling)

Κατ΄επεκταση, διεξήγαγαν τρεις εκτενείς μελέτες περιπτώσεων χρήσης ώστε να αξιολογήσουν τις παραπάνω μετρικές και να λυθεί η αρχική τους απορία. Η πρώτη μελέτη εξετάζει τη σχέση μεταξύ σύζευξης χαρακτηριστικών και τάσης για σφάλματα, η δεύτερη εκτιμά τη σύζευξη χαρακτηριστικών στα πλαίσια της ανάλυσης μετάδοσης αλλαγών, ενώ η τρίτη αφορά ένα ερωτηματολόγιο μεταξύ των προγραμματιστών ώστε να καταλήξουν αν οι παραπάνω μετρικές ταιριάζουν με ό,τι οι ίδιοι οι προγραμματιστές θεωρούν ως χαρακτηριστικά σύζευξης. Και οι τρεις μελέτες παρείχαν αποδείξεις πως οι νέες μετρικές είναι όντως χρήσιμες καθώς μπορούν να βοηθήσουν σε εντοπισμό bug, καθορισμό των μετέπειτα ελέγχων και εκτίμηση της επίδρασης των αλλαγών.

Στην πληροφορική, ένα σχεδιαστικό πρότυπο ή σχεδιαστικό μοτίβο (design pattern) ορίζεται ως μία αποδεδειγμένα καλή λύση που έχει εφαρμοστεί με επιτυχία στην επίλυση ενός επαναλαμβανόμενου προβλήματος σχεδίασης συστημάτων λογισμικού. Αφενός, τα σχεδιαστικά πρότυπα προσφέρουν λύσεις σε επαναλαμβανόμενα προβλήματα σχεδιασμού, με στόχο την αύξηση της επαναχρησιμοποίησης, της ευελιξίας και της συντήρησης. Ωστόσο, έχει διαπιστωθεί από πολλές προηγούμενες μελέτες ότι κάποια πρότυπα, όπως αυτή των Observer and Singleton, συσχετίζονται με εκτενείς δομές κώδικα και υποστήριξαν ότι είναι πιο πιθανό να είναι επιρρεπείς σε σφάλματα. Από την άλλη μέρα, τα αντι-προτυπα (anti-patterns) περιγράφουν φτωχές λύσεις στα προβλήματα σχεδίασης και υλοποίησης που επισημαίνουν τις αδυναμίες στο σχεδιασμό συστημάτων λογισμικού και που μπορεί να επιβραδύνουν τη συντήρηση και να αυξήσουν τον κίνδυνο των σφαλμάτων. Οι κλάσεις που συμμετέχουν σε πρότυπα και αντι-προτυπα εξαρτώνται από άλλες κλάσεις , όπως πχ στατικές ή δυναμικές εξαρτήσεις, με αποτέλεσμα να υπάρχει κίνδυνος διάδοσης προβλημάτων. Όλα τα παραπάνω ζητήματα θέλησαν να εκτιμήσουν οι Jaafar, F., Guéhéneuc, Y.-G., Hamel, S., Khomh, F., & Zulkernine, M. (Jaafar, Guéhéneuc, Hamel, Khomh, & Zulkernine, 2016) σε σχέση με την αστάθεια λογισμικού. Διερευνήσαν την επίδραση τέτοιων εξαρτήσεων σε αντικειμενοστρεφή συστήματα μελετώντας τις σχέσεις μεταξύ της παρουσίας στατικών και δυναμικών εξαρτήσεων και 1. Της τάσης για σφάλματα 2. Των τύπων των αλλαγών και 3. Των τύπων των σφαλμάτων που αυτές οι κλάσεις παρουσιάζουν. Ανέλυσαν έξι σχεδιαστικά πρότυπα και δέκα αντι-

προτυπα σε 39 εκδόσεις των ArgoUML, JfreeChart και XercesJ. Για να προσδιορίσουν τους τύπους αλλαγών και σφαλμάτων που πολλαπλασιάζονται ανάλογα με τα σχεδιαστικά (αντί)πρότυπα, χρησιμοποίησαν ορισμένες αποθήκες λογισμικού (Software repositories) όπως αρχεία καταγραφής και Bugzilla. Κατέγραψαν τις γραμμές του κώδικα που προστέθηκαν, διαγράφηκαν ή άλλαξαν, όπως τις καταγράφηκαν από το diff, ένα εργαλείο σύγκρισης αρχείων. Αυτές αποτέλεσαν και τις μετρικές καταγραφής της αστάθειας λογισμικού. Απέδειξαν ότι σχεδόν σε όλες τις εκδόσεις των τριών περιπτώσεων που εξετάστηκαν, οι κλάσεις που είχαν εξαρτήσεις με αντι-προτυπα ήταν πιο επισφαλείς από άλλες. Επίσης, παρατήρησαν ότι οι δομικές αλλαγές είναι αυτές που επηρεάζουν περισσότερο κλάσεις με εξαρτήσεις αντι-προτυπων. Οι προγραμματιστές θα μπορούσαν να χρησιμοποιήσουν αυτές τις γνώσεις για να εστιάσουν πιο στοχευμένη τον έλεγχο τους προς τις πιο επιρρεπείς κλάσεις .

Πολλές μετρικές έχουν αποδειχθεί αξιόπιστες στην πρόβλεψη εξωτερικών χαρακτηριστικών, όπως ο συσχετισμός με σφάλματα. Η επόμενη δημοσίευση παρέχει μια εναλλακτική εξήγηση, όσον αφορά στην αμφιλεγόμενη επίδραση του μεγέθους. Σε αντίθεση με την ευρέως αποδεκτή άποψη, οι Gil, Y., & Lalouche, G. (Gil & Lalouche, 2017) αμφισβήτησαν ότι η εγκυρότητα μιας μετρικής μπορεί να εξηγηθεί από τη συσχέτιση της με το μέγεθος του πηγαίου κώδικα. Το κύριο συμπέρασμα τους ήταν ότι, με τους κατάλληλους μετασχηματισμούς, η εγκυρότητα μιας μετρικής είναι εφικτό να προβλεφθεί με ακρίβεια από τη συσχέτιση της με το μέγεθος. Τα αποτελέσματα έχουν εξαχθεί με τη χρήση 26 μετρικών, συμπεριλαμβανομένων των δημοφιλών μετρικών των Chidamber and Kemerer. Πιο συγκεκριμένα, έδειξαν ότι όσο περισσότερο μια μετρική συσχετίζεται με το μέγεθος τόσο ικανότερη είναι στην πρόβλεψη εξωτερικών χαρακτηριστικών, και αντιστρόφως. Ορίσαν δυο ξεχωριστές μεθόδους για να ελέγξουν το μέγεθος, με γραμμικό μετασχηματισμό (linear transformation). Από το σύνολο των μετρικών που εξετάστηκαν βρέθηκε ότι οι μετρικές που επηρεάζουν την αστάθεια του λογισμικού είναι οι εξής:

- LOC (Lines of Code)
- NOT (Number of Tokens)
- NOS (Number of Statements)
- MCCS (McCabe Cyclomatic Complexity (simple))

Οι τρεις πρωτες ανήκουν στην ομάδα των μετρικών μεγέθους ενώ η τελευταία στην ομάδα των μετρικών της πολυπλοκότητας ροής (Flow

complexity). Με βάση τα αποτελέσματα των ερευνητών, οι διάσημες μετρικές των Chidamber and Kemerer δεν είναι καλύτερες από τις υπόλοιπες μετρικές της σουίτας που επέλεξαν.

Από πολύ νωρίς, η μελέτη της αρχιτεκτονικής λογισμικού έχει αναγνωρίσει την αρχιτεκτονική φθορά ως φαινόμενο που παρατηρείται τακτικά σε μακρόβια συστήματα. Η αρχιτεκτονική αποσύνθεση προκαλείται από επαναλαμβανόμενες, μερικές φορές απρόσεκτες, αλλαγές στη διάρκεια ζωής ενός συστήματος. Πάρα το γεγονός αυτό, υπάρχει μια σχετική έλλειψη εμπειρικών δεδομένων όσον αφορά στη φύση των αρχιτεκτονικών αλλαγών που οδηγούν σε αυτή τη φθορά/αποσύνθεση. Στην επόμενη εργασία, οι Behnamghader, P., Le, D. M., Garcia, J., Link, D., Shahbazian, A., & Medvidovic, N. (Behnamghader, et al., 2017) θέλησαν να εξερευνήσουν το παραπάνω ζήτημα εισάγοντας ένα νέο πλαίσιο ανάκτησης αρχιτεκτονικής, το ARCADE, με στόχο την εμπειρική μελέτη της αρχιτεκτονικής σε ευρεία κλίμακα και διάμεσου διαφορετικών εκδόσεων ενός συστήματος λογισμικού. Το ARCADE εισάγει καινοτόμες μετρικές αλλαγής αρχιτεκτονικής που αποτελούν τα κλειδιά αυτής της μελέτης. Για την μέτρηση των αλλαγών σε ολόκληρη την ιστορία ενός συστήματος λογισμικού εισήχθησαν δυο νέες μετρικές: η c και η a2a. Η c είναι μια μετρική που υπολογίζει την ομοιότητα μεταξύ δύο αρχιτεκτονικών με βάση τα συστατικά μέρη του κάθε αρχιτεκτονική. Η a2a είναι μια μετρική ομοιότητας σε επίπεδο συστήματος της οποίας ο υπολογισμός βασίζεται στο κόστος μετατροπής από μια αρχιτεκτονική σε μια άλλη. Προκειμένου να υπολογιστεί το ελάχιστο κόστος μετασχηματισμού μεταξύ της αρχιτεκτονικής δυο διαφορετικών εκδόσεων του ιδίου συστήματος, εισήχθη μια νέα μετρικής αρχιτεκτονικής απόστασης, η to, ενώ παρουσιάστηκε και ο αλγόριθμος υπολογισμού της. Χρησιμοποίησαν το ARCADE για να διεξάγουν εμπειρική μελέτη των αλλαγών που εντόπισαν στην αρχιτεκτονική μερικών εκατοντάδων εκδόσεων 23 διαφορετικών συστημάτων ανοιχτού λογισμικού. Η μελέτη αποκαλύπτει αρκετά νέα ευρήματα σχετικά με τη συχνότητα των αρχιτεκτονικών αλλαγών στα συστήματα λογισμικού, τα κοινά σημεία εκκίνησης στην αρχιτεκτονική ενός συστήματος κατά τη διάρκεια της συντήρησης και εξέλιξης του συστήματος, τη διαφορά μεταξύ της αρχιτεκτονικής αλλαγής σε επίπεδο συστήματος και επίπεδο components.

Τα συστήματα λογισμικού υφίστανται συνεχείς αλλαγές, προκαλώντας εκφυλισμό της αρχιτεκτονικής του συστήματος στο πέρασμα του χρόνου. Η αντιστροφή αυτού του εκφυλισμού απαιτεί επιπλέον προσπάθεια και καθυστερεί την έκδοση της επόμενης έκδοσης. Η βελτιωμένη αρχιτεκτονική

είναι μια έννοια αόριστη καθώς δεν μπορεί να μεταφραστεί σε ορατά χαρακτηριστικά. Εξαιτίας της έλλειψης αντιπροσωπευτικών μετρικών οι προγραμματιστές υποστηρίζουν ότι η διακοπή του εκφυλισμού της αρχιτεκτονικής είναι όντως απαραίτητη κι ότι η προσπάθεια για βελτίωση της όντως θα αποδώσει. Το 2003, οι Lindvall, M., Tvedt, R. T., & Costa, P. (Mikael Lindvall, 2003) θεωρώντας ότι οι μετρικές αρχιτεκτονικής θα μπορούσαν να παρέχουν βελτιωμένα εργαλεία, ορίσαν μια σουίτα μετρικών που περιγράφουν μια διαδικασία ανάλυσης της αρχιτεκτονικής. Η δημοσίευση μελετά την περίπτωση ενός project οπού αναδιαρθρώνεται η αρχιτεκτονικής ενός υπάρχοντος συστήματος τύπου server-client γραμμένο σε Java ενώ προστίθεται μια νέα λειτουργικότητα. Η υπάρχουσα έκδοση του συστήματος είχε μια ανοργάνωτη δομή επικοινωνίας και βασιζόταν κυρίως σε βιβλιοθήκες. Η νέα αρχιτεκτονική βασίζεται σε components. Στόχος είναι να εκτιμηθεί η νέα αρχιτεκτονική από την οπτική της συντήρησης λογισμικού και κατ' επέκταση της αστάθειας. Το άρθρο περιγράφει τη διαδικασία αξιολόγησης, τις μετρικές που χρησιμοποιούνται και παρέχει ορισμένα προκαταρκτικά αποτελέσματα. Για να καταγράψουν τη σύζευξη σε επίπεδο δομοστοιχείων (Module), ορίσαν μια μετρική που ονομάζεται 'σύζευξη μεταξύ modules' (coupling-between-modules(CBM)). CBM είναι ο αριθμός των διακριτών αναφορών μεταξύ των δομοστοιχειων. Η εισαγωγή κλάσης από ένα module, η κληρονομικότητα μιας κλάσης σε module και η δήλωση ή δημιουργία ενός αντικειμένου από ένα Module περιλαμβάνονται ως μορφές σύζευξης στην CMB. Ορίσαν την CBM(m) ως το πλήθος των Modules που συνδέονται με το module m. Προκειμένου να καταγραφεί ο βαθμός σύζευξης μεταξύ των module σε καλύτερο επίπεδο ορίσαν την μετρική 'σύζευξη μεταξύ κλάσεις module'(coupling between module classes(CBMC(m))). Χρησιμοποιείται για να καταγράψει το πλήθος των κλάσεων που περιλαμβάνονται στη σύζευξη των module. Επιπλέον, ορίσαν μια μετρική σύζευξης μέσα σε ένα module, την CIM. Για να υπολογιστεί αυτή για ένα module, πρέπει να υπολογιστεί για καθεμιά κλάση του module ξεχωριστά. Η τελική αξιολόγηση της αρχιτεκτονικής έδειξε ότι τα components ήταν χαλαρά συνδεδεμένα μεταξύ τους. Τα Module της νέας αρχιτεκτονικής ήταν components, που χαρτογραφούνται ευκολά σε σχέση με τις απαιτήσεις του συστήματος. Η διαδικασία αξιολόγησης είναι εύκολο να εφαρμοστεί και σε άλλες περιπτώσεις λόγω της γενικότητας της.

Σημαντικό κίνητρο για τον αντικεινοστραφή προγραμματισμό είναι η βελτίωση της μεταβλητότητας του λογισμικού και κατ' επέκταση του κόστους ανάπτυξης στη διάρκεια της ζωής του. Στην επόμενη δημοσίευση, οι Arisholm, E., & Sjøberg, D. I. K. (Arisholm & Sjøberg, Towards a framework

for empirical assessment of changeability decay, 2000) περιέγραψαν τα αποτελέσματα ορισμένων πειραμάτων που εκτίμησαν την αστάθεια συγκρίνοντας δυο διαφορετικούς σχεδιασμούς, ο ένας ήταν προσανατολισμένος σε μια πιο εθνοκεντρική κατεύθυνση (responsibility driven(RD)) , ο άλλος κινούταν περισσότερο στα κοινότυπα πλαίσια του ελέγχου (controlled oriented mainframe(MF)). Σύμφωνα με τις αρχές ποιοτικού σχεδιασμού των Coad και Yourdon, ο πρώτος είναι ο 'καλός' σχεδιασμός ενώ ο δεύτερος ο 'κακός'. Για να ερευνήσουν ποιος έχει λιγότερη αστάθεια διεξήγαγαν δυο ελεγχόμενα πειράματα, ένα πιλοτικό και ένα κύριο πείραμα. Στο πιλοτικό αξιολογούν το σχεδιασμό του πειράματος και διατυπώνουν μια υπόθεση ενώ στο κύριο αντιγράφουν το πιλοτικό με διαφορετικά άτομα και σε μεγαλύτερο εύρος. Και στα δυο, τα άτομα χωρίστηκαν σε δυο ομάδες οπού σχεδίαζαν, προγραμμάτιζαν και ήλεγχαν διάφορες πανομοιότυπες αλλαγές σε ένα από τα δυο πρότυπα σχεδιασμού. Οι μετρικές που αφορούσαν τον πηγαίο κώδικα κατά τη φάση της υλοποίησης ήταν οι εξής:

- Η διαφορά στο μέσο μέγεθος κλάσης πριν και μετα από μια αλλαγή
- Η διαφορά στο μέσο πλήθος μεθόδων σε μια κλάση
- Η διαφορά στο μέσο πλήθος κλήσεων στατικών μεθόδων για μια κλάση
- Η διαφορά στο μέσο πλήθος κλήσεων στατικών μεθόδων σε library και non library κλάσεις
- Γραμμές κώδικα που προστέθηκαν/ διεγράφησαν/ τροποποιήθηκαν για κάθε αλλαγή

Τα αποτελέσματα έδειξαν ότι ο σχεδιασμός RD απαιτεί μικρότερο κόστος αλλαγών. Αυτή η διαφορά στο κόστος/ προσπάθεια οφείλεται στην προσπάθεια κατανόησης του τρόπου υλοποίησης των αλλαγών. Τέλος, τα δομικά χαρακτηριστικά αλλάζουν λιγότερο και πάλι για το RD.

2.2 Σύνοψη

Ο πίνακας των μετρικών που μπορούν να καθορίσουν άμεσα ή έμμεσα την αστάθεια λογισμικού ακολουθεί:

Πίνακας 1

Ονόματα μετρικών	Περιγραφή
the difference in average number of static method invocations for a class c to non-library classes	
a metric that computes the minimum transforming cost between an architecture	

of two different versions of the same software system (mto)	
a system-level similarity metric calculated based on the cost of transforming one architecture to another (a2a)	
ACE	Καταγράφει τόσο τις δομικές όσο και τις εννοιολογικές σχέσεις μεταξύ των στοιχείων λογισμικού.
Average Age	υπολογίζει πόσο καιρό μια κλωνοποιημένη ή μη κλωνοποιημένη γραμμή κώδικα παραμένει αμετάβλητη κατά μέσο όρο
Average Last ChangeDate	πόσο πρόσφατα τροποποιείται μια περιοχή κώδικα (κλωνοποιημένη ή μη κλωνοποιημένη). σύμφωνα με αυτήν τη μετρική, μια περιοχή κώδικα που άλλαξε πιο πρόσφατα θεωρείται πιο ασταθής
Change Dispersion	ο βαθμός στον οποίο ο μη κλωνοποιημένος κώδικας αλλάζει στις περιοχές κλώνου ή μη κλώνου είναι διασκορπισμένοι στην αντίστοιχη περιοχή
Change Dispersion Impact	Η επίδραση της διασποράς των αλλαγών ποσοτικοποιεί το βαθμό στον οποίο οι αλλαγές στις περιοχές κλώνου ή μη κλώνου είναι διάσπαρτες στην αντίστοιχη περιοχή
Change Metric (CM)	η αλλαγή σε μια μετρική από τη μία κυκλοφορία(release) στην άλλη
ChangeSize	
CHM	
class growth	δείχνει την εξέλιξη του σχεδιασμού του λογισμικού
Class Implementation Instability (CII)	Μετρά τις αλλαγές σχεδιασμού σε επίπεδο κλάσης, π.χ. από το Design N στο Design N+1 όσον αφορά στις αλλαγές στο πλήθος των γραμμών κώδικα (LOC) μέσα σε μια κλάση
COH	
COM	
COMI	
Composition Volatility (CoV)	Οι χαμηλότερες τιμές για αυτήν τη μετρική είναι καλύτερες επειδή αυτό σημαίνει ότι απαιτείται λιγότερος κώδικας για την υλοποίηση που μπορεί να συνεπάγεται λιγότερη τροποποίηση
CUB	
CUBF	
DEPCC	
Depth of Dependency Chain (DDC)	modules που επηρεάζονται άμεσα και έμμεσα
DIT	μέγιστο μήκος της διαδρομής από μια κλάση έως τη ρίζα του ιεραρχικού δέντρου

External Stability (ES)	ο βαθμός που τα αρχιτεκτονικά στοιχεία παραμένουν σταθερά σε διαδοχικές εκδόσεις
Global Impact (GoI)	ο συνολικός αριθμός των στοιχείων (elements)του προγράμματος που εμπλέκονται
Hybrid Feature Coupling(HFC)	συνδυάζει τις δομικές και τις πληροφορίες κειμένου σε μια μετρική σύζευξης
I	Το set των API σε ένα module
I(A)	η μετρική αστάθειας του πακέτου (A)
Impact	μακροπρόθεσμος αντίκτυπος των αλλαγών εντοπίζοντας εξαρτήσεις μεταξύ αλλαγών πηγαίου κώδικα
impact factor	η προτεραιότητα των προς επιθεώρηση μεθόδων
Internal Stability (IS)	ο βαθμός που οι σχέσεις παραμένουν σταθερές σε διαδοχικές εκδόσεις
LCOMB	ο αριθμός των διαχωριστικών συνόλων τοπικών μεθόδων
Likelihood	Η πιθανότητα αλλαγής μιας μεθόδου m κατά τη διάρκεια της κλωνοποιημένης περιόδου (ή μη κλωνοποιημένης περιόδου) είναι ο λόγος μεταξύ του αριθμού των αλλαγών σε m και του συνολικού αριθμού αλλαγών στο σύστημα (όλες οι μέθοδοι) κατά τη διάρκεια της κλωνοποιημένη περιόδου (ή της μη κλωνοποιημένης περιόδου) .
lines of code added	
Lines of code changed	
lines of code deleted	Όπως περιγράφεται στο *diff*, το οποίο είναι ένα εργαλείο για τη σύγκριση αρχείων και τη δημιουργία μιας λίστας διαφορών.
LOC	γραμμές κώδικα
Local Impact (LoI)	Έστω ένα πρόγραμμα P. για κάθε module m Î Μη LoI. ο αντίκτυπος του m μπορεί να οριστεί LoIP, m = | DDm | / | EM |. Ως αποτέλεσμα, το LoI ενός προγράμματος P μπορεί να οριστεί ως LoIP, Me = S mÎ Me LoIP, m.
Maximum Textual Feature Coupling (TFCmax)	η σχέση μεταξύ δύο χαρακτηριστικών βασισμένη σε δομικές πληροφορίες
MCC	
McCabe Cyclomatic Complexity (MCC)	μια παραλλαγή του MCC που αποκλείει περιπτώσεις short-circuit evaluation, καθώς αυτές μπορεί να αποδοθούν στη σημασιολογία της γλώσσας και όχι στην πραγματική πολυπλοκότητα
MDS	
metric that computes the similarity between two architectures based on the constituent components of each architecture (cvg)	

minutes	Λεπτά συνολικής προσπάθειας
Modification Frequency	Η μετρικη του τρόπου με τον οποίο τροποποιείται μια περιοχή κώδικα (κλωνοποιημένος ή μη). Επικεντρώνεται στον αριθμό των αλλαγών αγνοώντας την ποσότητα των γραμμών που επηρεάζονται από μια αλλαγή.
Modification Probability	Αυτή η μετρικη ορίστηκε αρχικά ως συνολική αστάθεια από τους Gode και Harder (2011)
NAC	Αριθμός προστιθέμενων κλασεων
NALOC	Αριθμός προστιθέμενων γραμμών κώδικα (LOC)
NCC	Αριθμός τροποποιημενων κλασεων
NCLOC	Ο αριθμός των γραμμών κώδικα που τροποποιήθηκαν για να ικανοποιήσουν ένα αίτημα αλλαγής
NOC	αριθμός άμεσων υποκατηγοριών μιας κλάσης
NOCON	
NOM	Αριθμός μεθόδων
NON	
NOP	Αριθμός γονέων. Ο αριθμός κλάσεων από τις οποίες κληρονομείται άμεσα μια τάξη.
NPA	
NPPM	
NSFC	ο αριθμός των source files που τροποποιήθηκαν για να ικανοποιήσουν ένα αίτημα αλλαγής
Number of modules coupled with another module(CBM)	
number of modules or operations removed	
number of modules or operations added	
number of modules or operations changed	
Number of non-directional, distinct, inter-module, class-to-class references for another module (CBMC)	
Number of Tokens (NOT)	
OCMAIC	
OMAEC	
REM	Μετρικη επίδρασης ripple effect (αξιολογητής της πιθανότητας μιας τάξης να αλλάξει λόγω του φαινομένου των αλυσιδωτών αντιδράσεων)
Reuse Oriented Stability (ROS)	η σταθερότητα ενός συστήματος λογισμικού όσον αφορά την ανθεκτικότητα στην εισαγωγή νέων σφαλμάτων κατά την εξέλιξή του
SDI	Η μετρικη System Design Instability (SDI) υποδεικνύει την πρόοδο ενός

	αντικειμενοστρεφούς έργου (OO) όταν το έργο ξεκινήσει να μεταλλάσσεται.
Structural Feature Coupling (SFC)	
Structural Feature Coupling that includes the static callers and callees of a feature's methods (SFC')	
System Implementation Instability	οι αλλαγές σε ένα αντικειμενοστρεφές σύστημα , π.χ. η διαφορά του LOC από το Design N στο Design N 1.1
System Stability	100% − (Cumulative Component Dependency / Module Count^2) * 100
Textual Feature Coupling (TFC)	η σύζευξη μεταξύ των χαρακτηριστικών που βασίζονται σε μη δομημένες, πληροφορίες κειμένου στον πηγαίο κώδικα χρησιμοποιώντας μια τεχνική ανάκτησης πληροφοριών
the average of the number of modules coupled with another module, for the classes within the module (CIM)	
the difference in average class size before and after a change class	
the difference in average number of implemented methods in a class c	
the difference in average number of static method invocations for a class c to library classes	
the difference in average number of static method invocations for a class c to non-library classes	
total change impact	
WMC	Σύνοψη της πολυπλοκότητας κάθε μεθόδου σε μια κλάση. Εάν εξετάσουμε την πολυπλοκότητα κάθε μεθόδου ενιαία το WMC είναι ίσο με το NOM
WMCLOC	
the difference in average number of static method invocations for a class c to non-library classes	
a metric that computes the minimum transforming cost between an architecture of two different versions of the same software system (mto)	

3. Μεταβλητότητα

3.1 Εισαγωγή

Η μεταβλητότητα (change proneness) ορίζεται συνήθως ως πιθανότητα αλλαγής μιας κλάσης ή μεθόδου, λόγω εσωτερικών λόγων (π.χ., επιδιόρθωση σφαλμάτων ή αλλαγή των απαιτήσεων). Στην ερευνά μας με το ζήτημα ασχολήθηκαν 28 από τα άρθρα/ δημοσιεύσεις που εξετάστηκαν.

