ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΚΑΙ ΚΑΤΑΓΡΑΦΗ ΤΩΝ ΠΛΕΟΝΕΚΤΗΜΑΤΩΝ ΚΑΙ ΜΕΙΟΝΕΚΤΗΜΑΤΩΝ ΤΩΝ APIs ΕΝΑΝΤΙ ΑΛΛΩΝ ΒΙΒΛΙΟΘΗΚΩΝ ΛΟΓΙΣΜΙΚΟΥ»

Της φοιτήτριας

Τσιτσικάου Αριστέας

Αρ. Μητρώου: 05/2774

Επιβλέπων Καθηγητής

Δεληγιάννης Ιγνάτιος

Θεσσαλονίκη 2012

Πρόλογος

Η εξέλιξη του λογισμικού σε μεγάλο βαθμό εξαρτάται από τη χρήση των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών (APIs), τα πλαίσια και τα επαναχρησιμοποιήσιμα στοιχεία. Η διαδικασία της δημιουργίας του λογισμικού έχει αλλάξει σημαντικά. Αντί της δημιουργίας λειτουργικότητας από το μηδέν, ένα μεγάλο μέρος της ανάπτυξης λογισμικού είναι η ενσωμάτωση των χαρακτηριστικών γράφοντας τον κώδικα που διασυνδέεται με το API.

Η αποτελεσματικότητα του API είναι σημαντικό να εξασφαλιστεί προκειμένου να υπάρξει καλύτερη χρήση των πλαισίων και των βιβλιοθηκών. Επίσης η χρηστικότητα των APIs απαιτεί όλο και μεγαλύτερο ενδιαφέρον στις μέρες μας από ό,τι στο παρελθόν. Συνήθως, ένα API μπορεί να θεωρηθεί χρηστικό, όταν παρέχει σωστά την επιθυμητή λειτουργικότητα και αποτελεσματικότητα όσον αφορά τις επιδόσεις. Οι προγραμματιστές αναπτύσσουν ένα API λαμβάνοντας υπόψη τα κριτήρια του σχεδιασμού, όπως είναι η δυνατότητα επαναχρησιμοποίησης, την επανασχεδίαση, την τεκμηρίωση καθώς επίσης και ποιοτικά χαρακτηριστικά του λογισμικού όπως είναι η συντηρησιμότητα, η επεκτασιμότητα, η αποδοτικότητα, η ευελιξία κ.α.

Η διεξαγωγή της παρούσας μελέτης αποδείχθηκε ιδιαίτερα χρήσιμη για την εξοικείωση του συγγραφέα με προηγμένα θέματα τεχνολογίας λογισμικού, τις μεθόδους αξιολόγησης μιας μελέτης και τη συστηματική βιβλιογραφική ανασκόπηση ενός ερευνητικού πεδίου. Επιπλέον, τα αποτελέσματα της εργασίας θεωρούνται σημαντικά και ευρύτερου επιστημονικού ενδιαφέροντος, από τη στιγμή που αξιολογούν τη χρήση των διεπαφών προγραμματισμού εφαρμογών στην ποιότητα του λογισμικού, ζήτημα επίκαιρο στην κοινωνία ανάπτυξης λογισμικού.

Περίληψη

Ο όρος «διασύνδεση προγραμματισμού εφαρμογών» έχει πολύ μεγάλη σημασία για την ανάπτυξη λογισμικού, και εξακολουθεί να αποτελεί ένα σημαντικό πεδίο έρευνας μέχρι σήμερα. Δίνοντας τον ορισμό του API εννοούμε μια καλά καθορισμένη διεπαφή, η οποία διευκολύνει την ανταλλαγή μηνυμάτων ή δεδομένων μεταξύ δύο ή περισσότερων διαφορετικών εφαρμογών λογισμικού.

Το αντικείμενο μελέτης αυτής της εργασίας είναι η συγκριτική μελέτη και καταγραφή των πλεονεκτημάτων και μειονεκτημάτων των διασυνδέσεων προγραμματισμού εφαρμογών.

Αρχικά διεξαγάγαμε μια συστηματική ανασκόπηση της βιβλιογραφίας, προκειμένου να διαπιστώσουμε και να καταγράψουμε την ερευνητική δραστηριότητα μέχρι σήμερα, στον τομέα των APIs. Για τις ανάγκες της ανασκόπησης, συλλέξαμε και μελετήσαμε έναν ικανοποιητικό αριθμό άρθρων, που είχαν δημοσιευθεί σε κορυφαία περιοδικά, συνέδρια και συναντήσεις εργασίας (workshops). Αναλύοντας τα δεδομένα που συγκεντρώσαμε από τη βιβλιογραφία, με χρήση στατιστικών τεχνικών, προέκυψαν ενδιαφέροντα αποτελέσματα. Η μελέτη παρουσιάζεται αναλυτικά στο 2ο κεφάλαιο.

Στη συνέχεια, παρουσιάζουμε μια μελέτη σχετικά με τα πλεονεκτήματα και τα μειονεκτήματα των APIs. Για τη μελέτη αυτή συλλέξαμε 47 άρθρα, όπου συγκεντρώθηκαν και αναλύθηκαν τα δεδομένα και είχε ως συνέπεια, να καταγραφούν σημαντικά αποτελέσματα που παρουσιάζονται αναλυτικά στο 3ο κεφάλαιο.

Abstract

The term “application programming interface” is of great importance for software development and is still a significant field for research. By defining the API we mean a well-defined interface that facilitates exchange of message or data between two or more different software applications.

This degree project focuses on a comparative study of APIs while recording their advantages and disadvantages.

To begin with, a systematic review of the important literature is conducted in order to determine and document the research background until this day concerning the APIs. For the review purposes, a sufficient number of articles were collected and examined, articles that were published in top journals, conferences and workshops. Analyzing the collected data using statistical techniques some interesting results emerged. The study is presented in detail in the 2nd chapter.