Σε ένα αντικειμενοστρεφές σύστημα , η μεταβλητότητα είναι ένα σημαντικό εξωτερικό χαρακτηριστικό ποιότητας που υποδηλώνει την έκταση της αλλαγής μιας κλάσης στις διάφορες εκδόσεις του συστήματος ((Koru & Tian, 2005) (Arisholm, Briand, & Foyen, 2004). Δε βοηθά τους προγραμματιστές λογισμικού μόνο να λάβουν εστιασμένες προληπτικές ενέργειες για τη μείωση του κόστους συντήρησης και τη βελτίωση της ποιότητας, αλλά μπορεί επίσης να βοηθήσει τους διαχειριστές λογισμικού να κατανέμουν πόρους πιο αποτελεσματικά. Τις δύο τελευταίες δεκαετίες, έχουν γίνει πολλές προσπάθειες για τη διερεύνηση της ικανότητας των μετρικών να προβλέπουν επιρρεπείς σε αλλαγές κλάσεις (Koru & Tian, 2005), (Arisholm, Briand, & Foyen, 2004).

3.2 Συλλογή άρθρων

Διάφοροι ερευνητές έχουν δημιουργήσει επιτυχώς τη σύνδεση μεταξύ αντικειμενοστρεφών μετρικών και της μεταβλητότητας μιας κλάσης. Ωστόσο συνεχίζουν ενεργά να διερευνούν αποτελεσματικούς κατηγοριοποίησες (classifiers) για να αναπτυχθούν πιο αποτελεσματικά μοντέλα πρόβλεψης. Οι πρόσφατες εξελίξεις έχουν επιβεβαιώσει ότι υπάρχει δυνατότητα να βελτιωθεί η απόδοση των κατηγοριοποιητών. Οι Malhotra, R., & Khanna, M. (Malhotra & Khanna, Particle swarm optimization-based ensemble learning for software change prediction, 2018), προτείναν τέσσερις στρατηγικές συνολικής μάθησης ώστε να προβλέπεται η μεταβλητότητα των κλάσεων συνδυάζοντας επτά διαφορετικούς κατηγοριοποίητέςς βασισμένους στον αλγόριθμο Βελτιστοποίησης Σμήνους Σωματιδίων (Particle Swarm Optimization - PSO) χρησιμοποιώντας σταθμισμένη ψηφοφορία (weighted voting). Τα βάρη που κατανέμονται σε μεμονωμένους κατηγοριοποίητές βασίζονται στην ακρίβεια και την ικανότητά τους να προβλέψουν σωστά τις "δύσκολες περιπτώσεις", όπως τις κλάσεις που σπάνια κατηγοριοποιούνται σωστά. Κάθε κατηγοριοποιητής χρησιμοποιεί διαφορετική μαθηματική βελτιστοποίηση. Το σημαντικότερο είναι να συνδυαστούν τα αποτελέσματα ώστε να προκύψει ένας καλύτερος κατηγοριοποιητης με μεγαλύτερη ακρίβεια στην πρόβλεψη. Συνεπώς, η μεταβλητότητα μετριέται μέσω άλλων μετρικών στη φάση της υλοποίησης και μέσω του πηγαίου κώδικα. Οι μετρικές που εξετάστηκαν ανήκουν στη γνωστή σουίτα των Chidamber και Kemerer (CK) και είναι οι εξής:

- Depth of Inheritance Tree (DIT)
- Number of Children (NOC) Coupling

- Between Objects (CBO) metric
- Response For a Class (RFC)
- Lack of Cohesion amongst Methods of a Class (LCOM)
- Source Lines Of Code (SLOC)
- Accuracy
- Precision
- G-Mean
- Product
- F-measure
- G-measure
- Balance

Ο υπολογισμός των παραπάνω μετρικών έγινε χρησιμοποιώντας ένα εργαλείο ανοιχτού λογισμικού, το CKJM, ενώ τα στατιστικά που αφορούν τις αλλαγές όπως οι γραμμές κώδικα που προστέθηκαν, αλλάχθηκαν ή διαγράφησαν υπολογίστηκαν αναλύοντας τα αρχεία καταγραφής του GIT. Η μελέτη χρησιμοποίησε έξι πακέτα εφαρμογών του Android (Calendar, Contacts, Gallery, Bluetooth, MMS and Telephony) αλλά και από το δημοφιλές λογισμικό Apache (Apache Commons IO, Apache Commons Math, Apache Log4j and Apache Net). Οι προτεινόμενες στρατηγικές αποδείχτηκαν αποτελεσματικές. Η στατιστική ανάλυση των αποτελεσμάτων δείχνει βελτιωμένη απόδοση των συνδυασμένων κατηγοριοποιητών σε σχέση με τους μεμονωμένους.

Η πρόβλεψη από κοινού αλλαγής κάνει τους προγραμματιστές να γνωρίζουν ποια αντικείμενα θα αλλάξουν μαζί με το αντικείμενο που εργάζονται. Στο παρελθόν, οι ερευνητές βασίστηκαν στη δομική ανάλυση για να δημιουργήσουν μοντέλα προβλέψεων. Πιο πρόσφατα, έχουν προταθεί υβριδικές προσεγγίσεις που βασίζονται σε ιστορικές πληροφορίες και ανάλυση κειμένου. Παρά τις εξελίξεις στον τομέα, οι προγραμματιστές εξακολουθούν να μην χρησιμοποιούν αυτές τις προσεγγίσεις ευρέως, πιθανώς λόγω του αριθμού των λανθασμένων προτάσεων. Το 2016 δημοσιεύτηκε η επόμενη εργασία μας από τους Wiese, I. S., Ré, R., Steinmacher, I., Kuroda, R. T., Oliva, G. A., Treude, C., & Gerosa, M. A. (Wiese, et al., 2017). Υποθέτουν ότι οι πληροφορίες των αλλαγών λογισμικού που συλλέγονται από ανοιχτά issues, η επικοινωνία των προγραμματιστών και τα μεταδεδομένα καταγράφουν τα μοτίβα αλλαγών των τεχνουργημάτων λογισμικού και μπορούν να βελτιώσουν τα μοντέλα πρόβλεψης. Στόχος τους είναι να αναπτύξουν πιο ακριβή μοντέλα πρόβλεψης συναλλαγής χρησιμοποιώντας πληροφορίες των

συμφραζόμενων από αλλαγές λογισμικού. Επιλέξαν ζεύγη αρχείων με βάση τους σχετικούς κανόνες συσχέτισης και δημιουργήσαμε ένα μοντέλο πρόβλεψης για κάθε ζεύγος βασισμένο στις σχετικές πληροφορίες του περιβάλλοντος. Αξιολογήσαν την προσέγγισή τους σε δύο έργα ανοιχτού λογισμικού, και πιο συγκεκριμένα το Apache CXF και το Derby. Εκτός από τον υπολογισμό των μετρήσεων ακρίβειας του μοντέλου, πραγματοποιήσαν επίσης μια ανάλυση επιλογής χαρακτηριστικών για να εντοπίσουν τους καλύτερους προγνωστικούς παράγοντες όταν χαρακτηρίζουμε τις συν-αλλαγές. Οι μετρικές που χρησιμοποίησαν στην παραπάνω ανάλυση ήταν το πλήθος των γραμμών του πηγαίου κώδικα που αλλάχθηκαν, το πλήθος των λέξεων που περιέγραφαν ένα issue, το πλήθος των issues, η επικοινωνία μεταξύ προγραμματιστών και τα commits του κώδικα. Τα μοντέλα τους παρουσίασαν χαμηλά ποσοστά ψευδών αρνητικών (8% μέσο όρο) και ψευδών θετικών (~11% μέσο ορό). Οι μετρήσεις που σχετίζονται με τα commits ήταν οι πιο συχνά επιλεγμένες και για τα δύο έργα. Κατά μέσο όρο, 6 από τις 23 μετρήσεις ήταν απαραίτητες για τη δημιουργία των classifiers. Τα μοντέλα πρόβλεψης που βασίζονται σε πληροφορίες συμφραζόμενων από αλλαγές λογισμικού είναι ακριβή και, κατά συνέπεια, μπορούν να χρησιμοποιηθούν για την υποστήριξη της συντήρησης και της εξέλιξης του λογισμικού, προειδοποιώντας τους προγραμματιστές όταν χάνουν σχετικά αντικείμενα κατά την εκτέλεση μιας αλλαγής λογισμικού.

Η ανάπτυξη και η συντήρηση λογισμικού ανοιχτού κώδικα έχει γίνει σημαντική πηγή κέρδους για πολλές εταιρείες. Κλάσεις με υψηλή μεταβλητότητα αυξάνουν το κόστος έργου απαιτώντας από τους προγραμματιστές να ξοδεύουν προσπάθεια και χρόνο. Ο εντοπισμός και ο χαρακτηρισμός τάξεων με τάση αλλαγής μπορούν να επιτρέψουν στους προγραμματιστές να εστιάζουν έγκαιρες προληπτικές ενέργειες, για παράδειγμα, αξιολογήσεις , σε κλάσεις με παρόμοια χαρακτηριστικά στις μελλοντικές εκδόσεις. Σε αυτή τη δημοσίευση οι Koru, A. & Liu, Hongfang. (Koru & Liu, 2007) συνέλλεξαν ένα σύνολο στατικών μετρήσεων και αλλάξανε δεδομένα σε επίπεδο κλάσης από δύο έργα ανοιχτού κώδικα, το KOffice και το Mozilla. Χρησιμοποιώντας αυτά τα δεδομένα, δοκιμάσανε πρώτα και επιβεβαίωσαν τον Νόμο του Pareto, ο οποίος λέει ότι η μεγάλη πλειοψηφία (περίπου 80%) της αλλαγής βασίζεται σε ένα μικρό ποσοστό (περίπου 20%) κλάσεων. Στη συνέχεια, προσδιόρισαν και χαρακτήρισαν τις κλάσεις που είναι επιρρεπείς σε αλλαγές στα δύο προϊόντα με την παραγωγή tree-based μοντέλων. Επιπλέον, χρησιμοποιώντας μοντέλα που

βασίζονται σε δέντρα, προτείνουν μια στρατηγική ιεράρχησης για τη χρήση πόρων του έργου με στόχο εστιασμένες προληπτικές ενέργειες με αποτελεσματικό τρόπο. Τα εμπειρικά αποτελέσματα έδειξαν ότι αυτή η στρατηγική ήταν αποτελεσματική. Αυτή η μελέτη θα πρέπει να παρέχει χρήσιμη καθοδήγηση σε επαγγελματίες που συμμετέχουν στην ανάπτυξη και συντήρηση προϊόντων μεγάλης κλίμακας ανοιχτού κώδικα.

Οι συνεχείς αλλαγές που εφαρμόζονται κατά τη διάρκεια της συντήρησης λογισμικού κινδυνεύουν να επιδεινώσουν τη δομή ενός συστήματος και αποτελούν απειλή για τη συντήρηση του. Σε αυτό το πλαίσιο, η πρόβλεψη των τμημάτων του πηγαίου κώδικα όπου πρέπει να επικεντρωθούν συγκεκριμένες εργασίες συντήρησης μπορεί να είναι ζωτικής σημασίας για τους προγραμματιστές για την αποτροπή ζητημάτων συντήρησης. Οι Catolino, G., Palomba, F., De Lucia, A., Ferrucci, F., & Zaidman, A. (Catolino, Palomba, Lucia, Ferrucci, & Zaidman, 2018) πιστεύουν ότι οι υπάρχουσες προσεγγίσεις εξακολουθούν να χάνουν ένα σημαντικό κομμάτι πληροφοριών, δηλαδή παράγοντες που σχετίζονται με τους προγραμματιστές και οι οποίοι μπορούν να συλλάβουν την πολυπλοκότητα της διαδικασίας ανάπτυξης υπό διαφορετικές προοπτικές. Σε αυτό το άρθρο, πρώτα διερευνούν τρία μοντέλα πρόβλεψης αλλαγών που εκμεταλλεύονται παράγοντες που σχετίζονται με προγραμματιστές (π.χ., αριθμός προγραμματιστών που εργάζονται σε μια κλάση) ως προγνωστικοί παράγοντες για την αλλαγή της μεταβλητότητας των κλάσεων και στη συνέχεια τα συγκρίνουν με τα υπάρχοντα μοντέλα. Χρησιμοποίησαν το εργαλείο CHANGEDISTILLER στη φάση της υλοποίησης, το οποίο τρέχει για κάθε ζευγάρι commits. Τρέχει έναν αλγόριθμο ο οποίος εξάγει τις διάφορες στον κώδικα μεταξύ δυο commits. Το πλαίσιο της μελέτης αποτελείται από είκοσι λογισμικό ανοιχτού κώδικα συστήματα με διαφορετικό μέγεθος και πεδίο εφαρμογής. Συνολικά, η μελέτη εξετάζει 408 releases, 192.274 commits και 657 προγραμματιστές. Παρατηρήσαν ενδιαφέρουσες αλληλοσυμπληρώσεις μεταξύ των μοντέλων πρόβλεψης. Για αυτόν τον λόγο, επινοήσαν ένα νέο μοντέλο πρόβλεψης αλλαγών που εκμεταλλεύεται τον συνδυασμό παραγόντων που σχετίζονται με προγραμματιστές και μετρικές εξέλιξης. Τα αποτελέσματα δείχνουν ότι ένα τέτοιο συνδυαστικό μοντέλο είναι έως και 22% πιο αποτελεσματικό από ό,τι τα μεμονωμένα μοντέλα για τον προσδιορισμό μεταβλητότητας των κλάσεων.

Η λογική του σχεδιασμού παίζει σημαντικό ρόλο στο σχεδιασμό και στη συνέχεια συντήρηση μεγάλων και πολύπλοκων συστημάτων. Αυτή η

προσέγγιση δεν μπορεί να χρησιμοποιηθεί για να υποστηρίξει αποτελεσματικά την ανάλυση μετάδοσης αλλαγών, διότι η εξάρτηση μεταξύ στοιχείων σχεδίασης και αποφάσεων δεν αντιπροσωπεύεται καλά και δεν μπορεί να ποσοτικοποιήσεις. Χωρίς τέτοιες γνώσεις, οι σχεδιαστές και οι αρχιτέκτονες λογισμικού δεν μπορούν εύκολα να εκτιμήσουν πώς οι αλλαγές των απαιτήσεων και οι αποφάσεις σχεδιασμού μπορεί να επηρεάσουν το σύστημα. Στο επόμενο άρθρο, Tang, A., Nicholson, A., Jin, Y., & Han, J. (Tang, Nicholson, Jin, & Han, 2007) , παρουσιάζεται το μοντέλο Αρχιτεκτονικής Αιτιολογίας και Σύνδεσης Στοιχείου (Architecture Rationale and Element Linkage, AREL) για να αντιπροσωπευθούν οι αιτιώδεις σχέσεις μεταξύ στοιχείων αρχιτεκτονικής και αποφάσεων. Εφαρμόζουν Bayesian Belief Networks (BBN) στο AREL, για να συλλάβουν τις πιθανολογικές αιτιώδεις σχέσεις μεταξύ σχεδιαστικών στοιχείων και αποφάσεων. Χρησιμοποιούν τρεις διαφορετικές μεθόδους συλλογισμού με βάση το BBN για να αναλύσουν τον αντίκτυπο στην αλλαγή του σχεδιασμού: προγνωστική συλλογιστική, διαγνωστική συλλογιστική και συνδυασμένη συλλογιστική. Αξιολογούν την επίδραση στις μεθόδους συλλογιστικής βάσει πιθανότητας στην ανάλυση μετάδοσης αλλαγών. Τα BBN είναι γραφικά μοντέλα που είναι πλέον ευρέως αποδεκτοί στην κοινότητα AI ως πρακτική. Απεικονίζουν την εφαρμογή της BBN μοντελοποίησης και μεθόδους μετάδοσης αλλαγών με τη μερική σχεδίαση ενός συστήματος επεξεργασίας εικόνας ελέγχου του πραγματικού κόσμου. Για να υποστηρίξουν την εφαρμογή του, έχουν αναπτύξει ένα πρακτικό, ολοκληρωμένο σύνολο εργαλείων για χρήση από τους αρχιτέκτονες.

Αρκετές μελέτες έχουν διερευνήσει τη σχέση μεταξύ των μετρήσεων του αντικειμενοστρεφούς λογισμικού και της μεταβλητότητας των τάξεων. Αυτή η γνώση μπορεί να χρησιμοποιηθεί για να βοηθήσει στη λήψη αποφάσεων μεταξύ εναλλακτικών σχεδιασμών ή για την αξιολόγηση της ποιότητας του λογισμικού. Παρά την αυξανόμενη χρήση σύνθετων σχέσεων κληρονομιάς και πολυμορφισμού στον αντικειμενοστρεφή προγραμματισμό, έχει δοθεί λιγότερη έμφαση στην ανάπτυξη μετρικών που αποτυπώνουν την πτυχή της δυναμικής συμπεριφοράς. Η εξέταση των δυναμικών μετρήσεων συμπεριφοράς σε συνδυασμό με τις υπάρχουσες μετρήσεις μπορεί να συμβάλει σημαντικά στη λήψη πιο ακριβών προβλέψεων σχετικά με την μεταβλητότητα. Σε αυτή τη δημοσίευση, Han, A.-R., Jeon, S.-U., Bae, D.-H., & Hong, J.-E. (Han, Jeon, Bae, & Hong, 2010) , χρησιμοποιούν συμπεριφορικές πληροφορίες που λαμβάνονται από μοντέλα σχεδίασης UML 2.0. Η πρόβλεψη αλλαγής μεταβλητότητας βάσει μοντέλου συμβάλλει στην παραγωγή λογισμικού υψηλής ποιότητας, μέσω της εκμεταλλευσης

σχεδιαστικών μοντέλων από προγενέστερη φάση της διαδικασίας ανάπτυξης λογισμικού. Έτσι, έχουν αναπτύξει το Μέτρο Εξάρτησης Συμπεριφοράς (Behavioral Dependency Measure (BDM)). Αυτή η μέτρηση μπορεί να προέρχεται όχι μόνο από τις δομικές πληροφορίες, αλλά και από τις συμπεριφορικές πληροφορίες των μοντέλων σχεδιασμού UML 2.0. Τα σχεδιαστικά μοντέλα στο UML 2.0 διαθέτουν και τις δύο πληροφορίες του λογισμικού. Τα σχεδιαστικά μοντέλα στο UML 2.0 (OMG, 2007) διαθέτουν και τις δύο πληροφορίες του λογισμικού. Ένα διάγραμμα κλάσης παρέχει τις δομικές πληροφορίες των τάξεων και τις σχέσεις μεταξύ αυτών των κλάσεων. Το BDM έχει αξιολογηθεί σε ένα έργο ανοιχτού κώδικα μεσαίου μεγέθους πολλαπλών εκδόσεων που ονομάζεται Jflex. Τα αποτελέσματα που προέκυψαν δείχνουν ότι η προτεινομένη μετρική είναι ένας χρήσιμος δείκτης και μπορεί να συμπληρώνει τις υπάρχουσες αντικειμενοστρεφείς μετρικές για τη βελτίωση της ακρίβειας της πρόβλεψης μεταβλητότητας όταν το σύστημα περιέχει υψηλό βαθμό σχέσεων κληρονομιάς και πολυμορφικότητα.

Οι ασταθείς μονάδες λογισμικού, στα πλαίσια της επομένης εργασίας, ορίζονται ως εκείνες που είναι πολύ πιο επιρρεπείς σε αλλαγές από άλλες μονάδες στο ίδιο σύστημα ή υποσύστημα. Η δημοσίευση ανήκει στους Braunschweig, B., Dhage, N., Viera, M. J., Seaman, C., Sampath, S., & Koru, G. A. (Braunschweig, et al., 2012) . Υπάρχει πλούσια βιβλιογραφία για τη διερεύνηση μοντέλων για την πρόβλεψη των ενοτήτων σε ένα σύστημα που θα γίνουν ασταθή ή / και είναι επιρρεπή σε ελαττώματα. Μεγάλο μέρος αυτής της βιβλιογραφίας επικεντρώνεται στη χρήση μετρικών πηγαίου κώδικα (π.χ. μετρικές πολυπλοκότητας) και απλές μετρικές αλλαγών (π.χ. αριθμός προγενέστερων αλλαγών) ως input στα προγνωστικά μοντέλα. Η δουλειά των μελετητών προσπαθεί να διευρύνει το εύρος παραγόντων που εξετάζονται σε τέτοιες προσεγγίσεις προβλέψεων. Για το σκοπό αυτό, συνέλεξαν δεδομένα απευθείας από τους προγραμματιστές, σχετικά με τους παράγοντες στους οποίους βασίζονται για να προβλέψουν ποια μέρη ενός συστήματος πρόκειται να γίνουν ασταθή. Περιγράφουν μια μελέτη ομάδας εστίασης (focus group study) που διεξήχθη με την ομάδα ανάπτυξης ενός μικρού αλλά ενεργού έργου ανοιχτού κώδικα, στο οποίο έθεσαν την ίδια ερώτηση. Οι ομάδες εστίασης είναι μια ποιοτική ερευνητική μέθοδος κατάλληλη για να εντοπίζονται οι ιδέες και τα συναισθήματα των συμμετεχόντων, παράγοντες που ευθύνονται για αυτά. Μια ομάδα εστίασης δεν επιδιώκει συνοχή. Αντιθέτως από μια ατομική συνέντευξη, που είναι ένας διάλογος μεταξύ του ερευνητή και ένας πληροφοριοδότη, η έμφαση σε μια ομάδα εστίασης είναι σε μια συζήτηση

μεταξύ των πληροφοριοδοτών. Τα αποτελέσματα της ομάδας εστίασης δείχνουν, μεταξύ άλλων, ότι μια περίοδος μεταβλητότητας σε μια συγκεκριμένη περιοχή του συστήματος συχνά προβλέπεται από ένα μοτίβο που χαρακτηρίζεται από αδράνεια σε μια συγκεκριμένη περιοχή (με αποτέλεσμα ο τομέας αυτός γίνεται λιγότερο ώριμος από τους άλλους), αυξημένη επικοινωνία μεταξύ προγραμματιστών σχετικά βελτιώσεις σε αυτόν τον τομέα και, κατ' επέκταση, την εμφάνιση ενός 'μεσσία' που αναλαμβάνει την πρωτοβουλία να αρχίσει να εργάζεται σε αυτές τις βελτιώσεις. Οι αρχικές αλλαγές οδηγούν σε περισσότερες αλλαγές (τόσο για την επέκταση των βελτιώσεων που έχουν ήδη πραγματοποιηθεί όσο και για την επίλυση των προβλημάτων που έχουν εισαχθεί), οδηγώντας έτσι σε αστάθεια.

Η επόμενη δημοσίευση επικεντρώνεται στα ονόματα των τοπικών μεταβλητών και σχόλια που είναι σημαντικά αντικείμενα που αντικατοπτρίζουν την προτίμηση του προγραμματιστή. (Aman, Amasaki, Sasaki, & Kawahara, 2015) Διεξάγει μια εμπειρική ανάλυση σχετικά με τη χρησιμότητα αυτών των τεχνουργημάτων για την αξιολόγηση της ποιότητας του λογισμικού από την οπτική γωνία της μεταβλητότητας σε μεθόδους Java που αναπτύχθηκαν σε έξι δημοφιλή προϊόντα λογισμικού ανοιχτού κώδικα: Angry IP Scanner (IP-Scanner), Eclipse Checkstyle Plugin (Checkstyle), FreeMind, GNU ARM Eclipse Plug-ins, Hibernate ORM (Hibernate) and SQuirreL SQL Client (SQuirreL). Τα εμπειρικά αποτελέσματα δείχνουν: (1) μια μέθοδος με μεγαλύτερο μήκος ονόματος τοπικής μεταβλητής είναι πιο επιρρεπής σε αλλαγές και (2) η παρουσία σχολίων μέσα στο σώμα της μεθόδου ενισχύει τις πιθανότητες να τροποποιηθεί Μετα το release. Ενώ μια μεταβλητή με μεγαλύτερο όνομα μπορεί να παρέχει πλουσιότερες πληροφορίες και μπορεί να είναι πιο κατανοητή, οι μεταβλητές με μεγάλη ονομασία ενδέχεται να μην απαιτούνται συχνά. Παρόλο που τα ίδια τα σχόλια δεν έχουν επιβλαβείς επιπτώσεις στο πρόγραμμα, ορισμένοι προγραμματιστές γράφουν λεπτομερή σχόλια για να αντισταθμίσουν την έλλειψη σαφήνειας στα περίπλοκα τμήματα κώδικα.

Η επόμενη δημοσίευση παρόλο που αποτελείται από λίγες σελίδες έχει πληθώρα σχετικών μετρικών. (Moser, 2008) Περιγράφουν ένα πείραμα, το οποίο αναλύει τη σχετική σημασία και τη σταθερότητα των μετρήσεων αλλαγής για την πρόβλεψη ελαττωμάτων για 3 release του Eclipse. Τα αποτελέσματα δείχνουν ότι, από τις 18 μετρικές, οι 3 περιέχουν τις περισσότερες πληροφορίες σχετικά με ελαττώματα λογισμικού. Επιπλέον, αυτές οι 3 μετρικές παραμένουν σταθερές σε 3 release του Eclipse. Οι 18

μετρικές που αξιολογούν την επίδραση της μεταβλητότητας στην αξιοπιστία είναι οι εξής:

- Αναθεωρήσεις: Αριθμός αναθεωρήσεων ενός αρχείου
- Ανακατασκευές: πλήθος ανακατασκευών αρχείου
- BUGFIXES
- AUTHORS: πλήθος μοναδικών προγραμματιστών που ελέγξαν ένα αρχείο
- LOC_ADDED: Άθροισμα όλων των αναθεωρήσεων των γραμμών που προστέθηκαν σε ένα αρχείο
- MAX_LOC_ADDED: Ομοίως με LOC_ADDED
- AVE_LOC_ADDED: Ομοίως με LOC_ADDED
- LOC_DELETED: Ομοίως με LOC_ADDED
- MAX_LOC_DELETED: Ομοίως με LOC_ADDED
- AVE_LOC_DELETED: Ομοίως με LOC_ADDED
- CODECHURN: Άθροισμα (προστιθέμενες γραμμές κώδικα - διαγραμμένες γραμμές κώδικα) σε όλες τις αναθεωρήσεις
- MAX_CODECHURN
- AVE_CODECHURN
- MAX_CHANGESET: Μέγιστος αριθμός αρχείων που έχουν δεσμευτεί μαζί
- AVE_CHANGESET
- AGE: Ηλικία ενός αρχείου σε εβδομάδες (μετρώντας προς τα πίσω από μια συγκεκριμένη έκδοση)

Μια συγκριτική ανάλυση με το πλήρες μοντέλο δείχνει ότι η ακρίβεια της πρόβλεψης δεν επηρεάζεται πάρα πολύ με τη χρήση ενός υποσυνόλου 3 μετρήσεων.

Η ανάλυση μετάδοσης αλλαγών στοχεύει στον εντοπισμό τεχνουργημάτων λογισμικού που επηρεάζονται από μια αλλαγή. Στο παρελθόν, αυτό το πρόβλημα αντιμετωπίστηκε με προσεγγίσεις που βασίζονται σε στατική ανάλυση, δυναμική ανάλυση και ανάλυση κειμένου. Πρόσφατα, διερευνήθηκαν τεχνικές βάσει ιστορικών αναλύσεων και κανόνων συσχέτισης. Με μια δημοσίευση σύντομη αλλά περιεκτική (Ceccarelli, Cerulo, Canfora, & Penta, 2010) προτάθηκε μια νέα μέθοδος ανάλυσης μετάδοσης αλλαγών που βασίζεται στην ιδέα ότι οι αμοιβαίες σχέσεις μεταξύ αντικειμένων λογισμικού μπορούν να συναχθούν με μια προσέγγιση στατιστικής μάθησης. Χρησιμοποίησαν τον έλεγχο αιτιότητας κατά Granger (Granger causality test) , μια προσέγγιση πρόβλεψης πολλαπλών

παραλλαγών χρονομετρών που χρησιμοποιείται για να επαληθεύσουμε ένα οι προηγούμενες τιμές μιας χρονοσειράς είναι χρήσιμες για την πρόβλεψη μελλοντικών τιμών άλλης χρονοσειράς. Η προοπτική ενός προγραμματιστή είναι να κατανοήσει ποια αρχεία θα μπορούσαν ενδεχομένως να επηρεαστούν από αλλαγές που συμβαίνουν σε ένα δεδομένο αρχείο. Τα δεδομένα προήλθαν από σύνολα αλλαγών που εξάγονται από το ιστορικό εκδόσεων του Samba - μια γνωστή υπηρεσία που αναπτύχθηκε για να διασφαλίσει τη συνδεσιμότητα μεταξύ Microsoft Windows και Unixes. Τα προκαταρκτικά αποτελέσματα που ελήφθησαν από 10531 στιγμιότυπα του Samba υποδηλώνουν ότι ο έλεγχος αιτιότητας κατά Granger βοηθά στη σύλληψη των επιπτώσεων της αλλαγής σε σχέση με τους κανόνες συσχέτισης: ενώ οι κανόνες καταγράφουν σχέσεις μεταξύ των αρχείων πηγαίου κώδικα που αλλάζουν μαζί, ο Granger αναγνωρίζει σχέσεις χρονικής αιτιότητας μεταξύ των μεταβαλλόμενων αντικειμένων.

Τα συστήματα λογισμικού εξελίσσονται συνεχώς για να φιλοξενήσουν νέες δυνατότητες και σχέσεις μεταξύ αντικειμένων που δείχνουν όλο και πιο σημαντικές επιπτώσεις στην αλλαγή λογισμικού. Κατά τη διάρκεια της συντήρησης, οι προγραμματιστές πρέπει να διασφαλίσουν ότι οι συ σχετιζόμενες οντότητες ενημερώνονται ώστε να είναι συνεπείς με αυτές τις αλλαγές. Μελέτες στον τομέα της στατική ανάλυση μετάδοσης αλλαγών έχουν εντοπίσει ότι ένας συνδυασμός πληροφοριών πηγαίου κώδικα και λεκτικών ξεπερνά κάθε χρήση όταν καθεμιά από τις δυο παραπάνω υιοθετείται ανεξάρτητα. Ωστόσο, η εξαγωγή λεκτικών πληροφοριών και η μέτρηση του πόσο χαλαρά ή στενά σχετίζονται δύο τεχνουργήματα λογισμικού, λαμβάνοντας υπόψη τις σημασιολογικές πληροφορίες που είναι ενσωματωμένες στα σχόλια, έχει πραγματοποιηθεί χρησιμοποιώντας κάπως περίπλοκες τεχνικές ανάκτησης πληροφοριών (information retrieval). Στην επόμενη δημοσίευση (Ajienka, Capiluppi, & Counsell, An empirical study on the interplay between semantic coupling and co-change of software classes, 2018), στόχος ήταν η κάλυψη και των δύο κενών συγκρίνοντας την αποτελεσματικότητα της μέτρησης της σημασιολογικής σύζευξης κλάσεων αντικειμενοστρεφούς λογισμικού χρησιμοποιώντας (i) απλές τεχνικές που βασίζονται σε αναγνωριστικά και (ii) σωμάτων κείμενων όλων των κλάσεων σε ένα σύστημα λογισμικού. Στη συνέχεια, ερευνάται εμπειρικά η αλληλεπίδραση μεταξύ σημασιολογικής σύζευξης και σύζευξης αλλαγών(semantic and change coupling). Με απλά λόγια, η σημασιολογική σύζευξη είναι ένα μέτρο του πόσο χαλαρά ή στενά σχετίζονται δύο τεχνουργήματα λογισμικού, λαμβάνοντας υπόψη τις σημασιολογικές πληροφορίες που είναι ενσωματωμένες στα σχόλια και τα αναγνωριστικά.

Χρησιμοποίησαν έναν όγκο συνολικά 49459 project γραμμένα σε Java. Τα εμπειρικά αποτελέσματα δείχνουν ότι: (1) οι μέθοδοι που βασίζονται στο αναγνωριστικό έχουν περισσότερη υπολογιστική αποδοτικότητα (2) δεν υπάρχει συσχέτιση μεταξύ σημασιολογικής σύζευξης και σύζευξης αλλαγών. Επιπλέον, διαπίστωσαν ότι (3) υπάρχει σχέση μεταξύ των δύο, καθώς πάνω από το 70% των σημασιολογικών εξαρτήσεων συνδέονται επίσης με την σύζευξη αλλαγών αλλά όχι το αντίστροφο.

Η εξόρυξη κανόνων συσχέτισης είναι μια τεχνική μάθησης χωρίς επίβλεψη που προσβάλλει σχέσεις μεταξύ αντικειμένων σε ένα σύνολο δεδομένων. Αυτή η τεχνική έχει χρησιμοποιηθεί επιτυχώς για να αναλύσει το ιστορικό αλλαγών ενός συστήματος και να αποκαλύψει την εξελικτική σύνδεση μεταξύ των τεχνουργημάτων του συστήματος. Η εξελικτική σύζευξη μπορεί, με τη σειρά της, να χρησιμοποιηθεί για να προτείνει αντικείμενα που ενδεχομένως επηρεάζονται από ένα δεδομένο σύνολο αλλαγών στο σύστημα. Όπως μπορούμε να διαβάσουμε στη δημοσίευση των Moonen, L., Di Alesio, S., Binkley, D., & Rolfsnes, T. (Moonen, Di Alesio, Binkley, & Rolfsnes, 2016), σε γενικές γραμμές, η ποιότητα τέτοιων συστάσεων επηρεάζεται από (1) τις τιμές που επιλέγονται για διάφορες παραμέτρους του αλγορίθμου εξόρυξης, (2) χαρακτηριστικά του συνόλου αλλαγών που χρησιμοποιούνται για τη λήψη μιας σύστασης και (3) χαρακτηριστικά του ιστορικού αλλαγών του συστήματος για τα οποια δημιουργούνται προτάσεις. Σε αυτό το άρθρο, διερευνάται εμπειρικά ο βαθμός στον οποίο ορισμένες επιλογές για αυτούς τους παράγοντες αποτελούν μια πρόταση αλλαγής. Συγκεκριμένα, διεξάγεται μια σειρά συστηματικών πειραμάτων που βασίζεται στο ιστορικό των αλλαγών δύο μεγάλων βιομηχανικών συστημάτων και οκτώ μεγάλων συστημάτων ανοιχτού κώδικα, στα οποία ελέγχεται το μέγεθος του συνόλου αλλαγών για το οποίο πρέπει να αντληθεί μια σύσταση, το μέτρο που χρησιμοποιήθηκε για την αξιολόγηση της αντοχής της εξελικτικής σύζευξης, και το μέγιστο μέγεθος των ιστορικών αλλαγών που λαμβάνονται υπόψη κατά την εξαγωγή αυτών των συνδέσμων. Για κάθε σύστημα, εξάγεται τυχαία ένα αντιπροσωπευτικό δείγμα των συναλλαγών από το ιστορικό αλλαγών, χωρίζεται τυχαία κάθε δείγμα σε δύο μέρη, ένα ερώτημα και ένα αναμενόμενο αποτέλεσμα και αναλύεται μέσα από τον πηγαίο κώδικα πώς διάφορες παράμετροι επηρεάζουν την πρόβλεψη του αναμενόμενου αποτελέσματος με βάση το ερώτημα. Τα αποτελέσματα της μελέτης χρησιμοποιούνται για να αντληθούν

μια σειρά πρακτικών οδηγιών για την εφαρμογή της εξόρυξης κανόνων σύνδεσης για μια πρόταση αλλαγής.

Οι σχέσεις μεταξύ σύζευξης και εξωτερικών παραγόντων ποιότητας αντικειμενοστρεφούς λογισμικού έχουν μελετηθεί εκτενώς τα τελευταία χρόνια. Για παράδειγμα, αρκετές μελέτες έχουν εντοπίσει σαφείς εμπειρικές σχέσεις μεταξύ σύζευξης επιπέδου τάξης και προφοράς κατηγορίας. Ένας κοινός τρόπος καθορισμού και μέτρησης της σύζευξης είναι μέσω δομικών ιδιοτήτων και ανάλυσης στατικού κώδικα. Ωστόσο, λόγω του πολυμορφισμού, της δυναμικής δέσμευσης και της κοινής παρουσίας αχρησιμοποίητου («νεκρού») κώδικα σε εμπορικό λογισμικό, τα προκύπτοντας μέτρα σύζευξης είναι ανακριβή καθώς δεν αντικατοπτρίζουν τέλεια την πραγματική σύζευξη που λαμβάνει χώρα μεταξύ των κλάσεων κατά το χρόνο εκτέλεσης. Για παράδειγμα, όταν χρησιμοποιείται στατική ανάλυση για τη μέτρηση της σύζευξης, είναι δύσκολο και μερικές φορές αδύνατο να προσδιοριστεί ποιες πραγματικές μέθοδοι μπορούν να κληθούν από μια κλάση που παραλαμβάνει ένα αίτημα ένα αυτές οι μέθοδοι παρακαμφθούν στις υποκατηγορίες των κλάσεων που στέλνουν ένα αίτημα. Η μέτρηση σύζευξης εκτελείται παραδοσιακά χρησιμοποιώντας ανάλυση στατικού κώδικα, επειδή το μεγαλύτερο μέρος της έως τώρα δουλείας έγινε σε μη αντικειμενοστρεφή κώδικα και επειδή η δυναμική ανάλυση κώδικα είναι πιο ακριβή και πολύπλοκη. Ωστόσο, για τα σύγχρονα συστήματα λογισμικού, αυτή η εστίαση στη στατική ανάλυση μπορεί να είναι προβληματική, διότι αν και υπήρχε δυναμική δέσμευση πριν από την έλευση του αντικειμενοστρεφή προσανατολισμού, η χρήση του αυξήθηκε σημαντικά πρόσφατα. Στη δημοσίευση των Arisholm, E., Briand, L. C., & Foyen, A. (Arisholm, Briand, & Foyen, 2004) περιγράφεται πώς η σύζευξη μπορεί να οριστεί και να μετρηθεί με ακρίβεια με βάση τη δυναμική ανάλυση των συστημάτων. Αναφέρονται σε αυτόν τον τύπο σύζευξης ως δυναμική σύζευξη. Επιλέχθηκε ένα σύστημα λογισμικού ανοιχτού κώδικα που ονομάζεται Velocity για να αξιολογηθούν τα δυναμικά μέτρα σύζευξης. Υπολόγισαν πρώτα τα περιγραφικά στατιστικά στοιχεία για τη σύζευξη και τα μεγέθη τάξης με βάση την πρώτη sub-release της έκδοσης (1.2) του Velocity. Μερικές από τις μετρικές σύζευξης όπως οι EC_OC, EC_OM, EC_OD, φαίνεται να είναι σημαντικές, συμπληρωματικοί δείκτες μεταβλητότητας όταν συνδυάζονται τόσο με το μέγεθος όσο και με τα στατικά μέτρα ζεύξης.

Η έως τώρα έρευνα δείχνει ότι το μέγεθος της τάξης μπορεί να επηρεάσει τους συσχετισμούς μεταξύ αντικειμενοστρεφών (ΟΟ) μετρικών και μεταβλητότητας. Διαφορετικά, οι πραγματικοί τους συσχετισμοί ενδέχεται να παραμορφωθούν. Ωστόσο, δεν έχει καθοριστεί ένα αυτή η πρακτική εφαρμόζεται εξίσου σε άλλα εξωτερικά ποιοτικά χαρακτηριστικά. Σε αυτή την εργασία των Yuming Zhou, Leung, H., & Baowen Xu ({Xu}, 2009) χρησιμοποιούνται μετρήσεις τριών μετρικών, δύο από τις οποίες είναι διαθέσιμες κατά τη φάση σχεδιασμού υψηλού επιπέδου(high-level design) , για να εξεταστεί η δυνητικά ενοχλητική επίδραση του μεγέθους των κλάσεων στις συσχετίσεις μεταξύ των αντικεινοστραφών μετρικών και της μεταβλητότητας. Οι μετρικές που διερευνώνται περιλαμβάνουν μετρήσεις συνοχής, σύζευξης και κληρονομικότητας. Τα αποτελέσματά μας, βασισμένα στο Eclipse, δείχνουν ότι: 1) Η συγκεχυμένη επίδραση του μεγέθους μιας κλάσης στις συσχετίσεις μεταξύ των αντικεινοστραφών μετρικών και της μεταβλητότητας, γενικά, υπάρχει, ανεξάρτητα από τη μετρική μεγέθους που χρησιμοποιείται 2) η επίδραση του μεγέθους της κλάσης οδηγεί γενικά σε υπερεκτίμηση των συσχετίσεων μεταξύ των αντικειμενοστρεφών μετρικών και της μεταβλητότητας και 3) για πολλές αντικειμενοστρεφείς μετρικές, η επίδραση του μεγέθους της κλάσης εξηγεί πλήρως τις συσχετίσεις τους με τη μεταβλητότητα. Αυτά τα αποτελέσματα υποδηλώνουν έντονα ότι οι μελέτες που επικυρώνουν τις αντικειμενοστρεφείς μετρικές σχετικά με τη μεταβλητότητα πρέπει επίσης να θεωρήσουν το μέγεθος της κλάσης ως αναξιόπιστη μεταβλητή.

Ο σωστός εντοπισμός των κλάσεων που είναι επιρρεπείς σε αλλαγές, στις πρώτες φάσεις του κύκλου ζωής ανάπτυξης λογισμικού βοηθά στην ανάπτυξη οικονομικά αποδοτικού, καλής ποιότητας και βιώσιμου λογισμικού. Ένα αποτελεσματικό μοντέλο πρόβλεψης θα πρέπει να αναγνωρίζει εξίσου κλάσεις με τάση αλλαγής αλλά και όχι επιρρεπείς σε αλλαγές, με υψηλή ακρίβεια. Ωστόσο, αυτό δεν συμβαίνει, καθώς οι επαγγελματίες λογισμικού συχνά πρέπει να αντιμετωπίζουν ανομοιογενή σύνολα δεδομένων, όπου οι περιπτώσεις ενός είδους τάξης είναι πολύ υψηλότερες από το άλλο είδος. Σε ένα τέτοιο σενάριο, οι κλάσεις που είναι λιγότερες δεν προβλέπονται με μεγάλη ακρίβεια οδηγώντας σε στρατηγικές απώλειες. Η μελέτη των Malhotra, R., & Khanna, M (Malhotra & Khanna, An empirical study for software change prediction using imbalanced data, 2017) αξιολογεί μια σειρά τεχνικών για τον χειρισμό ανομοιογενών συνόλων δεδομένων χρησιμοποιώντας διάφορες μεθόδους δειγματοληψίας

δεδομένων και MetaCost learners σε έξι σύνολα δεδομένων ανοιχτού κώδικα. Η μελέτη χρησιμοποιεί τρία πακέτα εφαρμογών του Android και τρία κοινά λογισμικά που αναπτύχθηκαν από την Apache. Η ευρεία χρήση συνόλων δεδομένων Android και apache βελτιώνει την εξωτερική εγκυρότητα της μελέτης καθώς τα αποτελέσματα της μελέτης μπορούν να εφαρμοστούν αποτελεσματικά σε παρόμοια σενάρια. Προκειμένου να αναλυθεί ο πηγαίος κώδικας εξήχθησαν αρχεία καταγραφής αλλαγών μεταξύ δύο διαδοχικών εκδόσεων του συνόλου δεδομένων λογισμικού. Τα αποτελέσματα της μελέτης υποστηρίζουν τη χρήση της μεθόδου της αντικατάστασης για αποτελεσματική μάθηση με ανομοιογενή δεδομένα.

Μια πληθώρα προηγούμενων ερευνών έχει επικεντρωθεί στην πρόβλεψη τω στοιχείων λογισμικού που είναι επιρρεπή σε ελαττώματα. Μια πτυχή αυτών των ερευνών επικεντρώνεται στην πρόβλεψη αλλαγών λογισμικού που προκαλούν διορθώσεις. Αν και η προηγούμενες έρευνες σχετικά με τις διορθωτικές αλλαγές έχει πολλά πλεονεκτήματα όσον αφορά τα εξαιρετικά ακριβή αποτελέσματα, έχει ένα κύριο μειονέκτημα: Προσδίδει την ιδιά επίδραση σε όλες τις αλλαγές που προκαλούν επιδιόρθωση. Ωστόσο στην ερευνά των Misirli, A. T., Shihab, E., & Kamei, Y. (Misirli, Shihab, & Kamei, 2016) υποστηρίζεται ότι η αντιμετώπιση όλων των αλλαγών που προκαλούν διόρθωση δεν είναι ιδανική, καθώς ένα μικρό τυπογραφικό λάθος είναι πιο εύκολο να αντιμετωπιστεί, από έναν προγραμματιστή, από ένα ζήτημα συγχρονισμού μιας αλληλουχίας. Μελετήθηκαν οι αλλαγές που προκαλούν διορθώσεις (high impact fix-inducing changes (HIFCs)) σε επίπεδο αρχιτεκτονικής. Δεδομένου ότι ο αντίκτυπος μιας αλλαγής μπορεί να μετρηθεί με διαφορετικούς τρόπους, προτείνουν πρώτα ένα μέτρο του αντίκτυπου των αλλαγών που προκαλούν επιδιόρθωση, το οποίο λαμβάνει υπόψη τις υλοποιήσεις που πρέπει να γίνουν από προγραμματιστές σε μεταγενέστερες αλλαγές. Η μέτρηση της επίδρασης για μια αλλαγή που προκαλεί διόρθωση χρησιμοποιεί τον αριθμό των αρχείων και τον αριθμό των υποσυστημάτων που τροποποιήθηκαν από προγραμματιστές κατά τη διάρκεια μιας σχετικής διόρθωσης της αλλαγής που προκαλεί την επιδιόρθωση. Επιπλέον μετρικές για τη μέτρηση της μεταβλητότητας είναι οι εξής:

- the total number of subsystems
- Number of modified subsystems
- Number of modified directories
- Number of modified files
- Entropy

- Lines of code added
- Lines of code deleted
- Whether or not the change is a bug fix
- The number of developers that changed the modified files
- The average time interval between the last and the current change
- The number of prior changes to the modified files
- Developer Experience
- Recent Developer Experience
- Developer Experience on a subsystem

Η μελέτη διεξάγεται χρησιμοποιώντας έξι μεγάλα έργα ανοιχτού κώδικα για τη δημιουργία εξειδικευμένων μοντέλων που αναγνωρίζουν HIFCs, προσδιορίζουν τους καλύτερους δείκτες των HIFC και εξετάζουν τα οφέλη από την προτεραιότητα των HIFCs. Διαπιστώνουν ότι οι γραμμές κώδικα που προστέθηκαν, ο αριθμός των προγραμματιστών που εργάστηκαν σε μια αλλαγή και ο αριθμός των προηγούμενων τροποποιήσεων στα αρχεία που τροποποιήθηκαν κατά τη διάρκεια μιας αλλαγής είναι οι καλύτεροι δείκτες των HIFCs.

Πρόσφατα, υπήρξε αυξημένο ενδιαφέρον για τη μετάδοση αλλαγών με βάση την εξελικτική σύζευξη. Μια ιδιαίτερα υποσχόμενη προσέγγιση χρησιμοποιεί την εξόρυξη κανόνων συσχέτισης για να αποκαλύψει πιθανά επηρεαζόμενα αντικείμενα από μοτίβα στο ιστορικό αλλαγών του συστήματος. Δύο βασικές εκτιμήσεις κατά τη χρήση αυτής της προσέγγισης είναι το μέγεθος του ιστορικού, ο αριθμός των συναλλαγών από το ιστορικό αλλαγών που χρησιμοποιήθηκαν για τον εντοπισμό της επίδρασης μιας αλλαγής και η ηλικία του ιστορικού, ο αριθμός των συναλλαγών που έχουν συμβεί από τα μοτίβα που εξορύχθηκαν τελευταία από το ιστορικό. Παρόλο που το μέγεθος και η ηλικία του ιστορικού μπορούν να επηρεάσουν σημαντικά την ποιότητα των αποτελεσμάτων εξόρυξης, υπάρχουν λίγες οδηγίες για τον καλύτερο τρόπο επιλογής κατάλληλων τιμών για αυτές τις δύο παραμέτρους. Στη μελέτη των Moonen, L., Rolfsnes, T., Binkley, D., & Di Alesio, S. (Moonen, Rolfsnes, Binkley, & Di Alesio, 2018) διερευνώνται εμπειρικά οι επιπτώσεις μεγέθους και της ηλικίας του ιστορικού στην ποιότητα της ανάλυσης επιπτώσεων της αλλαγής χρησιμοποιώντας εξορθολογισμένη εξελικτική σύζευξη. Συγκεκριμένα, αναφέρονται μια σειρά συστηματικών πειραμάτων χρησιμοποιώντας τρεις αλγόριθμους εξόρυξης τελευταίας τεχνολογίας που περιλαμβάνουν τα ιστορικά αλλαγών δύο μεγάλων βιομηχανικών συστημάτων και 17 μεγάλων συστημάτων ανοιχτού κώδικα. Σε αυτά τα πειράματα, διαφοροποιούν το μέγεθος και την ηλικία του ιστορικού που χρησιμοποιείται για την εξόρυξη της επίδρασης της αλλαγής λογισμικού και υπολογίζουν πώς αυτό επηρεάζει την ακρίβεια. Καταλήγουνε στο ότι οι τρεις

μετρικές που επηρεάζουν τη μεταβλητότητα είναι οι μέσος αριθμός artifacts σε ένα commit, ο διάμεσος χρόνος μεταξύ των commits, ο αριθμός των μοναδικών artifacts στην ιστορία του συστήματος. Τα αποτελέσματα της μελέτης χρησιμοποιούνται για την εξαγωγή πρακτικών οδηγιών για την επιλογή του κατάλληλου μεγέθους και ηλικίας του ιστορικού κατά την εφαρμογή εξόρυξης κανόνων συσχέτισης για τη διεξαγωγή της ανάλυσης μετάδοσης αλλαγών.