Furthermore, an analysis of the advantages and disadvantages of APIs is presented. For this study we selected 47 articles where their data was collected and analyzed. This analysis is presented in detail in the 3rd chapter.

Ευχαριστίες

Πρωτίστως θα ήθελα να ευχαριστήσω όλους του καθηγητές του Τμήματος Πληροφορικής, του Αλεξάνδρειου Τεχνολογικού Εκπαιδευτικού Ιδρύματος Θεσσαλονίκης που προσπάθησαν να μεταλαμπαδεύσουν τις γνώσεις τους σε εμάς τους φοιτητές. Θα ήθελα να ευχαριστήσω θερμά, τον επιβλέποντα καθηγητή της πτυχιακής μου, τον κύριο Δεληγιάννη Ιγνάτιο, που μου έδωσε την ευκαιρία να εργαστώ στον τομέα ανάλυσης λογισμικού μέσω της συγκεκριμένης πτυχιακής εργασίας. Ιδιαίτερα θα ήθελα να ευχαριστήσω τον εργαστηριακό συνεργάτη του τμήματός μας, τον κύριο Αμπατζόγλου Απόστολο που καθ'όλη τη διάρκεια εκπόνησης της πτυχιακής μου εργασίας βρισκόταν δίπλα μου, σαν συνεργάτης και δάσκαλος.

Τέλος θα ήθελα να ευχαριστήσω την οικογένειά μου και τους κοντινούς μου φίλους, που τους δύσκολους αυτούς μήνες εκπόνησης της πτυχιακής μου εργασίας, στάθηκαν δίπλα μου υπομονετικά δείχνοντας άπειρη κατανόηση και προθυμία να με στηρίξουν με οποιονδήποτε τρόπο.

Ευρετήριο Περιεχομένων

Ευρετήριο Πινάκων – Σχημάτων

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

Στο πρώτο κεφάλαιο γίνεται ανασκόπηση και ανάλυση σε δυο βασικούς όρους αυτής της εργασίας. Οι όροι αυτοί είναι το «Ανοιχτό Λογισμικό» καθώς και οι «Διασυνδέσεις Προγραμματισμού Εφαρμογών (*APIs)*» από τους οποίους αντλούνται όλα τα δεδομένα προς ανάλυση και σύγκριση στη συγκεκριμένη πτυχιακή εργασία. Επιπλέον μελετάμε την μεθοδολογία της συστηματικής ανασκόπησης της βιβλιογραφίας, καθώς επίσης και τα πλεονεκτήματα και μειονεκτήματα των APIs που προκύπτουν από την μελέτη μας.

1.1 ΑΝΟΙΧΤΟ ΛΟΓΙΣΜΙΚΟ

Το ανοιχτό λογισμικό αποτελεί ένα σύγχρονο μοντέλο ανάπτυξης, διάθεσης και χρήσης λογισμικού που κερδίζει συνεχώς έδαφος διεθνώς. Ανοιχτό είναι το λογισμικό που ο καθένας μπορεί ελεύθερα να χρησιμοποιεί, να αντιγράφει, να διανέμει και να τροποποιεί ανάλογα με τις ανάγκες του. Η απήχηση του ανοιχτού λογισμικού γίνεται περισσότερο αντιληπτή όταν διαπιστώνουμε πώς εταιρείες λογισμικού προσαρμόζουν το επιχειρηματικό τους μοντέλο στα νέα δεδομένα που αυτό δημιουργεί: το λογισμικό τείνει να αντιμετωπίζεται ως το όχημα παροχής υπηρεσιών και υποστήριξης λύσεων σύμφωνα με τις ανάγκες των χρηστών. Πρόκειται δηλαδή για ένα εναλλακτικό μοντέλο ανάπτυξης και χρήσης λογισμικού, το οποίο βασίζεται στην ελεύθερη διάθεση του πηγαίου κώδικα (σειρά εντολών που γράφονται σε γλώσσα προγραμματισμού υπολογιστών), το οποίο παρέχει τη δυνατότητα αλλαγών ή βελτιώσεων ώστε να καλύπτονται οι απαιτήσεις αυτού που το χρησιμοποιεί.

Η ανάπτυξη μιας εφαρμογής ανοιχτού λογισμικού βασίζεται στη συνεργασία. Ένας μόνος προγραμματιστής, ή μία ομάδα προγραμματιστών ξεκινάει μια εφαρμογή και ανακοινώνει, μέσω του διαδικτύου, μια έκδοση που διατίθεται ελεύθερα, τόσο για χρήση όσο και για τροποποίηση. Έπειτα, η κοινωνία ανοιχτού λογισμικού επεκτείνει και συντηρεί την εφαρμογή. Αυτός ο τρόπος ανάπτυξης έχει και πλεονεκτήματα και μειονεκτήματα. Ένα μειονέκτημα της ανάπτυξης λογισμικού ανοιχτού κώδικα είναι η έλλειψη τεκμηρίωσης και τεχνικής υποστήριξης. Μερικά βασικά πλεονεκτήματα του ανοιχτού λογισμικού είναι το χαμηλό κόστος, η αξιοπιστία και το γεγονός ότι παρέχει τον πηγαίο κώδικά των εφαρμογών στους χρήστες ώστε να μπορούν να προσαρμόσουν το λογισμικό σύμφωνα με τις δικές

τους ανάγκες. Το λογισμικό ανοιχτού κώδικα παρέχει μεγάλες δυνατότητες επαναχρησιμοποίησης, από την άποψη ότι διατίθεται ελεύθερα στους προγραμματιστές παρέχει καλύτερη αξιοποίηση υλικού, δεν απαιτεί συνεχείς αναβαθμίσεις των υπολογιστών για κάθε νεότερη έκδοση του λογισμικού και είναι ασφαλές, γιατί οποιοσδήποτε μπορεί να ελέγξει τις λειτουργίες όλων των υπο-προγραμμάτων. Ένα τμήμα κώδικα προκειμένου να μπορεί να χρησιμοποιηθεί εύκολα και επιτυχώς από μια άλλη εφαρμογή πρέπει να είναι κατανοητό, ευκολοσυντήρητο και ευέλικτο.

1.2 ΔΙΑΣΥΝΔΕΣΕΙΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΕΦΑΡΜΟΓΩΝ (APIs)

Για να καταλάβουμε τι σημαίνει Διασύνδεση Προγραμματισμού Εφαρμογών (αγγλ. API, από το Application Programming Interface) ας μείνουμε στην λέξη διεπαφή. Διεπαφή είναι το κοινό σύνορο μεταξύ δύο χωριστών συστημάτων. Είναι το μέσο μέσω του οποίου αυτά τα δύο συστήματα επικοινωνούν. Το API λοιπόν είναι ένας πηγαίος κώδικας που προορίζεται να χρησιμοποιηθεί ως διεπαφή από στοιχεία λογισμικού για την μεταξύ τους επικοινωνία. Υλοποιείται από εταιρείες για μία “εφαρμογή”, και επιτρέπει σε άλλες “εφαρμογές” να επικοινωνούν με αυτήν.

Ένα API είναι το σύνολο των κλάσεων, οι προδιαγραφές για τις ρουτίνες, οι δομές δεδομένων, τα αντικείμενα των κλάσεων, οι μεταβλητές, οι υπογραφές των μεθόδων, τα αρχεία και τα περιεχόμενά τους, όπου μπορούν να προσπελαστούν από τον κώδικα του πελάτη. Μια προδιαγραφή API μπορεί να λαμβάνει πολλές μορφές, συμπεριλαμβάνοντας ένα διεθνές πρότυπο, όπως το POSIX, την τεκμηρίωση και τις βιβλιοθήκες μιας γλώσσας προγραμματισμού. Ένα API μπορεί να περιγράψει τους τρόπους με τους οποίους εκτελείται μια εργασία. Ένα API μπορεί να είναι:

- language - dependent (εξαρτώμενο από τη γλώσσα), που σημαίνει ότι είναι διαθέσιμο μόνο με χρήση της σύνταξης και των στοιχείων μιας συγκεκριμένης προγραμματιστικής γλώσσας, με την οποία το API έχει υλοποιηθεί.
- language - independent (ανεξάρτητο από τη γλώσσα), οι εντολές μπορεί να είναι γραμμένες σε διάφορες γλώσσες προγραμματισμού. Αυτό είναι ένα επιθυμητό χαρακτηριστικό για ένα service - oriented API το οποίο δεν είναι

δεσμευμένο με μια συγκεκριμένη διαδικασία ή σύστημα και μπορεί να παρέχεται ως απομακρυσμένες κλήσεις διαδικασίας (remote procedure calls) ή ως web services.

1.2.1 Πως λειτουργούν τα APIs

Ένας από τους βασικούς σκοπούς μίας διεπαφής είναι να ορίζει και να διατυπώνει το σύνολο των λειτουργιών - υπηρεσιών που μπορεί να παρέχει μια βιβλιοθήκη ή ένα λειτουργικό σύστημα σε άλλα συστήματα, χωρίς να επιτρέπει πρόσβαση στον κώδικα που υλοποιεί αυτές τις υπηρεσίες. Παραδείγματος χάριν, το λειτουργικό σύστημα Windows έχει τη δική του Διεπαφή Προγραμματισμού Εφαρμογών (κλήσεις συστήματος), η μορφή (format) της οποίας διατίθεται από την κατασκευάστρια εταιρεία Microsoft. Αυτή η διεπαφή περιγράφει τους τρόπους αξιοποίησης, από προγράμματα χρήστη, του συνόλου των υπηρεσιών που παρέχει το λειτουργικό, χωρίς όμως να χρειάζεται η γνώση του χαμηλότερου κώδικα.

Ένα API είναι μία λογισμικό - προς - λογισμικό διεπαφή και όχι μία διεπαφή χρήστη. Με τα APIs, οι εφαρμογές μιλούν μεταξύ τους χωρίς τη γνώση ή την παρέμβαση κάποιου χρήστη. Στη περίπτωση της αγοράς εισιτηρίων online και εισαγωγής των στοιχείων της πιστωτικής κάρτας, η ιστοσελίδα χρησιμοποιεί ένα API για να στείλει τα στοιχεία της πιστωτικής κάρτας σε μια απομακρυσμένη εφαρμογή που ελέγχει αν οι πληροφορίες είναι σωστές ή όχι. Μόλις επιβεβαιωθεί η πληρωμή, η απομακρυσμένη εφαρμογή στέλνει μια απάντηση πίσω στη ιστοσελίδα λέγοντας ότι είναι OK για την έκδοση των εισιτηρίων. Ο χρήστης μπορεί να δει μόνο ένα interface - την ιστοσελίδα - αλλά πίσω από τις σκηνές, πολλές εφαρμογές εργάζονται από κοινού με τη χρήση APIs. Αυτό το είδος της ενσωμάτωσης ονομάζεται απρόσκοπτη (seamless), μιας και ο χρήστης δεν καταλαβαίνει πότε οι λειτουργίες του λογισμικού χειρίζονται από μία εφαρμογή ή από την άλλη.