Ανάλογα με το πρότυπο προγραμματισμού που χρησιμοποιείται, την επιλογή της γλώσσας προγραμματισμού για την υλοποίηση και τον σχεδιασμό ενός συστήματος λογισμικού, η σύζευξη επηρεάζεται από διάφορους παράγοντες - όπως ο έλεγχος και η ροή δεδομένων - και ως εκ τούτου μπορεί να μετρηθεί διαφορετικά. Η ισχύς της σύζευξης που μετράτε μεταξύ των modules ενός λογισμικού χρησιμοποιείται συχνά ως προγνωστικός παράγοντας των εξωτερικών χαρακτηριστικών ποιότητας του λογισμικού, όπως η μεταβλητότητα, η αλυσιδωτή αντίδραση των αλλαγών και η μεταβλητότητα. Στην εργασία των Poshyvanyk, D., Marcus, A., Ferenc, R., & Gyimóthy, T. (Poshyvanyk, Marcus, Ferenc, & Gyimóthy, 2009) παρουσιάζεται ένα νέο σύνολο μετρικών σύζευξης για αντικειμενοστρεφή συστήματα λογισμικού (ΟΟ) που μετρούν την εννοιολογική σύζευξη κλάσεων. Η εννοιολογική σύζευξη βασίζεται στη μέτρηση του βαθμού στον οποίο τα αναγνωριστικά και τα σχόλια από διαφορετικές κλάσεις σχετίζονται μεταξύ τους. Αυτός ο τύπος σχέσης, που ονομάζεται εννοιολογική σύζευξη, μετράτε μέσω της χρήσης τεχνικών ανάκτησης πληροφοριών (Information Retrieval (IR)). Η εργασία ερευνά τη χρήση των εννοιολογικών μετρικών σύζευξης κατά την ανάλυση των μετάδοσης αλλαγών. Επίσης, αναφέρει τα ευρήματα μιας μελέτης περίπτωσης στον πηγαίο κώδικα του προγράμματος περιήγησης στο Web Mozilla, όπου οι εννοιολογικές μετρήσεις ζεύξης ήτανε σύγκριση με εννέα υπάρχουσες μετρήσεις δομικής ζεύξης και αποδείχθηκε καλύτερος προγνωστικός παράγοντας για τάξεις που επηρεάζονται από αλλαγές. Οι μετρικές που επηρεάζουν τη μεταβλητότητα σε αυτή την εργασία είναι οι εξής:

- Maximum Conceptual Coupling Between two Classes (CCBCm)
- The number of method invocations in a class ci ∈ C, of methods in a class di ∈ C, weighted by the number of parameters of the invoked methods (ICP)

Θεωρητικά, η ανάλυση μετάδοσης αλλαγών πρέπει να γίνει κατά τη συντήρηση λογισμικού, για να βεβαιωθείτε ότι οι αλλαγές δεν εισάγουν νέα σφάλματα. Προτείνονται πολλές προσεγγίσεις και τεχνικές για να βοηθήσουν τους προγραμματιστές να αλλάξουν αυτόματα την ανάλυση της επίδρασης. Ωστόσο, εξακολουθεί να είναι ένα ανοιχτό ερώτημα ένα και πώς οι προγραμματιστές αλλάζουν την ανάλυση μετάδοσης αλλαγών. Στη δημοσίευση των Jiang, S., McMillan, C., & Santelices, R. (Jiang, McMillan, & Santelices, 2017) διενεργήθηκαν δύο μελέτες, μία εις βάθος μελέτη και μία με μεγάλη έκταση. Για την

εις βάθος μελέτη, καταγράψαν βίντεο εννέα επαγγελματιών προγραμματιστών που επιδιορθώνουν δύο σφάλματα για δύο ώρες. Για την ευρεία μελέτη, ερευνήσαν 35 επαγγελματίες προγραμματιστές χρησιμοποιώντας ένα ηλεκτρονικό σύστημα. Διαπιστώσαν ότι οι προγραμματιστές στις μελέτες τους πραγματοποίησαν ανάλυση μετάδοσης στατικών αλλαγών προτού πραγματοποιήσουν αλλαγές χρησιμοποιώντας λειτουργίες πλοήγησης IDE και έκαναν ανάλυση δυναμικών αλλαγών μετά την πραγματοποίηση αλλαγών εκτελώντας τα προγράμματα. Διαπιστώσαν επίσης ότι δεν χρησιμοποίησαν κανένα εργαλείο ανάλυσης μετάδοσης αλλαγών.

Το πρόβλημα της εύθραυστης κλάσης (fragile base-class problem (FBCP)) είναι ένα θεμελιώδες αρχιτεκτονικό πρόβλημα των αντικειμενοστρεφών συστημάτων προγραμματισμού όπου οι βασικές τάξεις (superclasses) θεωρούνται «εύθραυστες» επειδή φαινομενικά ασφαλείς τροποποιήσεις σε μια βασική τάξη, όταν κληρονομούνται από τις παράγωγες τάξεις, μπορεί να προκαλέσουν δυσλειτουργία των παραγόμενων τάξεων . Ο προγραμματιστής δεν μπορεί να προσδιορίσει ένα μια αλλαγή κατηγορίας βάσης είναι ασφαλής απλά εξετάζοντας μεμονωμένα τις μεθόδους της βασικής τάξης. Πολλά ερευνητικά έργα έχουν επικεντρωθεί στην αποτροπή του FBCP προτείνοντας εναλλακτικούς μηχανισμούς για επαναχρησιμοποίηση, αλλά, εξ όσων είναι γνωστά, δεν υπάρχει προηγούμενη ερευνητική εργασία που να μελετά τον αντίκτυπο του FBCP σε συστήματα λογισμικού πραγματικού κόσμου. Ο στόχος της μελέτης των Sabané, A., Guéhéneuc, Y.-G., Arnaoudova, V., & Antoniol, G. (Sabané, Guéhéneuc, Arnaoudova, & Antoniol, 2017) είναι διπλός: (1) να αξιολογήσει, σε διαφορετικά συστήματα, τον αντίκτυπο των μικρό-αρχιτεκτονικών, που ονομάζονται FBCS, που θα μπορούσαν να οδηγήσουν σε δύο πτυχές του FBCP, (2) να διερευνήσει τη σχέση μεταξύ των εντοπισμένων συμβάντων και της ποιότητας των συστημάτων από την άποψη της αλλαγής του σφάλματος, και (3) αξιολογεί ένα υπάρχουν σφάλματα σε αυτά τα συστήματα που σχετίζονται με το FBCP. Εκτελούν ποσοτική και ποιοτική μελέτη. Ποσοτικά, αναλύουν πολλές εκδόσεις επτά διαφορετικών συστημάτων ανοιχτού κώδικα που χρησιμοποιούν 58 διαφορετικά πλαίσια, με αποτέλεσμα 301 παραμετροποιήσεις. Εντοπίζουμε σε αυτά τα συστήματα 112.263 περιστατικά FBCS και αναλύουμε ένα οι κλάσεις που παίζουν το ρόλο των υποκατηγοριών στις εμφανίσεις FBCS είναι περισσότερο επιρρεπείς σε αλλαγές και /ή σε σφάλματα από άλλες κλάσεις . Τα αποτελέσματα δείχνουν ότι οι κλάσεις που συμμετέχουν στο FBCS που έχουν αναλυθεί δεν είναι πιθανότερο να αλλάξουν ούτε είναι πιθανότερο να έχουν σφάλματα. Ποιοτικά, διεξάγουν μια έρευνα για να επιβεβαιωθεί ότι ορισμένα σφάλματα σχετίζονται με το FBCP. Η έρευνα περιλαμβάνει 41 συμμετέχοντες που αναλύουν συνολικά 104 σφάλματα τριών συστημάτων ανοιχτού κώδικα. Τα αποτελέσματα δείχνουν ότι κανένα από τα σφάλματα που αναλύθηκαν δεν σχετίζεται με το FBCP. Έτσι, παρά τις μεγάλες, αυστηρές ποσοτικές και ποιοτικές μελέτες, συμπεραίνεται ότι οι δύο πτυχές του FBCP που αναλύουν μπορεί να μην είναι τόσο προβληματικές από την άποψη της

αλλαγής και της προφορικής βλάβης όπως πιστεύεται προηγουμένως στη βιβλιογραφία.

Πολλές μελέτες έχουν διερευνήσει τις σχέσεις μεταξύ αντικειμενοστρεφών (ΟΟ) μετρικών και μεταβλητότητας και καταλήγουν στο συμπέρασμα ότι οι μετρικές είναι σε θέση να προβλέψουν την έκταση της αλλαγής μιας κλάσης στη διάρκεια των εκδόσεων ενός συστήματος. Ωστόσο, υπάρχει ανάγκη επανεξέτασης αυτού του θέματος για δύο λόγους. Πρώτον, οι περισσότερες μελέτες αναλύουν μόνο έναν μικρό αριθμό μετρικών και, επομένως, δεν είναι σαφές ένα αυτό το συμπέρασμα ισχύει για τις περισσότερες μετρικές. Δεύτερον, οι περισσότερες μελέτες χρησιμοποιούν μόνο λίγα σχετικά συστήματα για να διερευνήσουν τις σχέσεις μεταξύ των μετρικών και της μεταβλητότητας. Επομένως, δεν είναι σαφές ένα αυτό το συμπέρασμα μπορεί να γενικευτεί σε άλλα συστήματα. Η δημοσίευση των Lu, H., Zhou, Y., Xu, B., Leung, H., & Chen, L. (Lu, Zhou, Xu, Leung, & Chen, 2012) βασίζεται σε 102 συστήματα Java, οπού χρησιμοποιούνται στατιστικές τεχνικές ανάλυσης για να διερευνηθεί η ικανότητα 62 μετρικών να προβλέψουν τη μεταβλητά. Στο πλαίσιο της μελέτης, μια κλάση που αλλάζει στην επόμενη έκδοση ενός συστήματος ονομάζεται επιρρεπής σε αλλαγή. Οι μετρικές που διερευνήθηκαν καλύπτουν τέσσερις μετρικές διαστάσεις, συμπεριλαμβανομένων 7 μετρικές μεγέθους, 18 μετρικές συνοχής, 20 μετρήσεων σύζευξης και 17 μετρικές κληρονομικότητας:

•Information-f low -based cohesion (ICH)

•LCOM

•CAMC

•LCOM2

•Cohesion (Coh)

•LCOM3

•Normalised Hamming distance based cohesion (NHD)

•LCOM5

• Scaled NHD (SNHD)

•OCC

•LCOM4

•PCC

•Connectivity (Co)

•TCC

•DCD

•Loose class cohesion (LCC)

•DCI

•CBO

•NIH-ICP

•Information-f low -based coupling (ICP)

•OMMIC

•MPC

•RFC

•The number of connected components in a graph where the vertices are the methods of a class an edge exists if the class uses the methods of other classes (OCMIC)

•DAC

•OCAIC

•OMMEC

•OCAEC

•AMMIC

•IH-ICP

•Coupling based on inheritance (CBI)

•DMMEC

•the count of class-method relationships between a class and the methods of the classes that are not friend or ancestor classes (OCMEC)

•ACAIC

•DCMEC

•ACMIC

•DCAEC

•number of added methods

• Static polymorphism in inheritance relations (SP)

•Dynamic polymorphism in inheritance relations (DP)

•Static polymorphism in ancestors (SPA)

•Number of methods overridden (NMO)

•Dynamic polymorphism in ancestor (DPA)

•Dynamic polymorphism in descendants (DPD)

•Class-to-leaf depth (CLD)

•NOC

•Number of descendents (NOD)

• Static polymorphism in descendants (SPD)

•Number of methods inherited (NMI)

•NOP

•Average inheritance depth of a class (AID)

•DIT

•Number of ancestors (NOA)

•Specialization index (SIX)

• The number of declaration and executable statements in the methods of a class (Stmts)

•The noncommentary source lines of code in a class (SLOC)

•The number of methods implemented in a class (NMIMP)

•The sum of the number of parameters of the methods implemented in a class (NumPara)

•The number of attributes in a class, excluding inherited ones (NAIMP)

•The number of attributes in a class, both inherited and noninherited (NA)

•The number of methods in a class, both inherited and noninherited (NM)

Χρησιμοποιούν το AUC (περιοχής κάτωθι της καμπύλης, ROC) για να αξιολογήσουν την προγνωστική αποτελεσματικότητα των μετρικών. Για κάθε μετρική, υπολογίζουν πρώτα τα AUCs και τις αντίστοιχες διακυμάνσεις για μεμονωμένα συστήματα. Στη συνέχεια, χρησιμοποιούν ένα μοντέλο για τον υπολογισμό του μέσου AUC σε όλα τα συστήματα. Τέλος, πραγματοποιούν μια ανάλυση ευαισθησίας για να διερευνήσουν ένα το αποτέλεσμα της AUC από το μοντέλο είναι ισχυρό στην προκατάληψη της επιλογής δεδομένων σε αυτήν τη μελέτη.

Το τεστάρισμα είναι η πιο διαδεδομένη πρακτική για τη διασφάλιση της ποιότητας του λογισμικού. Ωστόσο, αυτή η δραστηριότητα αποτελεί συχνά συμβιβασμό μεταξύ των διαθέσιμων πόρων και της ποιότητας του λογισμικού. Στην αντικειμενοστρεφή ανάπτυξη, η προσπάθεια για τεστ πρέπει να επικεντρώνεται σε ελαττωματικές τάξεις. Δυστυχώς, ο προσδιορισμός αυτών των τάξεων είναι μια δύσκολη και δύσκολη δραστηριότητα στην οποία έχουν δοκιμαστεί πολλές μετρήσεις, τεχνικές και μοντέλα. Στη δημοσίευση των Kpodjedo, S., Ricca, F., Galinier, P., Guéhéneuc, Y.-G., & Antoniol, G. (Kpodjedo, 2011) ερευνάται η χρησιμότητα μετρικών του design για τον εντοπισμό ελαττωματικών κλάσεων. Οι μετρικές περιλαμβάνουν τους αριθμούς των προστιθέμενων, διαγραμμένων και τροποποιημένων μεθόδων και σχέσεων. Οι μετρικές χρησιμοποιούνται για να προτείνουν μια κατάταξη λίστας των κλάσεων που ενδέχεται να περιέχουν ελαττώματα για ένα σύστημα. Συγκρίνονται με τις μετρικές των Chidamber και Kemerer. Περαιτέρω σύγκριση πραγματοποιείται με τις μετρικές πολυπλοκότητας που υπολογίζονται από τους Zimmermann et al. σε αρκετές εκδόσεις του Eclipse, Μετρικές μεταβλητότητας είναι οι εξής:

- number of Added Methods
- number of Added Attributes
- number of Added Outgoing Relations
- number of Added Incoming Relations
- number of Deleted Methods
- number of Deleted Attributes
- number of Deleted Outgoing Relations
- number of Deleted Incoming Relations
- number of Modified Methods
- number of Modified Outgoing Relations
- number of Modified Incoming Relations

Δείχνουν ότι οι μετρικές design, όταν χρησιμοποιούνται σε συνδυασμό με γνωστές μετρικές, βελτιώνουν τον εντοπισμό ελαττωματικών κλάσεων. Επιπλέον, δείχνουν ότι οι μετρικές design κάνουν σημαντικά καλύτερες προβλέψεις για την πυκνότητα ελαττωμάτων από άλλες μετρικές και, ως εκ τούτου, μπορούν να βοηθήσουν στη

μείωση της προσπάθειας δοκιμών εστιάζοντας τη δραστηριότητα δοκιμής στη μείωση του όγκου του κώδικα.

Η εμπειρική έρευνα για το δωρεάν / ελεύθερο / Ανοιχτού κώδικα λογισμικό (FLOSS) έδειξε ότι οι προγραμματιστές τείνουν να συσπειρώνονται γύρω από δύο βασικούς ρόλους: οι «βασικοί» προγραμματιστές διαφέρουν από τους «περιφερειακούς» προγραμματιστές κυρίως επειδή έχουν έναν μεγαλύτερο αριθμό ευθυνών και ένα υψηλότερη παραγωγικότητα. Ένας περαιτέρω, οριζόντιος χαρακτηρισμός των προγραμματιστών θα μπορούσε να επιτευχθεί συνδέοντας τους προγραμματιστές με "χρονοθυρίδες" και διαφορετικά πρότυπα δραστηριότητας και προσπάθειας θα μπορούσαν να συσχετιστούν με αυτά τα χρονικά διαστήματα. Μια τέτοια ανάλυση θα μπορούσε να χρησιμοποιηθεί όχι μόνο για τη σύγκριση διαφορετικών κοινοτήτων FLOSS, και για την αξιολόγηση της σταθερότητας και της ωριμότητάς τους, αλλά και για τον προσδιορισμό εντός του εκάστοτε project, του τρόπο κατανομής της προσπάθειας σε μια δεδομένη περίοδο και για την εκτίμηση των μελλοντικών αναγκών σε σχέση με βασικά σημεία στον κύκλο ζωής του λογισμικού (π.χ. σημαντικές εκδόσεις). Στη δημοσίευση των Capiluppi, A., & Izquierdo-Cortázar, D. (Capiluppi, 2013). αναλύονται τα μοτίβα δραστηριότητας στο πρόγραμμα πυρήνα Linux, αρχικά εστιάζοντας στη συνολική κατανομή της προσπάθειας και της δραστηριότητας εντός εβδομάδων και ημερών και στη συνέχεια, διαιρώντας κάθε μέρα σε τρεις βάρδιες των 8 ωρών και εστιάζοντας στην προσπάθεια και τη δραστηριότητα γύρω από σημαντικές εκδόσεις. Τέτοιες αναλύσεις έχουν ως στόχο την αξιολόγηση της προσπάθειας και της παραγωγικότητας και γύρω από σημαντικές εκδόσεις. Τα αποτελέσματα αυτής της έρευνας δείχνουν ότι, συνολικά, η προσπάθεια μέσα στην κοινότητα του πυρήνα του Linux (case study) είναι σταθερή (αν και σε διαφορετικά επίπεδα) καθ' 'όλη τη διάρκεια της εβδομάδας, σηματοδοτώντας την ανάγκη για ανανεωμένα μοντέλα εκτίμησης, διαφορετικά από αυτά που χρησιμοποιούνται στις παραδοσιακές εταιρείες που δουλεύουν 9 π.μ. - 5 μ.μ., Δευτέρα έως Παρασκευή. Γίνεται επίσης προφανές ότι η δραστηριότητα πριν από μια έκδοση είναι πολύ διαφορετική από τη μετά την κυκλοφορία και ότι οι αλλαγές δείχνουν αύξηση της πολυπλοκότητας του κώδικα σε συγκεκριμένες χρονικές περιόδους (ιδίως στις αργά το βράδυ), η οποία θα απαιτήσει αργότερα πρόσθετες προσπάθειες συντήρησης. Επίδραση στη μεταβλητότητα είχαν οι γραμμές του κώδικα που αλλάχθηκαν.

3.3 Σύνοψη

Ο πίνακας των μετρικών που μπορούν να καθορίσουν άμεσα ή έμμεσα τη μεταβλητότητα λογισμικού ακολουθεί:

Πίνακας 2

Ονόματα μετρικών	**Περιγραφή**
System Design Instability (SDI)	Η μετρικη System Design Instability (SDI) υποδεικνύει την πρόοδο ενός αντικειμενοστρεφούς έργου (OO) όταν το έργο ενεργοποιηθεί
Dynamic polymorphism in ancestor (DPA)	Ο αριθμός των δυναμικά πολυμορφικών συναρτήσεων που εμφανίζονται σε μια κλάση και στις γονικές κλάσεις της.
Dynamic polymorphism in descendants (DPD)	Ο αριθμός των δυναμικά πολυμορφικών συναρτήσεων που εμφανίζονται σε μια κλάση και στις κλάσεις -κληρονόμους της.
Dynamic polymorphism in inheritance relations (DP)	DP = DPA + DPD
EC_OC	σύζευξη εξαγωγής σε διακριτές κλάσεις
EC_OD	σύζευξη εξαγωγής σε δυναμικά μηνύματα
EC_OM	σύζευξη εξαγωγής σε διακριτές μεθόδους
number of changed lines	
Scaled NHD (SNHD)	Scaled NHD
Static polymorphism in descendants (SPD)	Ο αριθμός των πολυμορφικών συναρτήσεων που εμφανίζονται σε μια κλάση και στις κλάσεις -κληρονόμους της.
Static polymorphism in inheritance relations (SP)	SP = SPA+SPD
The number of declaration and executable statements in the methods of a class (Stmts)	
ACAIC	Αυτές οι μετρικές σύζευξης μετράν αλληλεπιδράσεις μεταξύ κλάσεων. Τα ακρωνύμια για τις μετρικές υποδεικνύουν τις μετρήσεις των αλληλεπιδράσεων: 1. Το πρώτο γράμμα δείχνει τη σχέση (A: σύζευξη με γονικές, D: κλασεις κληρονομομοι, O: άλλοι, δηλαδή χωρίς σχέσεις κληρονομιάς). 2. Τα επόμενα δύο γράμματα υποδεικνύουν τον τύπο αλληλεπιδράσεων: (1) CA: υπάρχει αλληλεπίδραση χαρακτηριστικού κλάσης μεταξύ κλάσεων c και d, εάν το c έχει χαρακτηριστικό κλάσης τύπου d, (2) CM: υπάρχει αλληλεπίδραση μεταξύ των κατηγοριών c και d, εάν το c έχει μια μέθοδο με μια παράμετρο του τύπου class d.
ACMIC	
DCAEC	
OCAEC	
DCMEC	
OCAIC	
OCMIC	
OCMEC	
AMMIC	
DMMEC	
OMMIC	
OMMEC	
Average inheritance depth of a class (AID)	Το AID μιας κλάσης χωρίς γονέα είναι μηδέν. Για όλες τις άλλες κατηγορίες, το AID μιας κλάσης είναι το μέσο AID των γονικών του τάξεων, αυξημένο κατά ένα.

average number of artifacts in a commit	μέσος αριθμός αντικειμένων σε ένα commit
Behavioral dependency Measure (BDM)	μπορεί να προέλθει όχι μόνο από τις δομικές πληροφορίες, αλλά και από τις συμπεριφορικές πληροφορίες των μοντέλων σχεδιασμού UML 2.0
CAMC	Συνοχή μεταξύ των μεθόδων των τάξεων. Υπολογίζεται με τον τύπο n × l PO, όπου n είναι ο αριθμός των μεθόδων μιας κλάσης και l είναι ο αριθμός των διαφορετικών τύπων παραμέτρων σε αυτές τις μεθόδους
CBO	Σύζευξη μεταξύ κλάσεων. Ο αριθμός των διαφορετικών τάξεων που δεν έχουν μεταξύ τους κληρονομικότητα.
Class Implementation Instability (CII)	Μετρά τις αλλαγές σχεδιασμού σε επίπεδο κλάσης
Class-to-leaf depth (CLD)	Ο μέγιστος αριθμός επιπέδων στην ιεραρχία που είναι κάτω από την κλάση.
Cohesion (Coh)	Παραλλαγή της LCOM5
Connectivity (Co)	Έστω V είναι το σύνολο κορυφών του γραφήματος G από την LCOM4, και E το σύνολο των άκρων του. Τότε Co = 2(|E| - |V| + 1) / ((|V| - 1)(|V| - 2))
Coupling based on inheritance (CBI)	Ο αριθμός των κλάσεων κληρονόμων μιας κλάσης και το άθροισμα της πολυπλοκότητας της εσωτερικής μεθόδου όλων των μεθόδων που εφαρμόζονται στην κλάση.
DAC	Data abstraction coupling. Ο αριθμός των χαρακτηριστικών σε μια κλάση που υπάρχουν σε άλλη κλάση
DCD	Ομοίως με την TCC, δύο μέθοδοι θεωρούνται συνδεδεμένες, εάν (1) μία καλεί την άλλη μέθοδο. (2) χρησιμοποιούν άμεσα ή έμμεσα κοινά χαρακτηριστικά ή (3) καλούν άμεσα ή έμμεσα κοινές μεθόδους
DCI	Ομοίως με την LCC, δύο μέθοδοι θεωρούνται συνδεδεμένες, εάν (1) η μία καλέσει μια άλλη μέθοδο. (2) χρησιμοποιούν άμεσα ή έμμεσα κοινά χαρακτηριστικά ή (3) καλούν άμεσα ή έμμεσα κοινές μεθόδους
Depth of Inheritance	ο μέγιστος αριθμός επιπέδων έως τον ριζικό κόμβο του δέντρου κλάσεων
Developer Experience	
Developer Experience on a subsystem	
developers' Communication	επικοινωνία που πραγματοποιείται στο σύστημα παρακολούθησης ζητημάτων (issue tracker system)
DIT	Depth of inheritance tree. Το μήκος της μακρύτερης διαδρομής από την κλάση

	προς τη ρίζα στην ιεραρχία κληρονομικότητας
Entropy	Κατανομή τροποποιημένου κώδικα σε κάθε αρχείο
IH-ICP	Ομοίως με την ICP, αλλά μετράει τις κλήσεις μεθόδων γονικών κλάσεων (δηλαδή, σύζευξη βάσει κληρονομιάς)
Information-f low -based cohesion (ICH)	ο αριθμός κλήσεων άλλων μεθόδων της ίδιας κλάσης, σταθμισμένος με τον αριθμό των παραμέτρων της κλήσης μεθόδου
Information-f low -based coupling (ICP)	Ο αριθμός των κλήσεων μεθόδου σε μια κλάση ci ∈ C, των μεθόδων σε μια τάξη di C, σταθμισμένη με τον αριθμό των παραμέτρων των μεθόδων που καλούνται. Το μέτρο λαμβάνει επίσης υπόψη τον πολυμορφισμό
issues	Η επικοινωνία και η συνεργασία των προγραμματιστών που περιστρέφεται γύρω από αιτήματα αλλαγής λογισμικού
LCOM	Lack of cohesion in methods. Ο αριθμός ζευγαριών μεθόδων στην κλάση χωρίς κοινό χαρακτηριστικό.
LCOM2	Ο αριθμός ζευγαριών μεθόδων στην κλάση που δεν χρησιμοποιεί κοινά χαρακτηριστικά μείον τον αριθμό ζευγαριών μεθόδων που το κάνουν. Εάν αυτή η διαφορά είναι αρνητική, το LCOM2 γίνεται μηδέν
LCOM4	Έστω γράφημα G, όπου οι κορυφές είναι οι μέθοδοι μιας κλάσης και υπάρχει ένα άκρο μεταξύ δύο κορυφών εάν οι αντίστοιχες μέθοδοι χρησιμοποιούν τουλάχιστον ένα κοινό χαρακτηριστικό. Το LCOM3 ορίζεται ως ο αριθμός των συνδεδεμένων στοιχείων του G. Ομοίως με το LCOM3, οπού το γράφημα G έχει επιπλέον ένα άκρο μεταξύ κορυφών που αντιπροσωπεύουν τις μεθόδους m1 και m2 εάν το m1 επικαλείται m2 ή αντίστροφα.
LCOM5	Έστω ένα σύνολο μεθόδων {mi} (i = 1,…, n) που έχει πρόσβαση σε ένα σύνολο χαρακτηριστικών {aj} (j = 1,…, k). Έστω το μ (aj) ο αριθμός των μεθόδων που αναφέρονται στο χαρακτηριστικό aj. Τότε LCOM5 = $\left(n - \frac{1}{k}\sum_{j=1}^{k} \mu(a_j)\right)/(n-1)$
lengths of local variable names	το πλήθος των γραμμάτων και το πλήθος των λέξεων. η μέτρηση του αριθμού λέξεων βασίζεται στη λογική που συνήθως χρησιμοποιείται στην Java,

	όπου ο αριθμός λέξεων είναι ο αριθμός των πεζών γραμμάτων ακολουθούμενο από ένα κεφαλαίο γράμμα, συν ένα
LIC	Γραμμές εσωτερικών σχολίων
Lines of code added	
Lines of code deleted	
Loose class cohesion (LCC)	Ομοίως με το TCC, εκτός του ότι αυτή η μετρικη λαμβάνει επίσης υπόψη ζεύγη έμμεσα συνδεδεμένων μεθόδων
Maximum Conceptual Coupling Between two Classes (CCBCm)	
MPC	Message passing coupling. Ο αριθμός κλήσεων μεθόδων σε μια κλάση
NIH-ICP	
NOC	Number of children. Ο αριθμός των κλάσεων που κληρονομούνται άμεσα από μια κλάση.
NOP	Number of parents. Ο αριθμός των κλάσεων από τις οποίες κληρονομείται άμεσα μια τάξη.
NHD	Normalised Hamming distance based cohesion
number of Added Attributes	Μόλις η εξέλιξη των κλάσεων είναι διαθέσιμη, μετράμε τον αριθμό των απλών αλλαγών στο σχεδιασμό. θεωρούνται ως σχέσεις: ενώσεις, συγκεντρώσεις και γενικεύσεις (associations, aggregations, and generalizations)
number of Added Incoming Relations	
number of Added Methods	
number of Added Outgoing Relations	
number of Deleted Incoming Relations	
number of Deleted Methods	
number of Deleted Outgoing Relations	
number of Modified Incoming Relations	
number of Modified Methods	
number of Modified Outgoing Relations	
Number of ancestors (NOA)	Ο αριθμός κλάσεων από τις οποίες κληρονομεί μια τάξη άμεσα ή έμμεσα
Number of descendents (NOD)	Ο αριθμός των κλάσεων που κληρονομούν άμεσα ή έμμεσα από μια τάξη.
Number of methods inherited (NMI)	Ο αριθμός των μεθόδων σε μια κλάση που η κλάση κληρονομεί από τους προγόνους της και δεν παρακάμπτει.
Number of methods overridden (NMO)	Ο αριθμός των μεθόδων σε μια κλάση που παρακάμπτουν μια μέθοδο που κληρονομήθηκε από μια γονική κλάση.
Number of modified directories	
Number of modified subsystems	
number of source code lines changed	
number of words used to describe the issues	
OCC	Έστω γράφημα G, οπού οι κορυφές είναι οι μέθοδοι μιας κλάσης και υπάρχει ένα άκρο μεταξύ δύο κορυφών εάν οι αντίστοιχες μέθοδοι αναφέρονται άμεσα ή

	έμμεσα τουλάχιστον ένα κοινό χαρακτηριστικό. Έστω n ο συνολικός αριθμός μεθόδων στην τάξη. Τότε, το OCC είναι το μέγιστο των αναλογιών του αριθμού των μεθόδων που μπορούν να επιτευχθούν μέθοδο mi (1≤i≤n) στην τάξη για n-1 εάν n> 1 ή 0 διαφορετικά.
PCC	Ομοίως με το OCC, Έστω γράφημα G, στο οποίο υπάρχει ένα τόξο από μια κορυφή που αντιπροσωπεύει τη μέθοδο m1 σε άλλη κορυφή που αντιπροσωπεύει τη μέθοδο m2 εάν το m1 γράφει άμεσα ή έμμεσα το χαρακτηριστικό a και το m2 διαβάζει άμεσα ή έμμεσα a
Recent Developer Experience	
RFC	Response set for classes. Το σύνολο απαντήσεων μιας κλάσης αποτελείται από το σύνολο M μεθόδων της κλάσης και το σύνολο μεθόδων που επικαλούνται άμεσα ή έμμεσα με μεθόδους στο M. RFC είναι ο αριθμός των μεθόδων στο σύνολο απαντήσεων της κλάσης
Specialization index (SIX)	Specialization index. SIX = NMO * DIT / (NMO+NMA+NMI)
Static polymorphism in ancestors (SPA)	Ο αριθμός των πολυμορφικών συναρτήσεων που εμφανίζονται σε μια κλάση και στις γονικές κλάσεις της
System Implementation Instability	μετρά τις αλλαγές στην υλοποίηση αντικειμενοστρεφούς συστήματος , π.χ. η διαφορά του LOC από το Design N στο Design N 1.1
TCC	Tight class cohesion. Εκτός από μεθόδους που χρησιμοποιούν άμεσα χαρακτηριστικά, αυτή η μέτρηση λαμβάνει υπόψη χαρακτηριστικά που χρησιμοποιούνται έμμεσα από μια μέθοδο
The average time interval between the last and the current change	
the count of class-method relationships between a class and the methods of the classes that are not friend or ancestor classes (OCMEC)	
the median inter-commit time	
The noncommentary source lines of code in a class (SLOC)	
The number of connected components in a graph where the vertices are the methods of a class an edge exists if the class uses the methods of other classes (OCMIC)	

The number of developers that changed the modified files	
The number of method invocations in a class ci ∈ C, of methods in a class di ∈ C, weighted by the number of parameters of the invoked methods (ICP)	
The number of methods implemented in a class (NMIMP)	
The number of prior changes to the modified files	
the number of unique artifacts in the history	
The sum of the number of parameters of the methods	
the total churn	
the total number of subsystems	το σύνολο των τροποποιημένων υποσυστημάτων
the total number of files	
Whether or not the change is a bug fix	
REVISIONS	Αριθμός αναθεωρήσεων ενός αρχείου
REFACTORINGS	Αριθμός αναδιαμορφώσεων ενός αρχείου
BUGFIXES	Πόσες φορές έχει διορθωθεί ένα αρχείο
AUTHORS	Αριθμός διακριτών συγγραφέων που έλεγξαν ένα αρχείο στο αποθετήριο (repository)
LOC_ADDED	όλες οι αναθεωρήσεις των γραμμών κώδικα που προστέθηκαν σε ένα αρχείο
MAX_ LOC_ADDED	Μέγιστος αριθμός γραμμών κώδικα που προστέθηκαν για όλες τις αναθεωρήσεις
AVE_ LOC_ADDED	Μέσος ορός γραμμών κώδικα που προστέθηκαν ανά αναθεώρηση
LOC_DELETED	αναθεωρήσεις των γραμμών κώδικα που διαγράφηκαν από ένα αρχείο
MAX_ LOC_DELETED	Μέγιστος αριθμός γραμμών κώδικα που διαγράφηκαν για όλες τις αναθεωρήσεις
AVE_ LOC_DELETED	Μέσος ορός γραμμών κώδικα που διαγράφονται ανά αναθεώρηση
CODECHURN	Άθροισμα (προστιθέμενες γραμμές κώδικα - διαγραμμένες γραμμές κώδικα) σε όλες τις αναθεωρήσεις
MAX_ CODECHURN	Μέγιστο CODECHURN για όλες τις αναθεωρήσεις
AVE_ CODECHURN	Μέσος όρος CODECHURN για όλες τις αναθεωρήσεις
MAX_CHANGESET	Μέγιστος αριθμός αρχείων που έχουν γίνει commit μαζί στο αποθετήριο(repository)
AVE_CHANGESET	Μέσος ορός αρχείων που έχουν γίνει commit μαζί στο αποθετήριο(repository)
AGE	Ηλικία ενός αρχείου σε εβδομάδες (μετρώντας προς τα πίσω από μια συγκεκριμένη έκδοση)

WEIGHTED_AGE	Άθροισμα αριθμού των γραμμών που προστέθηκαν κατά τη διάρκεια μιας εβδομάδας σταθμισμένο με τον συνολικό αριθμό γραμμών που προστέθηκαν σε αυτό το αρχείο

4. Ποσότητα αλλαγής

4.1 Συλλογή άρθρων

Η δυνατότητα αλλαγής του λογισμικού μπορεί να θεωρηθεί ως η ικανότητα αλλαγής, η οποία, μεταξύ άλλων, υποδηλώνει την προσπάθεια που χρειάζεται για αλλαγή στο λογισμικό. Υποτίθεται ότι οι δομικές ιδιότητες του λογισμικού επηρεάζουν τη δυνατότητα αλλαγής, οπότε η μέτρηση αυτών των ιδιοτήτων μπορούν να χρησιμοποιηθούν ως δείκτες μεταβλητότητας. Στη δημοσίευση του Arisholm, E. (Arisholm E. , Empirical assessment of the impact of structural properties on the changeability of object-oriented software, 2006) οι τρόποι με τους οποίους μπορούν να μετρηθούν οι δομικές ιδιότητες του λογισμικού περιγράφονται και επικυρώνονται εμπειρικά με βάση δεδομένα που συλλέγονται από ένα βιομηχανικό project σε Java. Οι μετρικές επικυρώνονται χρησιμοποιώντας τις ως υποψήφιες μεταβλητές σε ένα μοντέλο παλινδρόμησης πολλαπλών παραλλαγών της πραγματικής προσπάθειας που απαιτείται για την πραγματοποίηση τροποποιήσεων στο εξελισσόμενο σύστημα λογισμικού. Πιστεύεται συνήθως ότι το μέγεθος συμβάλλει σημαντικά στην «πολυπλοκότητα». Ωστόσο, απέδειξαν ότι στατική και δυναμική σύζευξη μπορούν να συνδυαστούν ώστε να βελτιώσουν ακόμη περισσότερο την ακρίβεια τέτοιων μοντέλων πρόβλεψης. Οι παραπάνω μελέτες χρησιμοποίησαν ως μετρικές τις γραμμές κώδικα που προστέθηκαν και διαγράφηκαν αντί της πραγματικής ποσότητας αλλαγής. Κατά συνέπεια, μπορούν να βοηθήσουν τους προγραμματιστές στη φάση της υλοποίησης να εντοπίσουν και να διορθώσουν τα προβλήματα σχεδιασμού κατά την ανάπτυξη και συντήρηση αντικειμενοστρεφούς λογισμικού.

Η συνοχή αναφέρεται στον βαθμό συγγένειας των μελών σε μια κλάση. Έχουν προταθεί αρκετά μέτρα συνοχής για τη μέτρηση της συνοχής των κλάσεων σε ένα αντικειμενοστρεφές πρόγραμμα. Ωστόσο, οι υπάρχουσες μετρικές συνοχής δεν διαφοροποιούν τις αλληλεπιδράσεις εγγραφής από τις αλληλεπιδράσεις ανάγνωσης μεταξύ των μελών της κλάσης, επομένως, δεν αντικατοπτρίζουν σωστά τη συνοχή της κλάσης. Η μελέτη των Woo, G., Chae, H. S., Cui, J. F., & Ji, J.-H. (Woo, Chae, Cui, & Ji, 2009) παρουσιάζει αναθεωρημένες τις εκδοχες των υφιστάμενων πέντε μετρικών συνοχής λαμβάνοντας υπόψη τον αντίκτυπο των αλληλεπιδράσεων εγγραφής μεταξύ των μελών της κλάσης. Μελετά την ποσότητα

αλλαγής βασισμένη σε άλλες μετρικές. Ανέπτυξαν ειδικά εργαλεία για την αυτοματοποίηση των διαδικασιών υπολογισμού τόσο των αρχικών μετρικών συνοχής όσο και των αναθεωρημένων από τον πηγαίο κώδικα. Κανένα από αυτά όμως δεν εμπλέκεται με την άμεση μέτρηση της ποσότητας της αλλαγής. Πραγματοποιήσαν τη μελέτη περίπτωσης με API πηγαίου κώδικα στο JDK 1.3.10. Το JDK είναι ένα πακέτο ανάπτυξης λογισμικού από τη Sun Microsystems που εφαρμόζει το βασικό σύνολο εργαλείων που απαιτούνται για την εγγραφή, τη δοκιμή και τον εντοπισμό σφαλμάτων εφαρμογών και μικροεφαρμογών Java. Χρησιμοποίησαν τον αριθμό των αλλαγμένων γραμμών ως ένδειξη μεταβλητότητας. Με αυτό το πείραμα, προσπαθήσαν να διαπιστώσουν ότι οι αναθεωρημένες μετρικές μπορούν να είναι καλύτεροι δείκτες για την πρόβλεψη της μεταβλητότητας των κλάσεων.

Όλες οι δημοφιλείς γλώσσες προγραμματισμού που χρησιμοποιούνται ευρέως για ανάπτυξη εφαρμογών για κινητά παρέχουν μηχανισμούς χειρισμού σφαλμάτων. Οι εφαρμογές Android, πιο συγκεκριμένα, γράφονται συνήθως σε Java. Η Java περιλαμβάνει έναν μηχανισμό χειρισμού εξαιρέσεων που επιτρέπει στα προγράμματα να σηματοδοτούν την εμφάνιση σφαλμάτων βάζοντας εξαιρέσεις και να χειρίζονται αυτές τις εξαιρέσεις πιάνοντας τις. Όταν μια εφαρμογή εντοπίζει τις εξαιρέσεις από τις οποίες εξαρτάται ή από τις βιβλιοθήκες από τις οποίες εξαρτάται, μπορεί να συνεχίσει τη δραστηριότητά της ή, τουλάχιστον, να αποτύχει με πιο φιλικό τρόπο. Από την άλλη πλευρά, οι εξαιρέσεις που δεν έχουν προβλεφθεί μπορούν να οδηγήσουν σε μια εφαρμογή σε σφάλμα, ειδικά ένα εμφανιστούν στο κύριο thread. Η δημοσίευση των Oliveira, J., Borges, D., Silva, T., Cacho, N., & Castor, F. (Oliveira, Borges, Silva, Cacho, & Castor, 2018) παρουσιάζει μια εμπειρική μελέτη σχετικά με τη σχέση της χρήσης των αφαιρέσεων του Android (abstractions) και των εξαιρέσεων που δεν έχουν προβλεφθεί. Η προσέγγισή τους είναι επικεντρώνεται στην ποσότητα και τη συντήρηση. Ανέλυσαν τις αλλαγές τόσο στον κανονικό πηγαίο κώδικα όσο και στον κώδικα χειρισμού εξαιρέσεων σε 112 εκδόσεις που εξήχθησαν από 16 project λογισμικού που καλύπτουν περισσότερα από 3 εκατομμύρια γραμμές κώδικα. Οι μετρικές αντίκτυπου αλλαγής υπολογίστηκαν χειροκίνητα με τη βοήθεια του εργαλείου DiffMerge. Το DiffMerge εκτελεί μια οπτική σύγκριση μεταξύ δύο φακέλων, δείχνοντας ποια αρχεία υπάρχουν σε έναν μόνο φάκελο, καθώς και ζεύγη αρχείων (δύο αρχεία με το ίδιο όνομα αλλά σε διαφορετικούς φακέλους) που είναι πανομοιότυπα ή διαφορετικά. Για να ποσοτικοποιήσουν τις αλλαγές στον κώδικα χειρισμού εξαιρέσεων, συνέλλεξαν τη μετρική EHChurnedLOC. Αυτή η μετρική υπολογίζεται με βάση τη διαφορά μεταξύ του πηγαίου κώδικα μιας βασικής έκδοσης και του πηγαίου κώδικα μιας επόμενης έκδοσης. Το EHChurnedLOC μετρά τον αριθμό των γραμμών εξαίρεσης κώδικα χειρισμού που προστέθηκαν και άλλαξαν μεταξύ ενός ζεύγους εκδόσεων. Αυτή η μελέτη υπογραμμίζει την ανάγκη για καλύτερα εργαλεία δοκιμών και επαλήθευσης με

έμφαση στον κώδικα χειρισμού εξαιρέσεων και για μια αλλαγή κουλτούρας στην ανάπτυξη Android ή τουλάχιστον στο σχεδιασμό των API της.

Η εξελικτική ανάπτυξη επιτρέπει έγκαιρες και συχνές προσαρμογές σε νέες ή τροποποιημένες απαιτήσεις. Ωστόσο, τέτοιες απρόβλεπτες αλλαγές ενδέχεται να προκαλέσουν δομικές υποβαθμίσεις του λογισμικού, οι οποίες με τη σειρά τους μπορεί να επιταχύνουν τη μείωση της μεταβλητότητας. Στη δημοσίευση των Arisholm, E., & Sjøberg, D. I. K. (Arisholm & Sjøberg, Towards a framework for empirical assessment of changeability decay, 2000) στόχος ήταν η πρόταση μιας μετρικής για την ποσότητα αλλαγής. Ο ορισμός μας για τη μείωση μεταβλητότητας εστιάζεται στην αυξημένη προσπάθεια που απαιτείται για την εφαρμογή αλλαγών. Επίσης ορίσαν τρεις προσεγγίσεις για την αξιολόγηση της μεταβολής της μεταβλητότητας: (1) μέτρηση δομής, (2) μέτρηση πολυπλοκότητας αλλαγής και (3) συγκριτική αξιολόγηση. Η έρευνά τους στοχεύει στην αξιολόγηση και σύγκριση αυτών των προσεγγίσεων προκειμένου να αναπτυχθεί ένα εμπειρικό πλαίσιο αξιολόγησης. Στα άρθρο προτείνουν ένα σύνολο μετρικών πολυπλοκότητας αλλαγών και τις συγκρίνουν με μετρικές δομικών χαρακτηριστικών χρησιμοποιώντας λεπτομερή δεδομένα που συλλέγονται από ένα αντικειμενοστρεφές project. Το προτεινόμενο πλαίσιο αξιολόγησης αξιολογήθηκε σε τέσσερις βιομηχανικές μελέτες περιπτώσεων (Ericsson, Numerica-Taskon, Genera και Braathens) στη Νορβηγία. Η μετρική που αφορά την ποσότητα αλλαγής είναι το change size, δηλαδή ο αριθμός των γραμμών πηγαίου κώδικα (SLOC) που προστέθηκαν ή διαγράφηκαν από την κλάση c για μια δεδομένη αλλαγή στο σύστημα λογισμικού. Τα ευρήματα δείχνουν ότι πολλές πτυχές της μείωσης της μεταβλητότητας δεν μπορούν να ληφθούν υπόψη από τα έμμεσα μέτρα που χρησιμοποιούνται στην προσέγγιση (1) και (2).

Τα περισσότερα αντικειμενοστρεφή συστήματα λογισμικού αναπτύσσονται χρησιμοποιώντας ένα εξελικτικό μοντέλο διαδικασίας. Επομένως, η κατανόηση των φάσεων που έχει περάσει ο λογικός σχεδιασμός (design) του συστήματος και το στυλ της εξέλιξής τους μπορούν να παράσχουν πολύτιμες πληροφορίες για την υποστήριξη της συνεπούς συντήρησης και εξελίξεως του συστήματος, χωρίς να διακυβεύεται η ακεραιότητα και η σταθερότητα της αρχιτεκτονικής του. Στη δημοσίευση των Xing, Z., & Stroulia, E. ({Stroulia}, 2005) παρουσιάζεται μια μέθοδος για την ανάλυση της εξέλιξης των αντικειμενοστρεφών συστημάτων λογισμικού από την άποψη του design. Ο αλγόριθμος εντοπίζει αρκετά βασικά είδη δομικών αλλαγών ενός συστήματος λογισμικού, όπως για παράδειγμα τις προσθήκες, τις διαγραφές, τις μετονομασίες διάφορων συστατικών του (πακέτων, κλάσεων, μεθόδων, ιδιοτήτων), κλπ. Η εκτίμηση των αποτελεσμάτων της μεθοδολογίας πραγματοποιήθηκε σε πολλές γενιές δυο αντικειμενοστρεφών συστημάτων λογισμικού και με βάση τα ευρήματα, η μεθοδολογία εντόπισε όλες τις καταγεγραμμένες περιπτώσεις προβλημάτων των σχεδιαστών καθώς επίσης και μερικές επιπλέον μη καταγεγραμμένες περιπτώσεις. Με βάση τα

αποτελέσματα του UMLDiff, εκτελούνται τρεις αναλύσεις - phase, gamma και βέλτιστη ανάλυση αντιστοίχισης ακολουθίας - για την αναγνώριση ενδιαφέρων προτύπων στο ιστορικό εξέλιξης μεμονωμένων κατηγοριών συστημάτων, ομάδων τάξεων και του συστήματος στο σύνολό του. Το UMLDiff θεωρεί ότι δύο οντότητες του ίδιου τύπου UML σε δύο διαφορετικές εκδόσεις είναι οι «ίδιες» με βάση τις δύο συνιστώσες: Ομοιότητα ονομάτων και Ομοιότητα συσχετίσεων. Όταν μια οντότητα μετονομάζεται ή μετακινείται, οι σχέσεις της με άλλες οντότητες, όπως τα μέλη που περιέχει, τα πεδία που διαβάζει και γράφει, τις μεθόδους που καλεί ή καλείται, κ.λπ., τείνουν να παραμένουν οι ίδιες ως επί το πλείστον. Οι μετρικές βάσει των οποίων εξάγεται το συμπέρασμα για την ποσότητα αλλαγής είναι:

- το πλήθος των μελών που έχουν προστεθεί
- το πλήθος των μελών που έχει διαγραφεί
- το πλήθος των μελών που έχει μετονομαστεί
- το πλήθος των μελών που έχει μετακινηθεί

Στην περίπτωση μελέτης του JFreeChart που μελετάται, ασχολούνται με περισσότερα από 1100 προφίλ εξέλιξης.

Ο εντοπισμός ενοτήτων επιρρεπείς σε αλλαγές μπορούν να επιτρέψουν στους προγραμματιστές λογισμικού να λάβουν εστιασμένες προληπτικές ενέργειες που μπορούν να μειώσουν το κόστος συντήρησης και να βελτιώσουν την ποιότητα. Μερικοί ερευνητές παρατήρησαν μια συσχέτιση μεταξύ της μεταβλητότητας και των δομικών μέτρων, όπως το μέγεθος και η σύζευξη. Τα δομικά χαρακτηριστικά των ενοτήτων λογισμικού, όπως αποτυπώνονται από διάφορα μέτρα, έχουν συσχετιστεί συνήθως με προβλήματα, όπως αλλαγή ή εμφάνιση ελαττωμάτων, προσπάθεια συντήρησης. Στη δημοσίευση των Koru, A. G., & Tian, J. (Koru & Tian, 2005) εντοπίστηκαν και συγκρίθηκαν modules με πολλές αλλαγές και modules με τις υψηλότερες τιμές μέτρησης σε δύο προϊόντα ανοιχτού κώδικα μεγάλης κλίμακας, το Mozilla και το OpenOffice. Στο τέλος της διαδικασίας επιλογής τους, είχαν 51 μετρικές για το Mozilla και 46 μετρικές για το OpenOffice. Για τον υπολογισμό της ποσότητας αλλαγής για κάθε Module, αναπτύξαν ένα set προγραμμάτων PERL. Σε αντίθεση με την αίσθηση που επικρατεί, απέδειξαν μέσω testing ότι τα κορυφαία module στην κατάταξη μετρήσεων αλλαγών και τα module με τις υψηλότερες τιμές μέτρησης ήταν διαφορετικές. Κατέληξαν στο ότι μια θετική συσχέτιση μεταξύ των δομικών μέτρων και της μεταβλητότητας δεν παρέχει αρκετές γνώσεις σχετικά με το πού να εφαρμοστούν δραστηριότητες συντήρησης και διασφάλισης ποιότητας.