1.2.2 Ο κύκλος ζωής ενός API

Ένα API ορίζεται ως ένα σταθερό περιβάλλον όπου μπορεί να χρησιμοποιείται στην πάροδο του χρόνου. Ένα API μπορεί να αναπτυχθεί είτε αυθόρμητα, κάποιος προγραμματιστής αναπτύσσει ένα χαρακτηριστικό, και ένας

άλλος προγραμματιστής θεωρεί ότι είναι χρήσιμο και αρχίζει να το χρησιμοποιεί. Αργότερα μοιράζονται τις εμπειρίες τους και συζητούν αλλαγές για καλύτερο σχεδιασμό του API και μετά από λίγες επαναλήψεις μπορεί να γίνει ένα χρήσιμο API, είτε από τον σχεδιασμό, υπάρχει μια γνωστή ανάγκη για μια σύμβαση μεταξύ δύο στοιχείων του συστήματος, συλλέγονται οι απαιτήσεις, ερευνάται η περιοχή του προβλήματος, οι περιπτώσεις χρήσης γίνονται κατανοητές, και στη συνέχεια, σχεδιάζει και γράφει ο προγραμματιστής το API. Στη συνέχεια μελετούνται διάφορες μορφές του API:

- Private είναι μια κατηγορία για τα χαρακτηριστικά που είναι προσβάσιμα, αλλά δεν προορίζονται για χρήση εκτός της μονάδας. Τέτοια χαρακτηριστικά μπορούν να αλλάζουν με κάθε έκδοση, ανάλογα με την επικινδυνότητά τους θα πρέπει να αποφεύγονται.

- Το Friend API χρησιμοποιείται για χαρακτηριστικά που είναι προσβάσιμα σε συγκεκριμένα στοιχεία στο σύστημα, το οποίο βοηθά να ξεπεραστεί η έλλειψη ενός σταθερού API, αλλά προορίζεται μόνο για χρήση μεταξύ αυτών των friend στοιχείων και κανενός άλλου.

- Υπό ανάπτυξη είναι ένα όνομα για μια σύμβαση που αναμένεται να γίνει ένα σταθερό API, αλλά δεν έχει ακόμη ολοκληρωθεί. Συμβατές αλλαγές μπορούν να γίνουν μεταξύ των εκδόσεων, αλλά θα πρέπει να είναι σπάνιες και όχι ριζικές.

- Σταθερές διεπαφές είναι εκείνες που έχουν λάβει μια τελική κατάσταση και οι συντηρητές είναι έτοιμοι να τις υποστηρίξουν για πάντα και ποτέ δεν τις αλλάζουν. Το "πάντα" και "ποτέ" δεν θα πρέπει να εκληφθεί ως απόλυτα. Είναι δυνατόν να αλλάξει η σύμβαση, αλλά μόνο σε μεγάλες εκδόσεις και μόνο μετά από προσεκτική σκέψη και σε περιπτώσεις όπου είναι επιτακτική ανάγκη να γίνει η αλλαγή.

- Third party διεπαφές που παρέχονται από άλλα μέρη που δεν ακολουθούν τους NetBeans κανόνες και ως εκ τούτου είναι δύσκολο να ταξινομηθούν. Προτιμάται να μην εκθέσουν αυτές τις διασυνδέσεις, ως μέρος των δικών τους συμβάσεων, προκειμένου να απομονώσουν τους χρήστες του NetBeans API από αναπάντεχες αλλαγές στις διασυνδέσεις.

- Το Standard είναι παρόμοιο με το Third party, παρέχεται από το NetBeans, αλλά αναμένεται να εξελιχθεί το περιβάλλον με συμβατό τρόπο (για παράδειγμα JSRs). Το πρότυπο αυτό δεν αναμένεται να αλλάζει συχνά.

- Ξεπερασμένο API, σχεδόν κάθε API ανεξάρτητα από το σε ποια κατάσταση είναι, γίνεται ξεπερασμένο. Συνήθως μια νέα, καλύτερη υποστήριξη για το ίδιο έργο έχει αναπτυχθεί το οποίο αντικαθιστά το παλιό API. Ένα σταθερό προηγουμένως API που άλλαξε τη σταθερότητα του για να καταργηθεί θα πρέπει να υποστηριχθεί για εύλογο χρονικό διάστημα για να επικοινωνεί με τους χρήστες, για να μεταβούν από την εφαρμογή στη νέα αντικατάσταση. Μετά από εκείνη τη στιγμή το API μπορεί να αφαιρεθεί.

1.2.3 API Βιβλιοθήκες και Πλαίσια

Ένα API συνήθως σχετίζεται με μια βιβλιοθήκη λογισμικού. Το API περιγράφει και καθορίζει την αναμενόμενη συμπεριφορά, ενώ η βιβλιοθήκη είναι μια πραγματική εφαρμογή αυτού του συνόλου κανόνων. Ένα ενιαίο API μπορεί να έχει πολλαπλές εφαρμογές (ή καμία, να είναι αφηρημένο), με τη μορφή των διαφόρων βιβλιοθηκών που μοιράζονται την ίδια διεπαφή προγραμματισμού.

Ένα API μπορεί επίσης να σχετίζεται με ένα πλαίσιο λογισμικού. Ένα πλαίσιο μπορεί να βασίζεται σε αρκετές βιβλιοθήκες, εφαρμόζοντας αρκετά API, αλλά σε αντίθεση με την κανονική χρήση του API, η πρόσβαση με τη συμπεριφορά ενσωματωμένη στο πλαίσιο μεσολαβεί, από την επέκταση του περιεχομένου με νέες κλάσεις συνδεδεμένες στο ίδιο το πλαίσιο. Επιπλέον, η συνολική ροή του προγράμματος ελέγχου μπορεί να είναι έξω από τον έλεγχο του καλούντος, και στα χέρια του πλαισίου μέσω της αναστροφής του ελέγχου ή ενός παρόμοιου μηχανισμού.