Η επόμενη δημοσίευση των Stevanetic, S., & Zdun, U. (Stevanetic & Zdun, 2018) ασχολείται με την κατανόηση των μοντέλων συστατικών (components) που χρησιμοποιούνται συχνά στις αρχιτεκτονικές περιγραφές των συστημάτων λογισμικού. Εξετάζουν εμπειρικά πώς μπορούν να χρησιμοποιηθούν μετρικές διαφορετικών επίπεδων και η εμπειρία των συμμετεχόντων για την πρόβλεψη της

κατανόησης αυτών των μοντέλων. Επιπλέον, αναπτύσσουν ένα εργαλείο εφαρμόζεται ευρήματα στην πράξη. Τα αποτελέσματά μας δείχνουν ότι τα μοντέλα πρόβλεψης έχουν μεγάλη επίδραση, κάτι που σημαίνει ότι η ισχύς πρόβλεψής τους είναι σημαντική. Η εμπειρία των συμμετεχόντων παίζει σημαντικό ρόλο στην πρόβλεψη, αλλά τα λαμβανόμενα μοντέλα δεν είναι τόσο ακριβή όσο τα μοντέλα που χρησιμοποιούν τις μετρικές σε επίπεδο συστατικών στοιχείων (components). Για να δείξουν το εργαλείο τους χρησιμοποιούν το παράδειγμα του συστήματος Frag. Η Frag είναι μια δυναμική γλώσσα προγραμματισμού που εφαρμόζεται στην Java, ειδικά σχεδιασμένη για να είναι μια εξατομικευμένη γλώσσα, για τη δημιουργία Domain-Specific Languages (DSL), για εύκολη ενσωμάτωση στην Java. Ενώ η προσέγγιση της αρχιτεκτονικής που βασίζεται στο DSL επιτρέπει στους αρχιτέκτονες λογισμικού να διατηρούν τον πηγαίο κώδικα και την αρχιτεκτονική συνεκτικά, οι επεκτάσεις των μετρικών τους επιτρέπουν να κρίνουν και να βελτιώνουν συνεχώς την αναλυτικότητα των μοντέλων αρχιτεκτονικών συστατικών που βασίζονται στην κατανόηση των μεμονωμένων στοιχείων. Από ακαδημαϊκή άποψη, η μελέτη μπορεί να χρησιμεύσει ως ένα καλό σημείο εκκίνησης για μελέτες σχετικά με την κατανόηση των αρχιτεκτονικών στοιχείων και των μοντέλων συστατικών. Στο πλαίσιο των αντικειμενοστρεφών συστημάτων λογισμικού, ένα στοιχείο ομαδοποιεί συνήθως ένα σύνολο κλάσεων πηγαίου κώδικα, ενώ ένας σύνδεσμος θα μπορούσε να αντιπροσωπεύει οποιοδήποτε είδος εξάρτησης μεταξύ κλάσεων όπως κλήσεις μεθόδου, πρόσβαση πεδίων κλπ. Εάν από τα συμπεράσματα που κατέληξαν είναι ότι η ποσότητα αλλαγής επηρεάζεται από τον πλήθος των κλάσεων που προστέθηκαν, διαγράφησαν ή αλλάχθηκαν.

Η κλωνοποίηση κώδικα είναι ένας από τους ενεργούς ερευνητικούς τομείς στην κοινότητα του λογισμικού. Συγκεκριμένα, οι ερευνητές έχουν πραγματοποιήσει πολλές εμπειρικές μελέτες σχετικά με την κλωνοποίηση κώδικα και ανέφεραν ότι το 7% έως 23% του κώδικα σε ένα τυπικό σύστημα λογισμικού έχει κλωνοποιήσεις. Ωστόσο, υπάρχει λιγότερη ενημέρωση για τις δυναμικές γλώσσες καθώς οι περισσότερες μελέτες περιορίζονται σε στατικές γλώσσες όπως Java, C και C ++. Επιπλέον, οι περισσότερες προηγούμενες μελέτες δεν έλαβαν υπόψη διαφορετικούς τομείς εφαρμογών, όπως αυτόνομα project ή διαδικτυακές εφαρμογές. Συνεπώς, πολύ λίγα είναι γνωστά για κλώνους σε δυναμικές γλώσσες, όπως η JavaScript. Η δημοσίευση των Cheung, W. T., Ryu, S., & Kim, S. (Cheung, 2016) παρουσιάζει ένα μεγάλης κλίμακας πείραμα ανίχνευσης κλώνων κώδικα σε μια δυναμική γλώσσα προγραμματισμού, τη JavaScript, για διαφορετικούς τομείς εφαρμογών: ιστοσελίδες και μεμονωμένα project. Τα πειραματικά αποτελέσματα έδειξαν ότι σε αντίθεση με τα αυτόνομα project JavaScript, οι ιστοσελίδες JavaScript έχουν 95% κλώνων μεταξύ των αρχείων τους και 91-97% κλώνων ευρέως διασκορπισμένων. Αξιολογήσαν τρεις ανιχνευτές κλώνων σε πέντε αυτόνομα έργα JavaScript: Bootstrap, Web Font Loader, script.aculo.us, Foundation και jQuery. Παρατηρήσαν ότι οι προγραμματιστές web

εφαρμογών δημιούργησαν κλώνους σκόπιμα και τέτοιοι κλώνοι ενδέχεται να μην είναι τόσο επικίνδυνοι όσο ισχυρίζονται προηγούμενες μελέτες. Η κατανόηση των κινδύνων της κλωνοποίησης σε εφαρμογές web απαιτεί περαιτέρω μελέτες, καθώς η κλωνοποίηση μπορεί να οφείλεται είτε σε καλές είτε σε κακές προθέσεις. Η μετρική που επιδρά πραγματικά στην ποσότητα αλλαγής είναι οι γραμμές του πηγαίου κώδικα που αλλάζουν. Επίσης, εντόπισαν μοναδικές πρακτικές ανάπτυξης, όπως η χρήση κώδικα που εξαρτάται από το browser. Αυτό δείχνει ότι τα χαρακτηριστικά των γλωσσών προγραμματισμού και των τεχνολογιών επηρεάζουν τον τρόπο με τον οποίο οι προγραμματιστές αντιγράφουν κώδικα.

Η κατανάλωση ενέργειας γίνεται όλο και πιο σημαντική με την αυξημένη δημοτικότητα των smart-phones, των tablet και των φορητών υπολογιστών. Η απειλή μείωσης της διάρκειας ζωής της μπαταρίας ενός πελάτη κρέμεται τώρα από τον προγραμματιστή που ρωτά, "θα είναι αυτή η επόμενη αλλαγή που θα αναγκάσει το λογισμικό μου να εξαντλήσει την μπαταρία ενός πελάτη;". Μία λύση είναι η μέτρηση κατανάλωσης ενέργειας μετρώντας τη χρήση ισχύος των δοκιμών, αλλά αυτό είναι χρονοβόρο και συχνά θορυβώδες. Μια εναλλακτική λύση είναι να βασιστούμε σε μετρικές που μας επιτρέπουν να εκτιμήσουμε τον αντίκτυπο που μπορεί να έχει μια αλλαγή στην κατανάλωση ενέργειας, απαλλάσσοντας έτσι τον προγραμματιστή από δαπανηρές δοκιμές. Στο άρθρο του Spinellis, D. ({Spinellis}, 2005) παρουσιάζεται μια γενική μεθοδολογία για τη διερεύνηση των επιπτώσεων της αλλαγής λογισμικού στην κατανάλωση ενέργειας, συσχετίζεται η κατανάλωση ενέργειας με τις αλλαγές λογισμικού και, στη συνέχεια, διερευνάται η επίδραση των στατικών μετρικών λογισμικού στην κατανάλωση ενέργειας. Απέδειξε ότι η αλλαγή λογισμικού μπορεί να επηρεάσει την κατανάλωση ενέργειας χρησιμοποιώντας το browser Firefox και το Azureus / Vuze BitTorrent. Βρέθηκαν στοιχεία για πιθανή σχέση μεταξύ ορισμένων μετρήσεων λογισμικού και ποιότητας αλλαγής. Ορισμένα από αυτά είναι τα εξής:

- Added lines
- Removed lines
- Churned Lines
- Total files
- file Churn
- Lines Of Code
- ChurnLines Of Code

Συμπερασματικά, ο Spinnelis D. διερεύνησε την επίδραση της αλλαγής λογισμικού στην κατανάλωση ενέργειας σε δύο project και παρείχε μια αρχική έρευνα σχετικά με τον αντίκτυπο των μετρήσεων λογισμικού στην κατανάλωση ενέργειας.

4.2 Σύνοψη

Ο πίνακας των μετρικών που μπορούν να καθορίσουν άμεσα ή έμμεσα την ποσότητα αλλαγής λογισμικού ακολουθεί:

Πίνακας 3

Ονόματα μετρικών	**Περιγραφή**
Added lines	
Change Metric (CM)	η αλλαγή σε μια μετρικη από τη μία κυκλοφορία στην άλλη
change size	Ο αριθμός των γραμμών κώδικα (SLOC) που προστέθηκαν ή διαγράφηκαν από την κλάση c για μια δεδομένη αλλαγή στο σύστημα λογισμικού
Deleted class	
Added classes	
Changed classes	αλλαγές σε τρία διαφορετικά επίπεδα στο σύστημα: αλλαγές πηγαίου κώδικα, component model changes και αλλαγές DSL
Churned Lines	
class growth	δείχνει την εξέλιξη του λογισμικού
EHChurnedLOC	ο αριθμός των γραμμών του κώδικα χειρισμού εξαιρέσεων που προστέθηκε και άλλαξε μεταξύ ενός ζεύγους εκδόσεων
lines of added code	
lines of deleted code	
minutes	συνολικής προσπάθειας
NAC	Αριθμός προστιθέμενων κλασεων
NALOC	Αριθμός προστιθέμενων γραμμών κώδικα (LOC)
NCC	Αριθμός αλλαγμενων κλασεων
NCLOC	αριθμός των γραμμών κώδικα που τροποποιήθηκαν για να ικανοποιήσουν ένα αίτημα αλλαγής
NSFC	αριθμός των αρχείων πηγαίου κώδικα που τροποποιήθηκαν για να ικανοποιήσουν ένα αίτημα αλλαγής
number of added members	
number of changed lines	
number of changed lines	
number of changed lines	
Number of modified subsystems	
number of moved members	
number of removed members	
number of renamed members	
Removed lines	
SDI	Η μετρικη System Design Instability (SDI) υποδεικνύει την πρόοδο ενός έργου που έχει γίνει αντικειμενοστρεφές (OO) όταν το project ενεργοποιηθεί.
the difference in average class size before and after a change class	
the difference in average number of implemented methods in a class c	

the difference in average number of static method invocations for a class c to non-library classes	
the total churn	
the total number of files	
the total number of subsystems	

5. Δυνατότητα αλλαγής

5.1 Εισαγωγή

Σύμφωνα με το Αναθεωρημένο Λεπτό Λεξικό του Webster, δυνατότητα αλλαγής (changeability) είναι η ποιότητα της ικανότητας αλλαγής. Από αυτόν τον ορισμό, μπορεί κανείς να συμπεράνει δύο σημαντικά συμπεράσματα:

- Η ικανότητα αλλαγής σημαίνει ότι το καθήκον της μετρικής είναι να μειώνει την προσπάθεια αλλαγής του λογισμικού
- Η ικανότητα αλλαγής συνεπάγεται την ικανότητα αποφυγής μιας σταδιακής πτώσης από μια υγιή ή ικανοποιητική κατάσταση σε μια μη ικανοποιητική κατάσταση, στην οποία περίπτωση οι αλλαγές απαιτούν περισσότερη προσπάθεια από ό, τι θα έπρεπε.

5.2 Συλλογή άρθρων

Η αλλαγή λογισμικού είναι θεμελιώδες συστατικό της συντήρησης και της εξέλιξης του λογισμικού. Η αποτελεσματική υποστήριξη της τροποποίησης λογισμικού είναι απαραίτητη για την παροχή αξιοπιστίας και υψηλής ποιότητας εξέλιξης συστημάτων λογισμικού, καθώς μια μικρή αλλαγή μπορεί να προκαλέσει κάποια απρόβλεπτα και ανεπιθύμητα αποτελέσματα σε άλλα μέρη του λογισμικού. Για να αντιμετωπίσει αυτό το ζήτημα, αυτή η εργασία χρησιμοποίησε την ανάλυση μάταιής αλλαγών (change impact analysis (CIA)) για να καθοδηγήσει τις αλλαγές στο λογισμικού. Η CIA μπορεί να χρησιμοποιηθεί για να βοηθήσει στη λήψη σωστής απόφασης σχετικά με την πρόταση αλλαγής, δηλαδή την αξιολόγηση της δεινότητας αλλαγής και για να έχουμε αποτελεσματικές αλλαγές. Στο άρθρο των Sun, X., Leung, H., Li, B., & Li, B. (Xiaobing Sun, 2014) πραγματοποιήθηκε μια εμπειρική μελέτη σε τρία συστήματα ανοιχτού κώδικα Java για να δείξουν πώς μπορεί να χρησιμοποιηθεί η CIA κατά την τροποποίηση του λογισμικού. Οι πιθανές μέθοδοι που επηρεάζονται ταξινομούνται σύμφωνα με τη μετρική *impact factor* που αντιστοιχεί στην προτεραιότητα αυτών των μεθόδων που πρέπει να ελεγχθούν. Στη συνέχεια, χρησιμοποιούν τη μετρικής επίπτωσης (*impactness*) βάσει των αποτελεσμάτων της FCA – CIA για να προσδιορίσουν τη μεταβλητότητα αυτής της πρότασης αλλαγής. Η μέτρηση επιπτώσεων μετράει τον βαθμό που η προτεινόμενη πρόταση αλλαγής μπορεί να επηρεάσει το θεωρητικό σύστημα. Η μελέτη μας δείχνει ότι τα αποτελέσματα των επιπτώσεων που παράγει η CIA είναι ακατάλληλα για την αξιολόγηση της δυνατότητας αλλαγής. Αυτό σημαίνει ότι ορισμένες άλλες μετρικές πρέπει να αναπτυχθούν για την αξιολόγησή της. Οι εμπειρικές μελέτες δείχνουν ότι η προτεινόμενη μετρική επίπτωσης βάσει της CIA είναι αποτελεσματική για την αξιολόγηση της δυνατότητας αλλαγής, η οποία μπορεί να βοηθήσει τους χρήστες να λάβουν σωστή απόφαση σχετικά με την αποδοχή ή την απόρριψη της πρότασης αλλαγής. Με βάση μια μελέτη χρηστών

που περιελάβανε 16 μαθητές για να διορθώσουν τέσσερα σφάλματα στο πρόγραμμα jEdit, από τα αποτελέσματα της χρονικής απόδοσης και την αντίληψη των χρηστών, φαίνεται ότι η CIA μπορεί να κάνει τη διαδικασία υλοποίησης αλλαγών πιο αποδοτική και αποτελεσματική.

Η κατανόηση της επίδρασης μιας αλλαγής καθορίζει την επιτυχία ή την αποτυχία των εξελισσόμενων προϊόντων. Το πρόβλημα μεγεθύνεται για επιχειρήσεις που λειτουργούν με περιορισμένους πόρους. Η συνήθης εστίασή τους είναι στο Ελάχιστο Βιώσιμο Προϊόν (Minimum Viable Product (MVP's)) που παρέχει εξειδικευμένη λειτουργικότητα, επομένως έχει λίγο διαθέσιμο χρόνο για τον χειρισμό αλλαγών. Η Ανάλυση μετάδοσης αλλαγών (CIA) αναφέρεται στον προσδιορισμό των αρχείων πηγαίου κώδικα που επηρεάζονται κατά την εφαρμογή ενός αιτήματος αλλαγής. Στο άρθρο των Kabeer, S. J., Nayebi, M., Ruhe, G., Carlson, C., & Chew, F. (Kabeer, Nayebi, Ruhe, Carlson, & Chew, 2017) επεκτείνουν αυτήν την ερώτηση για να προβλέψουν όχι μόνο τα αρχεία που επηρεάζονται, αλλά και την προσπάθεια που απαιτείται για την εφαρμογή της αλλαγής και τη διάρκεια που απαιτείται για αυτό. Επισης αξιολογεί την απόδοση τριών τεχνικών ομοιότητας κειμένου για τη CIA με βάση το Bag of Word σε συνδυασμό είτε με μοντελοποίηση θεμάτων είτε με σύζευξη αρχείων. Οι προσεγγίσεις εφαρμόζονται σε δεδομένα από δύο βιομηχανικά project. Τα δεδομένα έρχονται ως μέρος ενός project βιομηχανικής συνεργασίας με την Brightsquid, μια καναδική εταιρεία που ειδικεύεται σε ασφαλείς επικοινωνιακές λύσεις. Χρησιμοποιήσαν τη Leave-One-Out Cross Validation (LOOCV) για να αξιολογήσουν την απόδοση του μοντέλου στην πρόβλεψη των επηρεασμένων αρχείων, της προσπάθειας(effort) και της διάρκειας(duration) που σχετίζονται με κάθε αίτημα αλλαγής. Η επέκταση της συλλογιστικής που βασίζεται στην πρόβλεψη όχι μόνο των αρχείων που θα επηρεαστούν , αλλά και τη διάρκεια και την προσπάθεια που απαιτείται για την εφαρμογή του αιτήματος αλλαγής, αυξάνει την αποτελεσματικότητα της μεθόδου και τη βιομηχανική συνάφεια της προσέγγισης. Τα αποτελέσματα έδειξαν ότι η προσπάθεια και η διάρκεια μπορούν να προβλεφθούν με ακρίβεια 84% και 72% χρησιμοποιώντας ομοιότητα κειμένου και μόνο.

Η μελέτη των Curtis, B., Sappidi, J., & Subramanyam, J. (Curtis, Sappidi, & Subramanyam, 2011) συνοψίζει τα αποτελέσματα μιας μελέτης της εσωτερικής, δομικής ποιότητας 288 επιχειρηματικών εφαρμογών που περιλαμβάνει 108 εκατομμύρια γραμμές κώδικα που συλλέχθηκαν από 75 εταιρείες σε 8 τομείς της βιομηχανίας. Αυτές οι αιτήσεις υποβλήθηκαν σε στατική ανάλυση που αξιολογεί την ποιότητα εντός και μεταξύ των στοιχείων της εφαρμογής που ενδέχεται να κωδικοποιούνται σε διαφορετικές γλώσσες. Η ανάλυση συνίσταται στην αξιολόγηση της εφαρμογής σε ένα αποθετήριο άνω των 900 κανόνων ορθής αρχιτεκτονικής και κωδικοποίησης. Τα αποτελέσματα παρουσιάζονται για μετρικές ασφάλειας, απόδοσης και δυνατότητας αλλαγής. Συγκρίναν τις βαθμολογίες μεταβλητότητας ανά κλάδο βιομηχανίας. Τα αποτελέσματα αποκάλυψαν ότι οι βαθμολογίες για εφαρμογές στο δημόσιο τομέα είναι σημαντικά χαμηλότερες από αυτές των άλλων τομέων ($p < 0.01$). Το δείγμα περιελάβανε κυβερνητικές αιτήσεις τόσο από τις ΗΠΑ όσο και από την ΕΕ. Τα χαμηλά ποσοστά μεταβλητότητας στον κυβερνητικό τομέα ενδέχεται να εξηγούνται εν μέρει από το υψηλό ποσοστό των εφαρμογών εξωτερικής ανάθεσης σε σύγκριση με τον ιδιωτικό τομέα.

Τα συστήματα λογισμικού εξελίσσονται συνεχώς για να φιλοξενήσουν νέες δυνατότητες και οι συσχετίσεις μεταξύ αντικειμένων δείχνουν όλο και πιο σημαντικές επιπτώσεις στην αλλαγή λογισμικού. Κατά τη διάρκεια της συντήρησης, οι προγραμματιστές πρέπει να διασφαλίσουν ότι οι σχετικές οντότητες ενημερώνονται ώστε να είναι συνεπείς με αυτές τις αλλαγές. Μελέτες στον τομέα της ανάλυσης μετάδοσης αλλαγών έχουν εντοπίσει ότι ένας συνδυασμός πηγαίου κώδικα και λεξικών πληροφοριών λειτουργεί καλυτέρα από ότι όταν υιοθετείται το καθένα ανεξάρτητα. Ωστόσο, η εξαγωγή λεκτικών πληροφοριών και η μέτρηση του πόσο χαλαρά ή στενά σχετίζονται δύο τεχνουργήματα λογισμικού, λαμβάνοντας υπόψη τις σημασιολογικές πληροφορίες που είναι ενσωματωμένες στα σχόλια και τα αναγνωριστικά τους, πραγματοποιήθηκε χρησιμοποιώντας κάπως περίπλοκες τεχνικές ανάκτησης πληροφοριών (information retrieval (IR)). Η αλληλεπίδραση μεταξύ σημασιολογίας του λογισμικού και των αλλαγών λογισμικού δεν έχει μελετηθεί εκτενώς. Η δημοσίευση των Ajienka, N., Capiluppi, A., & Counsell, S. (Ajienka, Capiluppi, & Counsell, An Empirical Study on the Interplay Between Semantic Coupling and Co-change of Software Classes, 2018) στοχεύει να καλύψει το κενό συγκρίνοντας την αποτελεσματικότητα της μέτρησης της σημασιολογικής σύζευξης των κλάσεων λογισμικού ΟΟ χρησιμοποιώντας (i) απλές τεχνικές βασισμένες σε αναγνωριστικά και (ii) μια συλλογή κειμένου όλων των κλάσεων σε ένα σύστημα λογισμικού. Σχετικά με την αλληλεπίδραση μεταξύ σημασιολογικών και λογικών εξαρτήσεων, σε 79 αντικειμενοστρεφή και ανοιχτού κώδικα προγράμματα λογισμικού δεν μπορέσαν να εντοπίσουν μια γραμμική σχέση μεταξύ τους. Παρόλα αυτά, πάνω από το 70% των κλάσεων που σχετίζονται με τη σημασιολογία συνήθως συν-εξελίσσονται και οι κλάσεις που σχετίζονται με την αλλαγή συνήθως μοιράζονται κάποιο βαθμό σημασιολογικής σύζευξης. Με βάση εμπειρικά αποτελέσματα που προέρχονται από σημαντικό αριθμό project, καταλήγουν στο συμπέρασμα ότι ο προσδιορισμός πιο αποτελεσματικών μεθόδων υπολογισμού σημασιολογικής σύζευξης μπορεί να βοηθήσει στη βελτίωση τεχνικών CIA που ενσωματώνουν πληροφορίες σημασιολογικής σύζευξης.