1.3 ΜΕΘΟΔΟΛΟΓΙΑ ΣΥΣΤΗΜΑΤΙΚΗΣ ΑΝΑΣΚΟΠΗΣΗΣ ΤΗΣ ΒΙΒΛΙΟΓΡΑΦΙΑΣ

Στο (Brereton et al., 2007) προτείνεται ότι μια συστηματική ανασκόπηση της βιβλιογραφίας πρέπει να αποτελείται από τρία βήματα: το σχεδιασμό, τη διενέργεια και τη τεκμηρίωση της ανασκόπησης.

Κατά τη φάση του σχεδιασμού πρέπει να καθοριστούν τα ερευνητικά ερωτήματα, να αναπτυχθεί ένα πρωτόκολλο ανασκόπησης και στο τέλος να

επικυρωθεί. Ο καθορισμός των ερευνητικών ερωτημάτων, αποτελεί το πιο κρίσιμο στοιχείο της συστηματικής ανασκόπησης. Τα ερευνητικά ερωτήματα χρησιμοποιούνται για να εντοπίσουν τις λέξεις κλειδιά που πρέπει να χρησιμοποιηθούν για την αυτόματη αναζήτηση των σχετικών ερευνών. Οι λέξεις αυτές, ουσιαστικά καθορίζουν τα δεδομένα που πρέπει να εξαχθούν από κάθε πρωταρχική μελέτη και περιορίζουν τη συνολική διαδικασία. Τα ερωτήματα της έρευνας είναι το κομμάτι του πρωτοκόλλου που δεν μπορεί να τροποποιηθεί μετά τη αποδοχή του πρωτοκόλλου. Το πρωτόκολλο ανασκόπησης που αναπτύσσεται, δίνει πληροφορίες για τον σχεδιασμό της ανασκόπησης, συμπεριλαμβάνοντας την προδιαγραφή της διαδικασίας που θα ακολουθηθεί, τις συνθήκες και τις μετρικές ποιότητας που θα εφαρμοστούν όταν επιλεγεί μια πρωταρχική μελέτη, και την κατανομή ορισμένων ενεργειών. Η επικύρωση του πρωτοκόλλου θεωρείται απαραίτητη, αφού το πρωτόκολλο αποτελεί κρίσιμο στοιχείο της ανασκόπησης.

Στη φάση της διενέργειας της ανασκόπησης, και εφόσον το πρωτόκολλο έχει τελειοποιηθεί, πρέπει να καταστρωθεί μια στρατηγική έρευνας για να εντοπιστούν οι σχετικές έρευνες που έχουν διεξαχθεί. Αμέσως μετά πρέπει να επιλεχθούν όσες κρίνονται κατάλληλες. Η διαδικασία επιλογής αποτελείται συνήθως από δυο στάδια. Αρχικά εξετάζεται ο τίτλος και στη συνέχεια η περίληψη των άρθρων που προκύπτουν από την αρχική αναζήτηση. Άρθρα που προκύπτει ότι δεν είναι σχετικά απορρίπτονται. Στην επόμενη φάση της διαδικασίας, εξετάζεται ολόκληρο το κείμενο των άρθρων που δεν έχουν απορριφτεί. Η ανασκόπηση γίνεται με βάση τα κριτήρια συμπερίληψης ή αποκλεισμού των άρθρων. Έπειτα, πρέπει να αξιολογηθεί η ποιότητα των ερευνών προκειμένου να ελαχιστοποιηθούν οι πιθανές προκαταλήψεις και να μεγιστοποιηθεί η εσωτερική και εξωτερική εγκυρότητα. Τέλος πρέπει να εξαχθούν τα απαιτούμενα δεδομένα ώστε να καταγραφούν με ακρίβεια οι πληροφορίες που θέλουν να αποσπάσουν οι ερευνητές από τις πρωταρχικές μελέτες και να ανασυντεθούν με τρόπο τέτοιο, ώστε να απαντούν στα ερωτήματα της έρευνας.

Η τελευταία φάση μιας συστηματικής ανασκόπησης είναι η διαδικασία της τεκμηρίωσης, κατά την οποία μετά την ολοκλήρωση της συστηματικής ανασκόπησης της βιβλιογραφίας, γράφεται μια λεπτομερής αναφορά για την ανασκόπηση και έπειτα αξιολογείται.

Σύμφωνα με τα (Kitchenham, Joint Technical Report και Kitchenham et al., 2009) το σχεδιάγραμμα της ανασκόπησης αποτελείται από έξι μέρη: (α) ορισμό των ερευνητικών ερωτημάτων, (β) ορισμό της διαδικασίας αναζήτησης, (γ) ορισμό των κριτηρίων συμπερίληψης και αποκλεισμού, (δ) ορισμό της ποιοτικής αξιολόγησης, (ε) ορισμό της διαδικασίας συλλογής δεδομένων, και (στ) ορισμό της ανάλυσης δεδομένων.

1.4 ΜΕΛΕΤΗ ΤΩΝ ΠΛΕΟΝΕΚΤΗΜΑΤΩΝ ΚΑΙ ΜΕΙΟΝΕΚΤΗΜΑΤΩΝ ΤΩΝ APIs

Στην ενότητα αυτή γίνεται μια σύντομη αναφορά σε μερικά από τα πλεονεκτήματα και τα μειονεκτήματα των διεπαφών προγραμματισμού εφαρμογών. Πιο διεξοδική ανάλυση γίνεται στο 3ο κεφάλαιο.

Πρόσφατα έχει υιοθετηθεί μια στρατηγική ανάπτυξης επαναχρησιμοποιήσιμων στοιχείων λογισμικού στην οποία, η έννοια του API έχει ένα σημαντικό ρόλο. Η βασική ιδέα είναι ότι κάθε στοιχείο του λογισμικού θα έχει ένα κοινό και σταθερό API μέσω του οποίου οι χρήστες μπορούν να έχουν πρόσβαση στο σύνολο των υπηρεσιών που παρέχονται από το στοιχείο αυτό. Τα APIs πρέπει να είναι δημόσια για να επιτρέπουν την πρόσβαση στις υπηρεσίες που παρέχουν. Τα APIs είναι διεπαφές που αποτελούνται από τις δημόσιες κλάσεις τις μεθόδους, καθώς και τη σχετική τεκμηρίωση (σε αυτή τη περίπτωση, τα αρχεία javadoc), διευκολύνουν την απομόνωση εργασιών ανάπτυξης, επιτρέποντας στους μηχανικούς λογισμικού να λειτουργήσουν χωρίς να επηρεάζονται από τους το έργο των συναδέλφων τους. Αυτή η ανάγκη για απομόνωση είναι ένα κοινό θέμα στη μηχανική λογισμικού, λόγω των πολλών αλληλεξαρτήσεων που συμβαίνουν σε αυτές τις προσπάθειες.

Μια σημαντική πτυχή του API είναι η σταθερότητα. Ένα σταθερό API δεν υπόκεινται σε συχνές αλλαγές. Οι αλλαγές στο API απαιτούν αλλαγές στον API κώδικα πελάτη. Αυτή η κατάσταση μπορεί να καταστεί προβληματική αν οι αλλαγές στο API συμβαίνουν πάρα πολύ συχνά. Η επίδραση της μεταβολής του API είναι υψηλή, διότι δυνητικά οδηγεί σε άλλες αλλαγές στον πηγαίο κώδικα. Η κατάσταση αυτή θα μπορούσε να είναι περισσότερο ή λιγότερο προβληματική, ανάλογα με τον τύπο και την ποσότητα των αλλαγών που εμφανίζονται επίσης στο API. Αυτή η έλλειψη τεχνολογικής υποστήριξης κάνει αυτές τις αλλαγές δαπανηρή και επώδυνη για τους χρήστες του API.

ΚΕΦΑΛΑΙΟ 2: ΑΝΑΣΚΟΠΗΣΗ ΤΗΣ ΒΙΒΛΙΟΓΡΑΦΙΑΣ " ΔΙΑΣΥΝΔΕΣΕΙΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΕΦΑΡΜΟΓΩΝ (APIs)"

Το κεφάλαιο αυτό, στοχεύει στο να συνοψίσει την υπάρχουσα μέχρι σήμερα ερευνητική δραστηριότητα στο πεδίο της χρήσης των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών (APIs). Πρώτο βήμα, για να επιτευχθεί αυτό, είναι η διεξαγωγή μιας συστηματικής βιβλιογραφικής ανασκόπησης.

Στην ενότητα 2.1, με βάση τη μεθοδολογία που παρουσιάζεται στην ενότητα 1.3 ορίζονται τα ερευνητικά ερωτήματα που θα διερευνήσει η μελέτη αυτή. Η ενότητα 2.2, παρουσιάζει διάφορα στατιστικά (descriptive statistics) για τα ευρήματα της έρευνας. Στην ενότητα 2.3 παρουσιάζονται τα αποτελέσματα της μελέτης, που χωρίζονται σε υποενότητες βάση των ερευνητικών ερωτημάτων που διατυπώνονται στην ενότητα 2.1.1. Τέλος παρουσιάζονται τα συμπεράσματα στην ενότητα 2.4.

2.1 ΜΕΘΟΔΟΛΟΓΙΑ ΑΝΑΣΚΟΠΗΣΗΣ

Σύμφωνα με την μεθοδολογία που περιγράφεται στην ενότητα 1.3 παρουσιάζουμε τα έξι βήματα της συστηματικής ανασκόπησης της βιβλιογραφίας που διεξαγάγαμε στα πλαίσια αυτής της μελέτης.

2.1.1 Τα ερωτήματα της έρευνας

Στη μελέτη αυτή, θα διερευνήσουμε ζητήματα που αφορούν την μέχρι σήμερα ερευνητική δραστηριότητα πάνω στις βιβλιοθήκες διασύνδεσης προγραμματισμού εφαρμογών (APIs). Τα βασικά ερευνητικά ερωτήματα που αποτυπώνονται στην μελέτη είναι:

Q1: Ποια είναι τα πιο δημοφιλή ερευνητικά θέματα σχετικά με τις βιβλιοθήκες διασύνδεσης προγραμματισμού εφαρμογών (APIs);

Q2: Ποια είναι τα χαρακτηριστικά ποιότητας των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών;

Q3: Πως μπορούν οι επανασχεδιάσεις σε επίπεδο API να διευκολύνουν τη διόρθωση σφαλμάτων;

Q4: Ποια είναι τα εμπόδια που συντελούν στην δυσκολία εκμάθησης του API;

Q5: Ποιοι είναι οι παράγοντες που επηρεάζουν την χρηστικότητα του API;

2.1.2 Η διαδικασία αναζήτησης

Η διαδικασία αναζήτησης διεξήχθη μέσω των ιστοχώρων των εξής πέντε ψηφιακών βιβλιοθηκών: ACM, IEEE, ScienceDirect, Springer, Scholar. Οι διευθύνσεις URL των παραπάνω ψηφιακών βιβλιοθηκών είναι:

1. http://www.computer.org/ ,
2. http://dl.acm.org/ ,
3. http://www.sciencedirect.com/,
4. http://www.springerlink.com/,
5. http://scholar.google.gr/.