5.3 Σύνοψη

Ο πίνακας των μετρικών που μπορούν να καθορίσουν άμεσα ή έμμεσα την δυνατότητα αλλαγής του λογισμικού ακολουθεί:

Πίνακας 4

Ονόματα μετρικών	**Περιγραφή**
impact factor	Οι μέθοδοι που πιθανώς επηρεάζονται ταξινομούνται σύμφωνα με μια μετρική που αντιστοιχεί στην προτεραιότητα με την οποία οι μέθοδοι πρέπει να επιθεωρηθούν
Number of file changes	
changeability index	καταγράφει την τάση ενός συστήματος να αλλάζει με την πάροδο του χρόνου
minutes	Λεπτά συνολικής προσπάθειας
impactness Metric	ο βαθμός που μια πρόταση αλλαγής μπορεί να επηρεάσει το αρχικό σύστημα

number of effort	προσπάθεια που απαιτείται για την εφαρμογή της αλλαγής
the difference in average class size before and after a change class	
number of duration	η απαιτούμενη διάρκεια για την εφαρμογή της αλλαγής
the difference in average number of implemented methods in a class c	
the difference in average number of static method invocations for a class c to non-library classes	
the difference in average number of static method invocations for a class c to non-library classes	
the difference in average number of static method invocations for a class c to library classes	
ChangeSize	

6. Συμπεράσματα

Αυτό το κεφάλαιο ανακεφαλαιώνει τις παραμέτρους που εξετάστηκαν σε αυτή την εργασία. Περιλαμβάνει δημογραφικά στατιστικά που απαντούν στα βασικά ερευνητικά ερωτήματα:

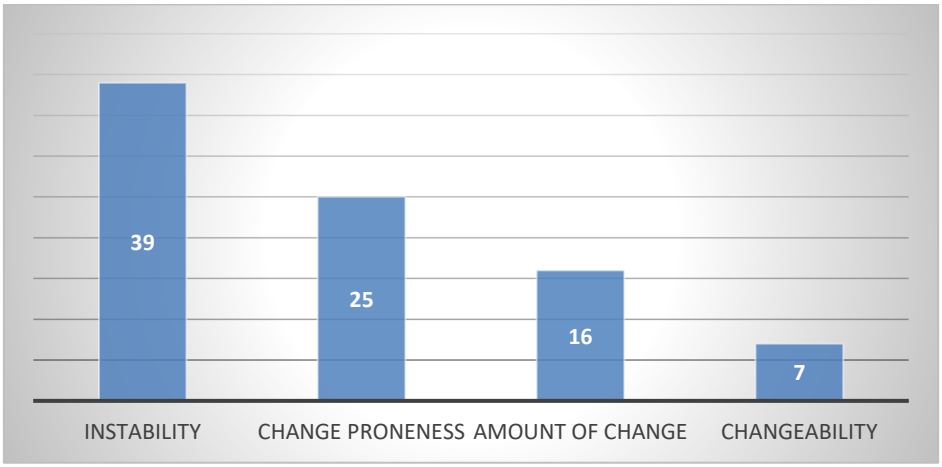
1. Ποια χαρακτηριστικά ποιότητας έχουν μελετηθεί περισσότερο;

2. Σε ποιες φάσεις και μέσω ποιων τεχνουργημάτων (artifacts) έχουν μελετηθεί τα χαρακτηριστικά ποιότητας;

3. Ποιο είναι το επίπεδο εμπειρικής αξιολόγησης για κάθε χαρακτηριστικό ποιότητας;

6.1.Χαρακτηριστικα ποιότητας

Τα χαρακτηριστικά ποιότητας που αναζητήθηκαν αρχικά στα πλαίσια της παρούσας εργασίας ήταν δυο: η αστάθεια και η μεταβλητότητα. Στη συνέχεια προέκυψαν δυο ακόμα που σχετίζονται άμεσα με τα δυο πρώτα: (i) η εναλλαξιμότητα (changeability) και (ii) η ποσότητα αλλαγής (amount of change). Φαίνεται ότι στη βιβλιογραφία πιο δημοφιλές από τα παραπάνω χαρακτηριστικά είναι η αστάθεια του λογισμικού ενώ ακολουθεί η μεταβλητότητα, όπως φαίνεται στο ακόλουθο διάγραμμα:



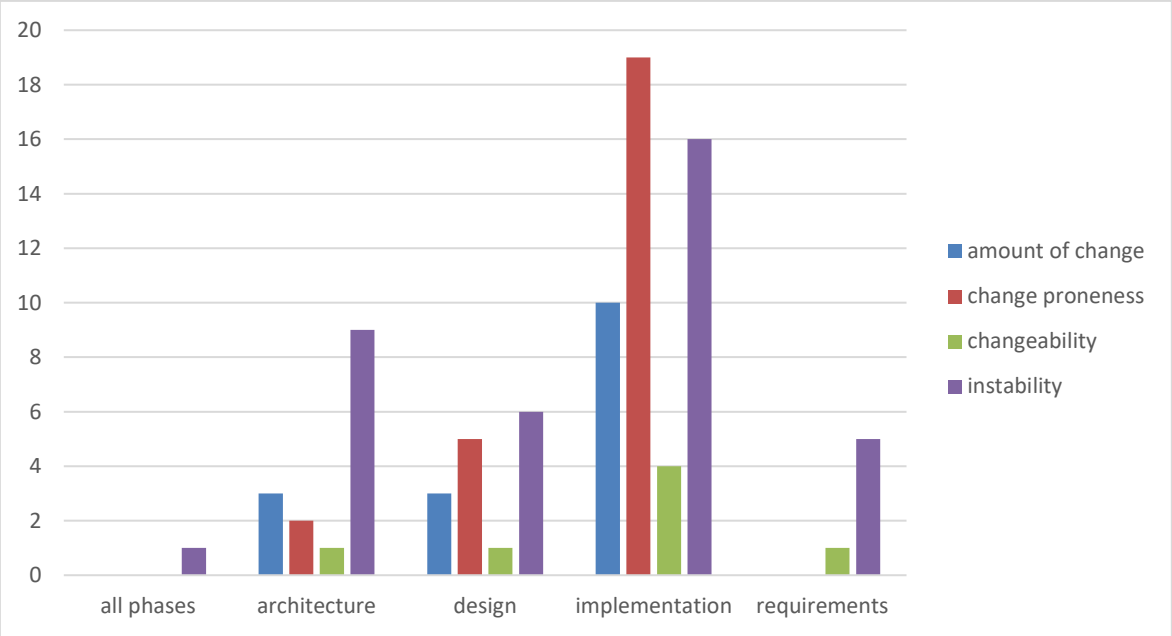
1 Δημοσιεύσεις ανά ποιοτικό χαρακτηριστικό

Ορισμένες δημοσιεύσεις μελετούν περισσότερα από ένα χαρακτηριστικά. Αυτός είναι ο λόγος που το άθροισμα των χαρακτηριστικών είναι μεγαλύτερο από αυτό των δημοσιεύσεων.

Όπως γίνεται αντιληπτό η αστάθεια παρουσιάζει ελαφρώς μεγαλύτερο ενδιαφέρον από τη μεταβλητότητα στην ανάλυση μετάδοσης αλλαγών.

6.2.Φασεις Ανάπτυξης

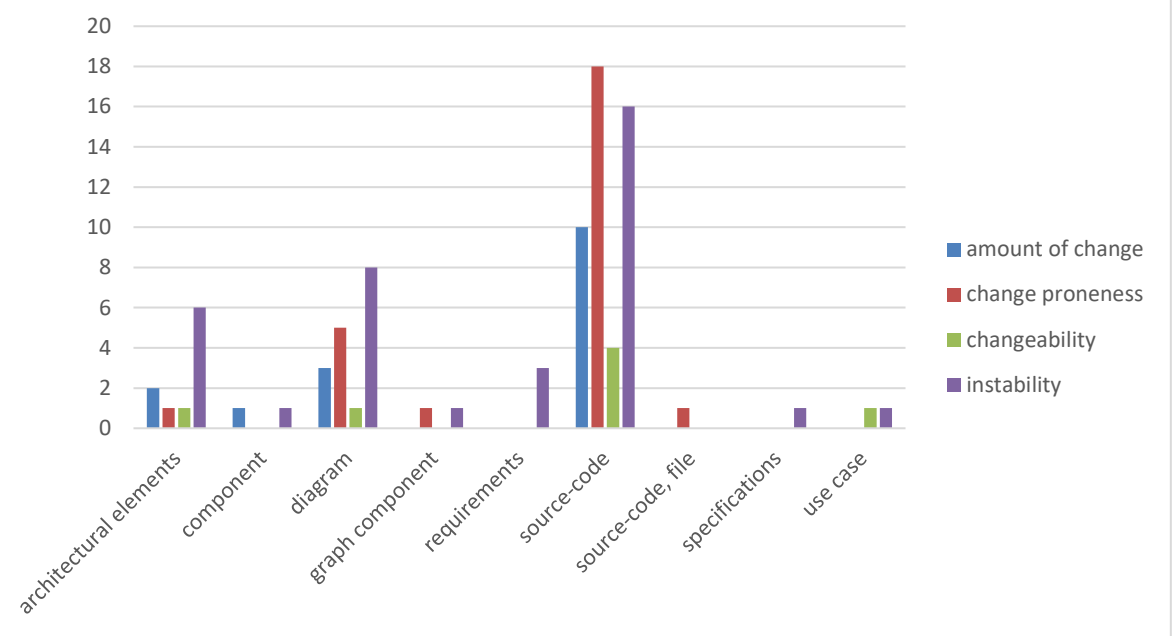
Οι φάσεις ανάπτυξης του λογισμικού δεν παρουσιάζουν το ίδιο ενδιαφέρον για όλα τα ποιοτικά χαρακτηριστικά. Πιο δημοφιλής φάση ανάπτυξης είναι η υλοποίηση (implementation) ενώ σπάνια χρησιμοποιείται η ανάλυση μετάδοσης αλλαγών στη φάση των απαιτήσεων ενός project. Κάτι που ακούγεται λογικό αφού είναι μόλις το πρώτο στάδιο ανάπτυξης.



2 Ποιοτικά χαρακτηριστικά ανά φάση υλοποιήσης

Από τον παραπάνω πίνακα συμπεραίνουμε ότι στη διάρκεια της αρχιτεκτονικής δίνεται προσοχή στην αστάθεια του λογισμικού, όχι όμως και στη μεταβλητότητα. Η μεταβλητότητα παίζει σπουδαίο ρολό στην υλοποίηση ενός project μεν, αλλά σε καμία άλλη φάση δεν είναι η πρώτη σε σημασία.

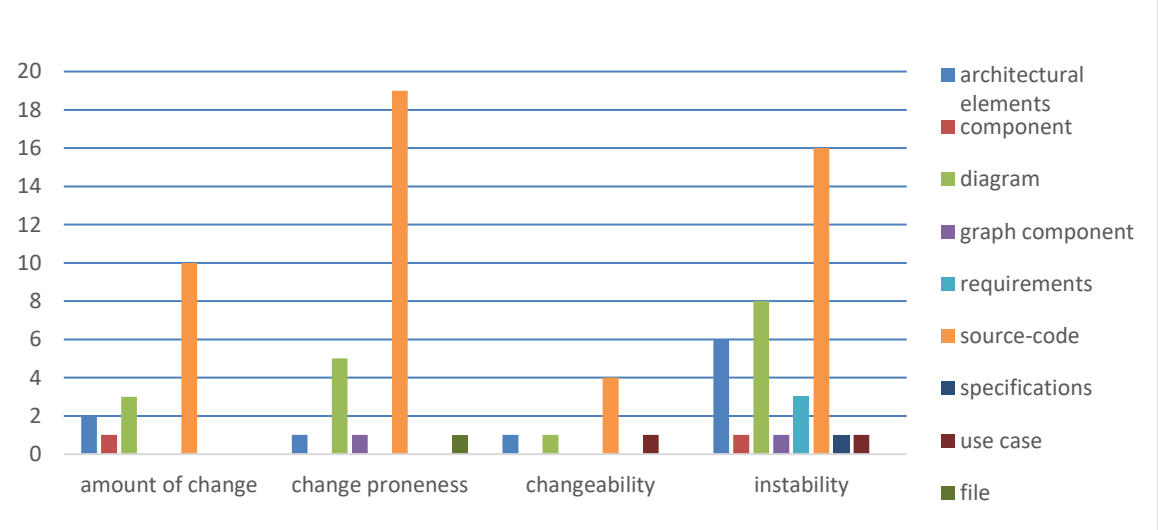
Για τη μελέτη των τεχνουργημάτων (artifacts) προέκυψαν πολλές μοναδικές τιμές. Για να καταλήξουμε σε ένα πιο ξεκάθαρο συμπέρασμα θεωρήσαμε ότι οποιοδήποτε τεχνούργημα σχετιζόταν με διάγραμμα είναι σκέτο διάγραμμα πχ class diagram, sequence diagram, component diagram κλπ.



3 Ποιοτικά χαρακτηριστικά ανά τεχνούργημα

Όπως αναμενόταν, ο πηγαίος κώδικας (source-code) λαμβάνει τη μερίδα του λέοντος σε αυτή τη σύγκριση καθώς είναι το σημείο που προκύπτουν και οι περισσότερες μετρικές. Όσον αφορά στον πηγαίο κώδικα, η εναλλαξιμότητα εμφανίζεται τις λιγότερες φορές στη βιβλιογραφία.

Ακολουθούν και τα τεχνουργήματα ανά χαρακτηριστικό:



4 Τεχνουργήματα ανά φάση υλοποίησης

Η αστάθεια εμφανίζεται σε όλα τα τεχνουργήματα εκτός από το file, το οποίο όμως υπάρχει σε μια μόνο δημοσίευση αρά πρακτικά η αστάθεια μελετάται σε κάθε τεχνούργημα. Δεύτερο σε δημοφιλή τεχνούργημα είναι τα διαγράμματα τα οποία

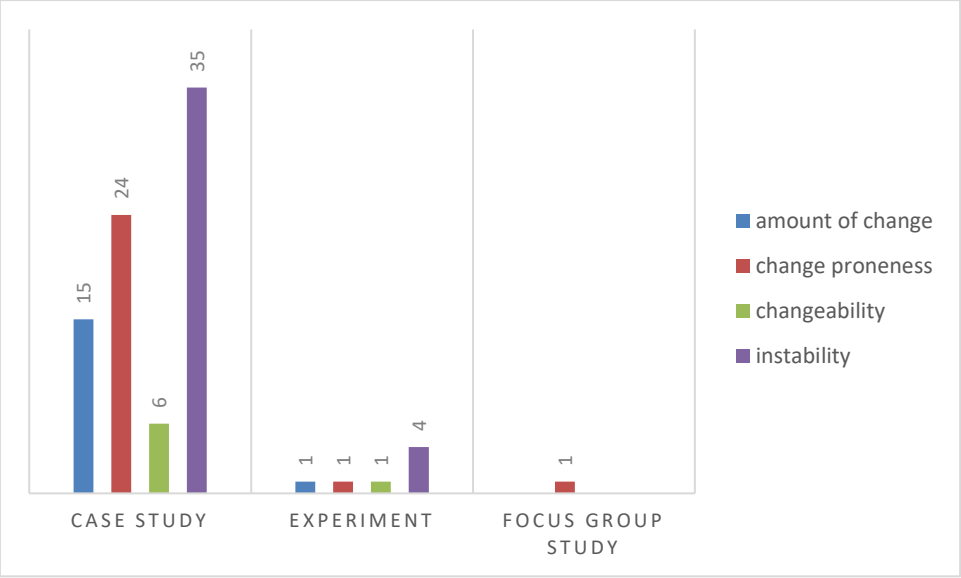
περιλαμβάνουν ένα ευρύτερο πλαίσιο τεχνουργημάτων όπως εξηγήσαμε νωρίτερα.

6.3.Μεθοδοι εμπειρικής αξιολόγησης

Σε αυτό το attribute η κατάσταση είναι πολύ λιγότερο μοιρασμένη. Οι πιθανότητες λιγότερες καθώς μόνο τρεις πιθανότητες εμφανίζονται στη βιβλιογραφία: (i) περίπτωση χρήσης (ii) πείραμα και (iii) ομάδα μελέτης. Όπως φαντάζονται οι περισσότεροι η μελέτη μιας ή περισσοτέρων περιπτώσεων χρήσης είναι μακράν ο δημοφιλέστερος τρόπος. Χρησιμοποιήθηκαν κάθε λογής εφαρμογές από ανοιχτού κώδικα μέχρι πολύπλοκες βιομηχανικές εγκαταστάσεις εκατοντάδων προγραμματιστών. Ο πίνακας που ακολουθεί είναι ξεκάθαρος:

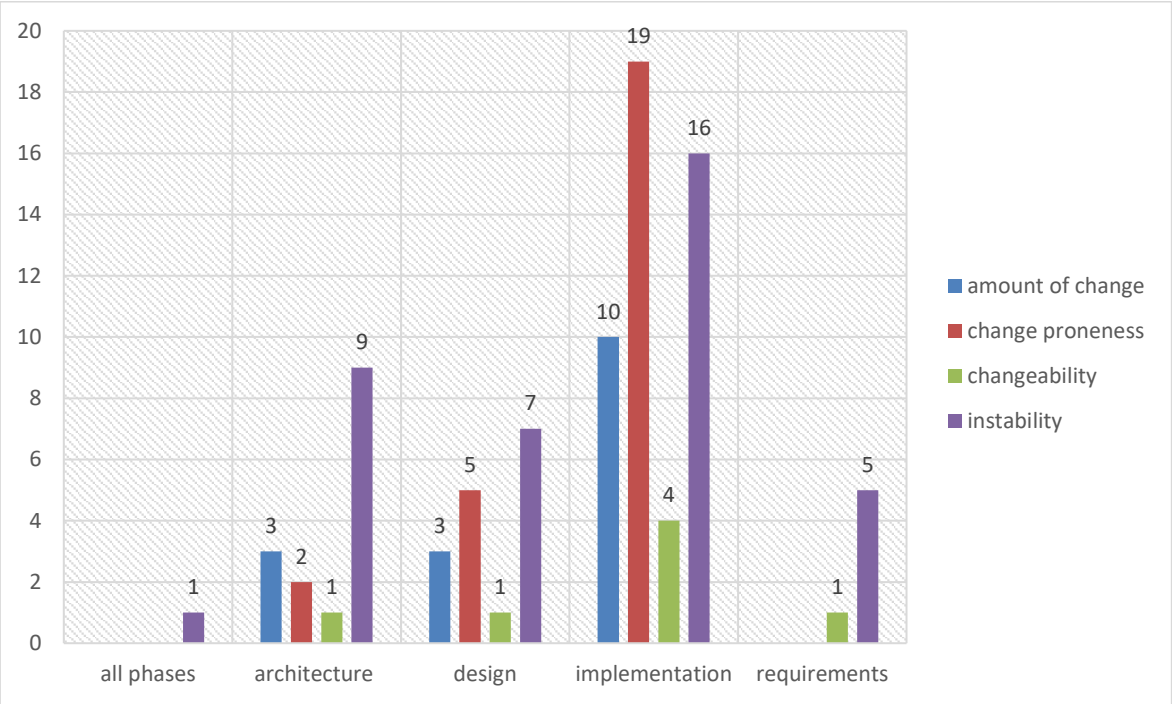
Μέθοδος	Πλήθος
case study	79
experiment	7
focus group study	1
Σύνολο	**87**

Γίνεται ευκολά αντιληπτό ότι αντίστοιχα άνισα μοιρασμένα θα είναι και τα ποιοτικά χαρακτηριστικά:



5 Ποιοτικά χαρακτηριστικά ανά μέθοδο έρευνας

Επιπλέον δημογραφικό χαρακτηριστικό που αξίζει να σημειωθεί είναι τεχνουργήματα που χρησιμοποιήθηκαν ανά φάση:



6 Τεχνούργημα ανά φάση

Παρόλο που η αστάθεια είναι συνολικά δημοφιλέστερη φαίνεται ότι στη φάση της υλοποίησης η μεταβλητότητα έχει προσεγγίσει περισσοτέρους ερευνητές.

Επιπρόσθετα, οι 8 από τις 74 (10.81%) δημοσιεύσεις που μελετήθηκαν ανέπτυξαν ένα ή περισσότερα custom εργαλεία για να προσεγγίσουν το ζήτημα της ανάλυσης μετάδοσης αλλαγών.

6.4 Συνολικά συμπεράσματα

Στην παρούσα εργασία αναλυθήκαν δημοσιεύσεις που αφορούν άμεσα την αστάθεια λογισμικού, τη μεταβλητότητα του, την ποσότητα και τη δυνατότητα αλλαγής.

Οι μετρικές που έχουν σκαρφιστεί οι ερευνητές κατά καιρούς για την αξιολόγηση των παραπάνω ξεπερνούν τη φαντασία του καθενός. Πιο συνηθισμένες είναι οι μετρικές που αφορούν το μέγεθος, οτιδήποτε είναι μετρήσιμο από τις γραμμές των σχολίων και τις γραμμές κώδικα που έχουν αλλαχθεί μέχρι το συνολικό αριθμό αρχείων και τον αριθμό των modules που έχουν διαγραφεί. Πέρα από τις κλασικές μετρικές μεγέθους υπάρχουν πάρα πολλές μετρικές που αφορούν τη σύζευξη. Αφορούν τη σύζευξη όχι μόνο μεταξύ 2 κλάσεων, αλλά μεταξύ όλων των γονικών και κληρονόμων μιας κλάσης, ακόμα και σύζευξη μεταξύ πληροφοριών κειμένου (textual information). Ο ρόλος των προγραμματιστών, των αρχιτεκτόνων και τον ελεγκτών ενός λογισμικού είναι σπουδαίος, όχι μόνο ως προς την εμπειρία τους όπως θα περίμενε κάνεις αλλά ακόμα και οι στόχοι που θέτουν και οι τεχνικές που επιλέγουν καθορίζουν την αστάθεια του προϊόντος. Πολλοί ερευνητές ένιωσαν την

ανάγκη να επεκτείνουν την έρευνα προτείνοντας νέες μετρικές που υπολογίζονται με ολοκληρωμένους τύπους ενώ δοκιμάζονται ενδελεχώς για την αξιοπιστία τους.

Η χρήση μιας ή περισσοτέρων μελετών περίπτωσης είναι η πιο συνηθισμένη μέθοδος για να διεξαχθεί μια επιστημονική ερευνά και σε αυτό τον τομέα. Περιπτώσεις γνωστών projects με χιλιάδες γραμμές κώδικα και πολλές εργατοώρες που έχουν καταναλωθεί από δεκάδες προγραμματιστές. Όσο πιο μεγάλο και πολύπλοκο ένα έργο τόσο περισσότερα χρήσιμα συμπεράσματα μπορούν να εξαχθούν από τη μελέτη του. Η επιλογή ενός έργου με μεγάλη διάρκεια ζωής εμποδίζει τη μεροληψία των αποτελεσμάτων που ενδεχομένως να προκύψουν παρατηρώντας μικρές χρονικές περιόδους. Σε πολλές περιπτώσεις για να δοθεί ακόμα μεγαλύτερη εγκυρότητα στην εργασία τα ερωτήματα αξιολογούνται μέσα από δεκάδες διαφορετικά projects. Από την άλλη, η επιλογή της διεξαγωγής πειραμάτων προϋποθέτει πολλαπλές προσπάθειες και ανάλυση σε βάθος των αποτελεσμάτων που θα προκύψουν σε πολλαπλές διαστάσεις. Κυκλοφορούν διάφορες εκδοχές για την υλοποίηση των πειραμάτων, ενώ η μεθοδολογία περιγράφεται αναλυτικά για κάθε βήμα.

Σε ένα τόσο σημαντικό τομέα όπως η ανάλυση μετάδοσης αλλαγών η ύπαρξη εργαλείων που θα αυτοματοποιούν τη διαδικασία θα μπορούσε να είναι περισσότερο διαδεδομένη καθώς μόλις μια στις δέκα δημοσιεύσεις χρησιμοποιούν κάποιο εργαλείο που να παράγει χρήσιμα σχετικά αποτελέσματα. Η ανάπτυξη τέτοιων εργαλείων θα μπορούσε να παρακινήσει τους προγραμματιστές να ξεκινήσουν να χρησιμοποιούν πιο ένθερμα την ανάλυση μετάδοσης αλλαγών. Ο τομέας των εργαλείων θα μπορούσε να ερευνηθεί περαιτέρω σε μελλοντικές εργασίες.

7. Κίνδυνοι εγκυρότητας

Η εγκυρότητα της βιβλιογραφικής εργασίας περιορίζεται κυρίως από το επιλεγμένο dataset δημοσιεύσεων, τον σχεδιασμό της αναζήτησης και την ανθρώπινη κρίση στην επιλογή μελετών και την εξαγωγή δεδομένων.

Για να μετριάσουμε τις απειλές χρησιμοποιούμε τις πιο δημοφιλείς βιβλιοθήκες στον τομέα της μηχανικής λογισμικού. Στην αρχή της εργασίας αναπτύχθηκε ένα πρωτόκολλο αναζήτησης και επιλογής, αξιολογήθηκε και αναθεωρήθηκε.

Η αναζήτηση έγινε συνδυαστικά από δυο διαφορετικούς ερευνητές. Τα αποτελέσματα συγκρίθηκαν μεταξύ τους, η αναζήτηση σε συγκεκριμένες βιβλιοθήκες επαναλήφθηκε με διαφορετικό τρόπο και καταλήξαμε στο τελικό dataset που εν τελεί εξετάστηκε.

Βιβλιογραφία

{Schneidewind, N. F. (1999). Measuring and evaluating maintenance process using reliability, risk, and test metrics. *IEEE Transactions on Software Engineering*, 769-781.

{Spinellis}, D. (2005). Tool writing: a forgotten art? . *IEEE Software*, 9-11.

{Stroulia}, Z. {. (2005). Analyzing the evolutionary history of the logical design of object-oriented software. *IEEE Transactions on Software Engineering*, 850-868.

{Xu}, Y. {. (2009). Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness. *IEEE Transactions on Software Engineering*, 607-623.

Abdeen, H., Bali, K., Sahraoui, H., & Dufour, B. (2015). Learning dependency-based change impact predictors using independent change histories. *Information and Software Technology, 67*, 220-235. doi:https://doi.org/10.1016/j.infsof.2015.07.007

Agrawal, A., Fu, W., & Menzies, T. (2018). What is wrong with topic modeling? And how to fix it using search-based software engineering. *Information and Software Technology, 98*, 74-88. doi:https://doi.org/10.1016/j.infsof.2018.02.005

Ajienka, N., Capiluppi, A., & Counsell, S. (2018, 6). An empirical study on the interplay between semantic coupling and co-change of software classes. *Empirical Software Engineering, 23*, 1791–1825. doi:10.1007/s10664-017-9569-2

Ajienka, N., Capiluppi, A., & Counsell, S. (2018). An Empirical Study on the Interplay Between Semantic Coupling and Co-change of Software Classes. *Proceedings of the 40th International Conference on Software Engineering* (σσ. 432–432). New York, NY, USA: ACM. doi:10.1145/3180155.3190833

Alégroth, E., Feldt, R., & Ryrholm, L. (2015, 6). Visual GUI testing in practice: challenges, problemsand limitations. *Empirical Software Engineering, 20*, 694–744. doi:10.1007/s10664-013-9293-5

Almugrin, S., Albattah, W., & Melton, A. (2016). Using indirect coupling metrics to predict package maintainability and testability. *Journal of Systems and Software, 121*, 298-310. doi:https://doi.org/10.1016/j.jss.2016.02.024

Alshayeb, M., & Li, W. (2005). An empirical study of system design instability metric and design evolution in an agile software process. *Journal of Systems and Software, 74*, 269-274. doi:https://doi.org/10.1016/j.jss.2004.02.002

Aman, H., Amasaki, S., Sasaki, T., & Kawahara, M. (2015, 10). Empirical Analysis of Change-Proneness in Methods Having Local Variables with Long Names and Comments. *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, (σσ. 1-4). doi:10.1109/ESEM.2015.7321197

Arcangeli, J.-P., Boujbel, R., & Leriche, S. (2015). Automatic deployment of distributed software systems: Definitions and state of the art. *Journal of Systems and Software, 103*, 198-218. doi:https://doi.org/10.1016/j.jss.2015.01.040

Arif, M. M., Shang, W., & Shihab, E. (2018, 6). Empirical study on the discrepancy between performance testing results from virtual and physical environments. *Empirical Software Engineering, 23*, 1490–1518. doi:10.1007/s10664-017-9553-x

Arisholm, E. (2006). Empirical assessment of the impact of structural properties on the changeability of object-oriented software. *Information and Software Technology, 48*, 1046-1055. doi:https://doi.org/10.1016/j.infsof.2006.01.002

Arisholm, E., & Sjøberg, D. I. (2000). Towards a framework for empirical assessment of changeability decay. *Journal of Systems and Software, 53*, 3-14. doi:https://doi.org/10.1016/S0164-1212(00)00003-0

Arisholm, E., Briand, L. C., & Foyen, A. (2004, 8). Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering, 30*, 491-506. doi:10.1109/TSE.2004.41

Arvanitou, E., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2015, 10). Introducing a Ripple Effect Measure: A Theoretical and Empirical Validation. *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, (σσ. 1-10). doi:10.1109/ESEM.2015.7321204

Basili VR, B. L. (1996). A validation of object-oriented design metrics as quality. *IEEE Trans Softw Eng*, 51–761.