Σαν λέξεις κλειδιά για την αναζήτηση, χρησιμοποιήθηκαν τρεις διαφορετικές λέξεις. Στην πρώτη περίπτωση διεξήχθησαν σύνθετες αναζητήσεις, όπου η λέξη κλειδί, δηλαδή η λέξη «API» είχε τον περιορισμό να βρίσκετε σε όλο το κείμενο, την περίληψη και τον τίτλο όλων των μελετών. Στην δεύτερη περίπτωση χρησιμοποιήθηκε η λέξη «Framework», ενώ στην τρίτη περίπτωση η λέξη «Library». Το σύνολο των άρθρων που επιστράφηκαν από τις παραπάνω αναζητήσεις αντιστοιχούσε σε 2291 άρθρα. Ωστόσο, ο μεγαλύτερος αριθμός από τα άρθρα αυτά θεωρήθηκε ότι δεν σχετίζονταν ικανοποιητικά με τις βιβλιοθήκες διασύνδεσης προγραμματισμού εφαρμογών (API) και αποκλείστηκαν. Ο αποκλεισμός των μη σχετικών άρθρων διεξήχθη με το χέρι, σύμφωνα με τα κριτήρια συμπερίληψης και αποκλεισμού που ορίζονται στην επόμενη υποενότητα.

2.1.3 Κριτήρια Συμπερίληψης και Αποκλεισμού

Σύμφωνα με το (Dyba and Dingsoyr, 2008), υπάρχουν τέσσερα στάδια φιλτραρίσματος του συνόλου των άρθρων, προκειμένου να προκύψει το σύνολο των πρωταρχικών μελετών. Τα βήματα αυτά είναι τα εξής:

- Εύρεση των σχετικών μελετών – αναζήτηση σε ψηφιακές βιβλιοθήκες. (με την ολοκλήρωση του βήματος αυτού το σύνολο των άρθρων ανέρχονταν στα 2291 άρθρα.)
- Αποκλεισμός μελετών βάση του τίτλο τους. (με την ολοκλήρωση του βήματος αυτού το σύνολο των άρθρων ανέρχονταν στα 843 άρθρα.)
- Αποκλεισμός μελετών βάση της περίληψής τους. (με την ολοκλήρωση του βήματος αυτού το σύνολο των άρθρων ανέρχονταν στα 113 άρθρα.)
- Προσεκτική μελέτη των άρθρων και επιλογή των πιο σχετικών με τις διασυνδέσεις προγραμματισμού εφαρμογών, βάση του πλήρες κειμένου. (με την ολοκλήρωση του βήματος αυτού το τελικό σύνολο των πρωταρχικών μελετών αποτελούνταν από 47 άρθρα.)

Ο πιο συνηθισμένος λόγος για τον αποκλεισμό ενός άρθρου από τον τίτλο του, ήταν ότι το άρθρο δεν ασχολούνταν με τις διασυνδέσεις προγραμματισμού εφαρμογών, τις βιβλιοθήκες και τα πλαίσια. Επιπλέον, κατά την εξέταση των περιλήψεων, η πλειοψηφία των άρθρων που αποκλείστηκαν ασχολούνταν με μη σχετική αναφορά στα APIs. Τέλος, το βασικό κριτήριο απόρριψης άρθρων στην τελευταία φάση της μελέτης του πλήρες κειμένου, ήταν η απουσία αναφοράς στα πλεονεκτήματα και μειονεκτήματα των APIs.

Το τελικό σύνολο των πρωταρχικών μελετών αποτελείται από τα ερευνητικά άρθρα που παρουσιάζονται στο Παράρτημα.

2.1.4 Ποιοτική Αξιολόγηση

Η ποιότητα ενός άρθρου που συντάσσει μια συστηματική ανασκόπηση, σχετίζεται άμεσα με την ποιότητα των πρωταρχικών μελετών, από την άποψη ότι τα αποτελέσματα και τα συμπεράσματα της δευτερογενούς μελέτης βασίζονται στα ευρήματα των πρωταρχικών μελετών. Έτσι, σε μια ανασκόπηση, είναι σημαντικό να συμπεριληφθούν πρωταρχικές μελέτες που βασίζονται σε σταθερές μεθόδους και παρουσιάζουν ξεκάθαρα τα αποτελέσματά τους. Για να επιτευχθεί ο στόχος αυτός, στην ανασκόπηση μας, συμπεριλάβαμε άρθρα που έχουν δημοσιευθεί στα καλύτερα περιοδικά και συνέδρια, καθώς και σε workshops που διεξήχθησαν στα πλαίσια κορυφαίων συνεδρίων στον τομέα της μηχανικής λογισμικού.

2.1.5 Συλλογή Δεδομένων

Κατά τη διάρκεια της επιλογής των άρθρων, συγκεντρώσαμε ένα σύνολο μεταβλητών που περιέγραφαν την εκάστοτε πρωταρχική μελέτη. Για κάθε μελέτη, καταγράφαμε τα εξής δεδομένα:

[A1] Τύπο δημοσίευσης (περιοδικό, συνέδριο, workshop)
[A2] Τόπο δημοσίευσης (όνομα συνεδρίου ή περιοδικού)
[A3] Έτος δημοσίευσης
[A4] Αντικείμενο έρευνας

Για τις μελέτες που ασχολούνταν με τις βιβλιοθήκες διασυνδέσεων προγραμματισμού εφαρμογών, καταγράφονται επιπλέον πληροφορίες:

[B1] Τα χαρακτηριστικά ποιότητας που βρέθηκαν
[B2] Τις μετρικές λογισμικού που χρησιμοποιήθηκαν
[B3] Το είδος της εμπειρικής μελέτης
[B4] Χρήση επανασχεδιάσεων σε επίπεδο API για τη διευκόλυνση διόρθωσης σφαλμάτων
[B5] Εμπόδια εκμάθησης του API
[B6] Παράγοντες που επηρεάζουν την χρηστικότητα του API

2.1.6 Ανάλυση Δεδομένων

Τα δεδομένα που συλλέχθηκαν από τις μεταβλητές [A1]–[A3] χρησιμοποιήθηκαν για να παρέχουν περιγραφικά στατιστικά για την έρευνα στο πεδίο των διασυνδέσεων προγραμματισμού εφαρμογών (APIs). Η μεταβλητή [A4] χρησιμοποιήθηκε για να εντοπιστούν τα πιο δημοφιλή ερευνητικά θέματα που ασχολούνται με τις διεπαφές προγραμματισμού εφαρμογών (APIs) καθώς και για να βοηθήσει στην περιγραφή της μέχρι τώρα ερευνητικής διαδικασίας σε κάθε πεδίο (αφορά το Q1).