Behnamghader, P., Le, D. M., Garcia, J., Link, D., Shahbazian, A., & Medvidovic, N. (2017, 6). A large-scale study of architectural evolution in open-source software systems. *Empirical Software Engineering, 22*, 1146–1193. doi:10.1007/s10664-016-9466-0

Bettenburg, N., Nagappan, M., & Hassan, A. E. (2015, 4). Towards improving statistical modeling of software engineering data: think locally, act globally! *Empirical Software Engineering, 20*, 294–335. doi:10.1007/s10664-013-9292-6

Braunschweig, B., Dhage, N., Viera, M. J., Seaman, C., Sampath, S., & Koru, G. A. (2012). Studying Volatility Predictors in Open Source Software. *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (σσ. 181–190). New York, NY, USA: ACM. doi:10.1145/2372251.2372286

Breech. (2006). Integrating influence mechanisms into impact analysis for. *22nd IEEE international conference on software maintenance*, 55–65.

Capiluppi, A. I.-C. (2013). Effort estimation of FLOSS projects: a study of the Linux kernel. *Empir Software Eng*, 60–88.

Casavant, T. L., & Kuhl, J. G. (1988, 11). Effects of response and stability on scheduling in distributed computing systems. *IEEE Transactions on Software Engineering, 14*, 1578-1588. doi:10.1109/32.9046

Catolino, G., Palomba, F., Lucia, A. D., Ferrucci, F., & Zaidman, A. (2018). Enhancing change prediction models using developer-related factors. *Journal of Systems and Software, 143*, 14-28. doi:https://doi.org/10.1016/j.jss.2018.05.003

Ceccarelli, M., Cerulo, L., Canfora, G., & Penta, M. D. (2010, 5). An eclectic approach for change impact analysis. *2010 ACM/IEEE 32nd International Conference on Software Engineering. 2*, σσ. 163-166. New York, NY, USA: ACM. doi:10.1145/1810295.1810320

Chan, B. J. (2011). ring the model evolution of protocol software specifications by regression testing process improvement. *10th International Conference on Quality Software (QSIC 2010)*.

Chen, Z., Guo, H.-F., & Song, M. (2018). Improving regression test efficiency with an awareness of refactoring changes. *Information and Software Technology, 103*, 174-187. doi:https://doi.org/10.1016/j.infsof.2018.07.003

Cheung, W. R. (2016). Development nature matters: An empirical study of code clones in JavaScript applications. *Empir Software Eng*, 517–564.

Chidamber SR, K. C. (1994). A metrics suite for object oriented design. *IEEE Trans Softw*, 476–493.

Conejero, J. M., Figueiredo, E., Garcia, A., Hernández, J., & Jurado, E. (2012). On the relationship of concern metrics and requirements maintainability. *Information and Software Technology, 54*, 212-238. doi:https://doi.org/10.1016/j.infsof.2011.09.003

Curtis, B., Sappidi, J., & Subramanyam, J. (2011). An Evaluation of the Internal Quality of Business Applications: Does Size Matter? *Proceedings of the 33rd International Conference on Software Engineering* (σσ. 711–715). New York, NY, USA: ACM. doi:10.1145/1985793.1985893

Czibula, I. G., Czibula, G., Miholca, D.-L., & Onet-Marian, Z. (2019). An aggregated coupling measure for the analysis of object-oriented software systems. *Journal of Systems and Software, 148*, 1-20. doi:https://doi.org/10.1016/j.jss.2018.10.052

d'Amorim, F., & Borba, P. (2012). Modularity analysis of use case implementations. *Journal of Systems and Software, 85*, 1012-1027. doi:https://doi.org/10.1016/j.jss.2011.11.1025

Dantas, F., Garcia, A., & Whittle, J. (2012). On the Role of Composition Code Properties on Evolving Programs. *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (σσ. 291–300). New York, NY, USA: ACM. doi:10.1145/2372251.2372304

David Grosser, H. A. (2002). Predicting software stability using case-based reasoning. *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on.*

de la Vara, J. L., Borg, M., Wnuk, K., & Moonen, L. (2016, 12). An Industrial Survey of Safety Evidence Change Impact Analysis Practice. *IEEE Transactions on Software Engineering, 42*, 1095-1117. doi:10.1109/TSE.2016.2553032

Decan, A., Mens, T., & Grosjean, P. (2019, 2). An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering, 24*, 381–416. doi:10.1007/s10664-017-9589-y

Díaz, J., Pérez, J., & Garbajosa, J. (2014). Agile product-line architecting in practice: A case study in smart grids. *Information and Software Technology, 56*, 727-748. doi:https://doi.org/10.1016/j.infsof.2014.01.014

Donzelli, P., & Iazeolla, G. (2001). Hybrid simulation modelling of the software process. *Journal of Systems and Software, 59*, 227-235. doi:https://doi.org/10.1016/S0164-1212(01)00064-4

Eggert, P. R., & Parker, D. S. (2005). Perturbing and evaluating numerical programs without recompilation—the wongldiff way. *Software: Practice and Experience, 35*, 313–322. doi:10.1002/spe.637

Ephremides, A., & Mowafi, O. A. (1982, 1). Analysis of a Hybrid Access Scheme for Buffered Users-Probabilistic Time Division. *IEEE Transactions on Software Engineering, SE-8*, 52-61. doi:10.1109/TSE.1982.234774

Figueiredo, E., Sant'Anna, C., Garcia, A., & Lucena, C. (2012). Applying and evaluating concern-sensitive design heuristics. *Journal of Systems and Software, 85*, 227-243. doi:https://doi.org/10.1016/j.jss.2011.09.060

Gil, Y., & Lalouche, G. (2017, 10). On the correlation between size and metric validity. *Empirical Software Engineering, 22*, 2585–2611. doi:10.1007/s10664-017-9513-5

Goknil, A., Kurtev, I., van den Berg, K., & Spijkerman, W. (2014). Change impact analysis for requirements: A metamodeling approach. *Information and Software Technology, 56*, 950-972. doi:https://doi.org/10.1016/j.infsof.2014.03.002

Hajri, I. (2016). Supporting Change in Product Lines Within the Context of Use Case-driven Development and Testing. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (σσ. 1082–1084). New York, NY, USA: ACM. doi:10.1145/2950290.2983945

Han, A.-R., Jeon, S.-U., Bae, D.-H., & Hong, J.-E. (2010). Measuring behavioral dependency for improving change-proneness prediction in UML-based design models. *Journal of Systems and Software, 83*, 222-234. doi:https://doi.org/10.1016/j.jss.2009.09.038

Herbold, S., Grabowski, J., & Waack, S. (2011, 12). Calculation and optimization of thresholds for sets of software metrics. *Empirical Software Engineering, 16*, 812–841. doi:10.1007/s10664-011-9162-z

Herzig, K. S. (2010). Capturing the long-term impact of changes. *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, 393–396.

Hotomski, S., & Glinz, M. (2019). GuideGen: An approach for keeping requirements and acceptance tests aligned via automatically generated guidance. *Information and Software Technology, 110*, 17-38. doi:https://doi.org/10.1016/j.infsof.2019.01.011

Islam, M. R., Zibran, M. F., & Nagpal, A. (2017, 11). Security Vulnerabilities in Categories of Clones and Non-Cloned Code: An Empirical Study. *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, (σσ. 20-29). doi:10.1109/ESEM.2017.9

Jaafar, F., Guéhéneuc, Y.-G., Hamel, S., Khomh, F., & Zulkernine, M. (2016, 6). Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults. *Empirical Software Engineering, 21*, 896–931. doi:10.1007/s10664-015-9361-0

Jabangwe, R., Börstler, J., Šmite, D., & Wohlin, C. (2015, 6). Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. *Empirical Software Engineering, 20*, 640–693. doi:10.1007/s10664-013-9291-7

Jezek, K., Dietrich, J., & Brada, P. (2015). How Java APIs break – An empirical study. *Information and Software Technology, 65*, 129-146. doi:https://doi.org/10.1016/j.infsof.2015.02.014

Jiang, S., McMillan, C., & Santelices, R. (2017, 4). Do Programmers do Change Impact Analysis in Debugging? *Empirical Software Engineering, 22*, 631–669. doi:10.1007/s10664-016-9441-9

K. El Emam, S. B. (2002). The Optimal Class Size for Object-Oriented Software. *IEEE Trans. Software Eng., vol. 28, no. 5,*, 494-509.

Kabeer, S. J., Nayebi, M., Ruhe, G., Carlson, C., & Chew, F. (2017, 11). Predicting the Vector Impact of Change - An Industrial Case Study at Brightsquid. *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, (σσ. 131-140). doi:10.1109/ESEM.2017.20

Kabinna, S., Bezemer, C.-P., Shang, W., Syer, M. D., & Hassan, A. E. (2018, 2). Examining the stability of logging statements. *Empirical Software Engineering, 23*, 290–333. doi:10.1007/s10664-017-9518-0

Kagdi, H., Gethers, M., & Poshyvanyk, D. (2013, 10). Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering, 18*, 933–969. doi:10.1007/s10664-012-9233-9

Khan, S. U., Niazi, M., & Ahmad, R. (2011). Barriers in the selection of offshore software development outsourcing vendors: An exploratory study using a systematic literature review. *Information and Software Technology, 53*, 693-706. doi:https://doi.org/10.1016/j.infsof.2010.08.003

Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., . . . Pohthong, A. (2017, 4). Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering, 22*, 579–630. doi:10.1007/s10664-016-9437-5

Kocaguneli, E., & Menzies, T. (2013). Software effort models should be assessed via leave-one-out validation. *Journal of Systems and Software, 86*, 1879-1890. doi:https://doi.org/10.1016/j.jss.2013.02.053

Koru, A. G., & Liu, H. (2007). Identifying and characterizing change-prone classes in two large-scale open-source products. *Journal of Systems and Software, 80*, 63-73. doi:https://doi.org/10.1016/j.jss.2006.05.017

Koru, A. G., & Tian, J. (2005, 8). Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products. *IEEE Transactions on Software Engineering, 31*, 625-642. doi:10.1109/TSE.2005.89

Kpodjedo, S. R. (2011). Design evolution metrics for defect prediction in object oriented systems. *Empir Software Eng*, 141–175.

Krishna, R., & Menzies, T. (2018). Bellwethers: A Baseline Method For Transfer Learning. *IEEE Transactions on Software Engineering*, 1-1. doi:10.1109/TSE.2018.2821670

L.C. Briand, J. W. (2000). Exploring the Relationships between Design Measures And Software Quality in Object-Oriented Systems. *J. Systems and Software, vol. 51, no. 3*, 245-273.

Law J, R. G. (2003). Whole program path-based dynamic impact analysis. *Proceedings of the international conference on software engineering. IEEE*, pp 308–318.

Li B, S. X. (2013). A survey of code-based change impact analysis techniques. 613–646.

Li W, H. S. (1993). Object-oriented metrics that predict maintainability. *J Syst Softw*, 111–122.

Li, B., Sun, X., & Keung, J. (2013). FCA–CIA: An approach of using FCA to support cross-level change impact analysis for object oriented Java programs. *Information and Software Technology, 55*, 1437-1449. doi:https://doi.org/10.1016/j.infsof.2013.02.003

Li, W., Etzkorn, L., Davis, C., & Talburt, J. (2000). An empirical study of object-oriented system evolution. *Information and Software Technology, 42*, 373-381. doi:https://doi.org/10.1016/S0950-5849(99)00088-9

Liu, C., Yang, D., Xia, X., Yan, M., & Zhang, X. (2019). A two-phase transfer learning model for cross-project defect prediction. *Information and Software Technology, 107*, 125-136. doi:https://doi.org/10.1016/j.infsof.2018.11.005

Lu, H., Zhou, Y., Xu, B., Leung, H., & Chen, L. (2012, 6). The ability of object-oriented metrics to predict change-proneness: a meta-analysis. *Empirical Software Engineering, 17*, 200–242. doi:10.1007/s10664-011-9170-z

Luo, Q. (2016). Automatic Performance Testing Using Input-sensitive Profiling. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (σσ. 1139–1141). New York, NY, USA: ACM. doi:10.1145/2950290.2983975

Luo, Q. (2016). Input-sensitive Performance Testing. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (σσ. 1085–1087). New York, NY, USA: ACM. doi:10.1145/2950290.2983953

M, L. (1998). Are large C++ classes change-prone?An empirical investigation. *Software Pract Ex*, 1551–1558.

Malhotra, R., & Khanna, M. (2017, 12). An empirical study for software change prediction using imbalanced data. *Empirical Software Engineering, 22*, 2806–2851. doi:10.1007/s10664-016-9488-7

Malhotra, R., & Khanna, M. (2018). Particle swarm optimization-based ensemble learning for software change prediction. *Information and Software Technology, 102*, 65-84. doi:https://doi.org/10.1016/j.infsof.2018.05.007

Mantyla, M. V., Claes, M., & Farooq, U. (2018). Measuring LDA Topic Stability from Clusters of Replicated Runs. *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (σσ. 49:1–49:4). New York, NY, USA: ACM. doi:10.1145/3239235.3267435

Mensah, S., Keung, J., Bosu, M. F., & Bennin, K. E. (2018). Duplex output software effort estimation model with self-guided interpretation. *Information and Software Technology, 94*, 1-13. doi:https://doi.org/10.1016/j.infsof.2017.09.010

Mikael Lindvall, R. T. (2003). An Empirically-Based Process for Software Architecture Evaluation. *Empirical Software Engineering volume*, pages83–108.

Misirli, A. T., Shihab, E., & Kamei, Y. (2016, 4). Studying high impact fix-inducing changes. *Empirical Software Engineering, 21*, 605–641. doi:10.1007/s10664-015-9370-z

Mittas, N., & Angelis, L. (2012, 2). A permutation test based on regression error characteristic curves for software cost estimation models. *Empirical Software Engineering, 17*, 34–61. doi:10.1007/s10664-011-9177-5

Mittas, N., Mamalikidis, I., & Angelis, L. (2015). A framework for comparing multiple cost estimation methods using an automated visualization toolkit. *Information and Software Technology, 57*, 310-328. doi:https://doi.org/10.1016/j.infsof.2014.05.010

Mondal, M., Rahman, M. S., Roy, C. K., & Schneider, K. A. (2018, 4). Is cloned code really stable? *Empirical Software Engineering, 23*, 693–770. doi:10.1007/s10664-017-9528-y

Moonen, L., Di Alesio, S., Binkley, D., & Rolfsnes, T. (2016). Practical Guidelines for Change Recommendation Using Association Rule Mining. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (σσ. 732–743). New York, NY, USA: ACM. doi:10.1145/2970276.2970327

Moonen, L., Rolfsnes, T., Binkley, D., & Di Alesio, S. (2018, 8). What are the effects of history length and age on mining software change impact? *Empirical Software Engineering, 23*, 2362–2397. doi:10.1007/s10664-017-9588-z

Mori, T., Tamura, S., & Kakui, S. (2013, 10). Incremental Estimation of Project Failure Risk with Naive Bayes Classifier. *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, (σσ. 283-286). doi:10.1109/ESEM.2013.40

Moser, R. a. (2008). Analysis of the reliability of a subset of change metrics for defect prediction. *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 309–311.

Myrtveit, I., & Stensrud, E. (2012, 2). Validity and reliability of evaluation procedures in comparative studies of effort prediction models. *Empirical Software Engineering, 17*, 23–33. doi:10.1007/s10664-011-9183-7

Nejati, S., Sabetzadeh, M., Arora, C., Briand, L. C., & Mandoux, F. (2016). Automated Change Impact Analysis Between SysML Models of Requirements and Design. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (σσ. 242–253). New York, NY, USA: ACM. doi:10.1145/2950290.2950293

Okutan, A., & Yıldız, O. T. (2014, 2). Software defect prediction using Bayesian networks. *Empirical Software Engineering, 19*, 154–181. doi:10.1007/s10664-012-9218-8

Oliveira, J., Borges, D., Silva, T., Cacho, N., & Castor, F. (2018). Do android developers neglect error handling? a maintenance-Centric study on the relationship between android abstractions and uncaught exceptions. *Journal of Systems and Software, 136*, 1-18. doi:https://doi.org/10.1016/j.jss.2017.10.032

Omerovic, A., Solhaug, B., & Stølen, K. (2012). Assessing practical usefulness and performance of the PREDIQT method: An industrial case study. *Information and Software Technology, 54*, 1377-1395. doi:https://doi.org/10.1016/j.infsof.2012.07.006

Phannachitta, P., Keung, J., Monden, A., & Matsumoto, K. (2017, 2). A stability assessment of solution adaptation techniques for analogy-based software effort estimation. *Empirical Software Engineering, 22*, 474–504. doi:10.1007/s10664-016-9434-8

Porter, A. A., & Selby, R. W. (1990). Evaluating techniques for generating metric-based classification trees. *Journal of Systems and Software, 12*, 209-218. doi:https://doi.org/10.1016/0164-1212(90)90041-J

Poshyvanyk, D., Marcus, A., Ferenc, R., & Gyimóthy, T. (2009, 2). Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering, 14*, 5–32. doi:10.1007/s10664-008-9088-2

Ramírez, A., Romero, J. R., & Ventura, S. (2016, 12). A comparative study of many-objective evolutionary algorithms for the discovery of software architectures. *Empirical Software Engineering, 21*, 2546–2600. doi:10.1007/s10664-015-9399-z

Revelle, M., Gethers, M., & Poshyvanyk, D. (2011, 12). Using structural and textual information to capture feature coupling in object-oriented software. *Empirical Software Engineering, 16*, 773–811. doi:10.1007/s10664-011-9159-7

Riņķevičs, K., & Torkar, R. (2013). Equality in cumulative voting: A systematic review with an improvement proposal. *Information and Software Technology, 55*, 267-287. doi:https://doi.org/10.1016/j.infsof.2012.08.004

Robert L. Nord, I. O. (2014). Architectural dependency analysis to understand rework costs for safety-critical systems. *ICSE Companion 2014: Companion Proceedings of the 36th International Conference on Software Engineering*, 185–194.

Rolfsnes, T., Moonen, L., Alesio, S. D., Behjati, R., & Binkley, D. (2018, 4). Aggregating Association Rules to Improve Change Recommendation. *Empirical Software Engineering, 23*, 987–1035. doi:10.1007/s10664-017-9560-y

Sabané, A., Guéhéneuc, Y.-G., Arnaoudova, V., & Antoniol, G. (2017, 10). Fragile base-class problem, problem? *Empirical Software Engineering, 22*, 2612–2657. doi:10.1007/s10664-016-9448-2

Saini, V., Sajnani, H., & Lopes, C. (2018, 8). Cloned and non-cloned Java methods: a comparative study. *Empirical Software Engineering, 23*, 2232–2278. doi:10.1007/s10664-017-9572-7

Schneidewind, N. F. (1998). How To Evaluate Legacy System Maintenance. *IEEE Software*.

Seo, Y.-S., & Bae, D.-H. (2013, 8). On the value of outlier elimination on software effort estimation research. *Empirical Software Engineering, 18*, 659–698. doi:10.1007/s10664-012-9207-y

Shihab, E., Bird, C., & Zimmermann, T. (2012). The Effect of Branching Strategies on Software Quality. *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (σσ. 301–310). New York, NY, USA: ACM. doi:10.1145/2372251.2372305

Shin, Y., & Williams, L. (2013, 2). Can traditional fault prediction models be used for vulnerability prediction? *Empirical Software Engineering, 18*, 25–59. doi:10.1007/s10664-011-9190-8

Shiri, M. S., Hassine, J. H., & Rilling, J. (2007). Feature Interaction Analysis: A Maintenance Perspective. *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering* (σσ. 437–440). New York, NY, USA: ACM. doi:10.1145/1321631.1321703

Stevanetic, S., & Zdun, U. (2018, 12). Supporting the analyzability of architectural component models - empirical findings and tool support. *Empirical Software Engineering, 23*, 3578–3625. doi:10.1007/s10664-017-9583-4

Sugiki, A., Kono, K., & Iwasaki, H. (2008). Tuning mechanisms for two major parameters of Apache web servers. *Software: Practice and Experience, 38*, 1215–1240. doi:10.1002/spe.861

Sun, X., & Li, B. (2011, 11). Using Formal Concept Analysis to Support Change Analysis. *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering* (σσ. 641–645). Washington: IEEE Computer Society. doi:10.1109/ASE.2011.6100146

Sun, X., Leung, H., Li, B., & Li, B. (2014). Change impact analysis and changeability assessment for a change proposal: An empirical study ☆☆. *Journal of Systems and Software, 96*, 51-60. doi:https://doi.org/10.1016/j.jss.2014.05.036

Sun, X., Li, B., Leung, H., Li, B., & Li, Y. (2015). MSR4SM: Using topic models to effectively mining software repositories for software maintenance tasks. *Information and Software Technology, 66*, 1-12. doi:https://doi.org/10.1016/j.infsof.2015.05.003

Tang, A., Nicholson, A., Jin, Y., & Han, J. (2007). Using Bayesian belief networks for change impact analysis in architecture design. *Journal of Systems and Software, 80*, 127-148. doi:https://doi.org/10.1016/j.jss.2006.04.004

Tizzei, L. P., Dias, M., Rubira, C. M., Garcia, A., & Lee, J. (2011). Components meet aspects: Assessing design stability of a software product line. *Information and Software Technology, 53*, 121-136. doi:https://doi.org/10.1016/j.infsof.2010.08.007

Turhan, B. (2012, 2). On the dataset shift problem in software engineering prediction models. *Empirical Software Engineering, 17*, 62–74. doi:10.1007/s10664-011-9182-8

Vale, T., & de Almeida, E. S. (2018, 11). Experimenting with information retrieval methods in the recovery of feature-code SPL traces. *Empirical Software Engineering*. doi:10.1007/s10664-018-9652-3

Vokáč, M., Tichy, W., SjØberg, D. I., Arisholm, E., & Aldrin, M. (2004, 9). A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns—A Replication in a Real Programming Environment. *Empirical Software Engineering, 9*, 149–195. doi:10.1023/B:EMSE.0000027778.69251.1f

Watson, D., Malan, G. R., & Jahanian, F. (2004). An extensible probe architecture for network protocol performance measurement. *Software: Practice and Experience, 34*, 47–67. doi:10.1002/spe.557

Wermelinger, M., Yu, Y., Lozano, A., & Capiluppi, A. (2011, 10). Assessing architectural evolution: a case study. *Empirical Software Engineering, 16*, 623–666. doi:10.1007/s10664-011-9164-x

Wiese, I. S., Ré, R., Steinmacher, I., Kuroda, R. T., Oliva, G. A., Treude, C., & Gerosa, M. A. (2017). Using contextual information to predict co-changes. *Journal of Systems and Software, 128*, 220-235. doi:https://doi.org/10.1016/j.jss.2016.07.016

Wnuk, K., Gorschek, T., & Zahda, S. (2013). Obsolete software requirements. *Information and Software Technology, 55*, 921-940. doi:https://doi.org/10.1016/j.infsof.2012.12.001

Woo, G., Chae, H. S., Cui, J. F., & Ji, J.-H. (2009). Revising cohesion measures by considering the impact of write interactions between class members. *Information and Software Technology, 51*, 405-417. doi:https://doi.org/10.1016/j.infsof.2008.05.014

Xiaobing Sun, H. L. (2014). Change impact analysis and changeability assessment for a change proposal: An empirical study. *Journal of Systems and Software*, 51-60.

Zhang, H., Li, J., Zhu, L., Jeffery, R., Liu, Y., Wang, Q., & Li, M. (2014). Investigating dependencies in software requirements for change propagation analysis. *Information and Software Technology, 56*, 40-53. doi:https://doi.org/10.1016/j.infsof.2013.07.001

Zhao, Y., Yang, Y., Lu, H., Zhou, Y., Song, Q., & Xu, B. (2015). An empirical analysis of package-modularization metrics: Implications for software fault-proneness. *Information and Software Technology, 57*, 186-203. doi:https://doi.org/10.1016/j.infsof.2014.09.006

Zhou, Y., Leung, H., & Xu, B. (2009, 9). Examining the Potentially Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness. *IEEE Transactions on Software Engineering, 35*, 607-623. doi:10.1109/TSE.2009.32

Zimmermann T, Z. A. (2005). Mining version histories to guide software changes. *IEEE Trans Softw Eng*, 429–445.