Όσον αφορά τα χαρακτηριστικά ποιότητας των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών (αφορά το Q2), αξιοποιούνται οι μεταβλητές [B1], [B2], [B3]. Η μεταβλητή [B4] χρησιμοποιήθηκε για τη διόρθωση σφαλμάτων που μπορούν να αντιμετωπιστούν με τη βοήθεια των επανασχεδιάσεων σε επίπεδο API (αφορά το Q3).

Η μεταβλητή [B5] χρησιμοποιήθηκε για τον εντοπισμό των εμποδίων εκμάθησης του API (αφορά το Q4). Τέλος, η μεταβλητή [B6] χρησιμοποιήθηκε για να προσδιορίσει τους παράγοντες που επηρεάζουν την χρηστικότητα του API (αφορά το Q5). Το σύνολο των δεδομένων, αναλύεται και οποτικοποιείται με τη βοήθεια στατιστικών μεθόδων και γραφημάτων.

2.2 ΑΠΟΤΕΛΕΣΜΑΤΑ

Στο κεφάλαιο αυτό παρουσιάζονται τα περιγραφικά στατιστικά που προέκυψαν από την ανάλυση του συνόλου των δεδομένων. Στον Πίνακα 1, συγκεντρώσαμε τον αριθμό των δημοσιεύσεων κατηγοριοποιώντας σύμφωνα με το πού δημοσιεύτηκαν. Το συνέδριο *International Conference on Software Engineering* φαίνεται ότι είναι το πιο ενεργό, διότι έχουν γίνει 5 δημοσιεύσεις σε αυτό, ακολουθεί το *International Conference on Program Comprehension* με 3 δημοσιεύσεις, και στη συνέχεια ακολουθούν τα υπόλοιπα στα οποία έχουν γίνει 2 ή 1 δημοσιεύσεις.

Πίνακας 1. Τόποι Δημοσίευσης

A/A	**Όνομα**	**Δημοσιεύσεις**		**Ψηφιακή Βιβλιοθήκη**
1	International Conference on Automated Software Engineering	1	2.12%	IEEE/ACM
2	International Conference on Software Engineering (ICSE)	5	10.63%	IEEE/ACM
3	IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)	1	2.12%	IEEE
4	International Conference on Program Comprehension	3	6.38%	IEEE
5	International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM)	1	2.12%	IEEE
6	International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST)	1	2.12%	IEEE
7	International Conference on e-Business (ICEBE)	1	2.12%	IEEE
8	Working Conference on Reverse Engineering	2	4.25%	IEEE
9	International Professional Communication Conference	1	2.12%	IEEE
10	IEEE Computer Society Press Los Alamitos	1	2.12%	IEEE
11	International Conference on Quality Software (QSIC)	2	4.25%	IEEE
12	International Conference on Computational Intelligence and Security	1	2.12%	IEEE
13	IEEE Transactions on Software Engineering	2	4.25%	IEEE
14	Journal Parallel Computing	2	4.25%	ACM

15	The SIGCHI Conference on Human Factors in Computing Systems	2	4.25%	ACM
16	SIGSOFT International Symposium on Foundations of software engineering (SIGSOFT)	1	2.12%	ACM
17	Journal ACM Transactions on Mathematical Software (TOMS)	1	2.12%	ACM
18	Journal Software—Practice & Experience	1	2.12%	ACM
19	Technology of Object-Oriented Languages and Systems (TOOLS)	1	2.12%	ACM
20	USENIX Symposium on Internet Technologies and Systems	1	2.12%	ACM
21	ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE)	1	2.12%	ACM
22	International Workshop on Mining Software Repositories (MSR)	2	4.25%	ACM
23	Procedia Engineering	1	2.12%	Science Direct
24	Future Generation Computer Systems	1	2.12%	Science Direct
25	Journal of Systems and Software	2	4.25%	Science Direct
26	Information and Software Technology	1	2.12%	Science Direct
27	Avionics Conference and Exhibition	1	2.12%	Science Direct
28	Workshop on Generative Technologies (WGT)	1	2.12%	Science Direct
29	The Semantic Web-ISWC Lecture Notes in Computer Science	1	2.12%	Springer
30	Advances in Software Engineering	1	2.12%	Springer
31	International Conference on Algorithms and architectures for parallel processing	1	2.12%	Springer
32	Methods and Tools of Parallel Programming Multicomputers	1	2.12%	Springer
33	Research and Advanced Technology for Digital Libraries	2	4.25%	Springer

Ο Πίνακας 2, παρουσιάζει την ερευνητική δραστηριότητα σε πιο εξειδικευμένα θέματα που αφορούν τις διασυνδέσεις προγραμματισμού εφαρμογών, καθώς επίσης και την καταμέτρηση των δημοσιευμένων άρθρων σε κάθε ερευνητικό θέμα. Κατά τη διάρκεια της ανασκόπησής μας, βρήκαμε τρεις βασικούς τομείς στην έρευνα των διασυνδέσεων προγραμματισμού εφαρμογών, που είναι οι εξής: (α) ποιότητα των διασυνδέσεων προγραμματισμού εφαρμογών, (β) ανάλυση των διασυνδέσεων προγραμματισμού εφαρμογών, και (γ) εφαρμογή των διασυνδέσεων προγραμματισμού εφαρμογών. Όσα άρθρα δεν μπορούσαν να κατηγοριοποιηθούν σε μία από τις κατηγορίες αυτές, τοποθετούνταν σε μια τέταρτη γενικευμένη κατηγορία.

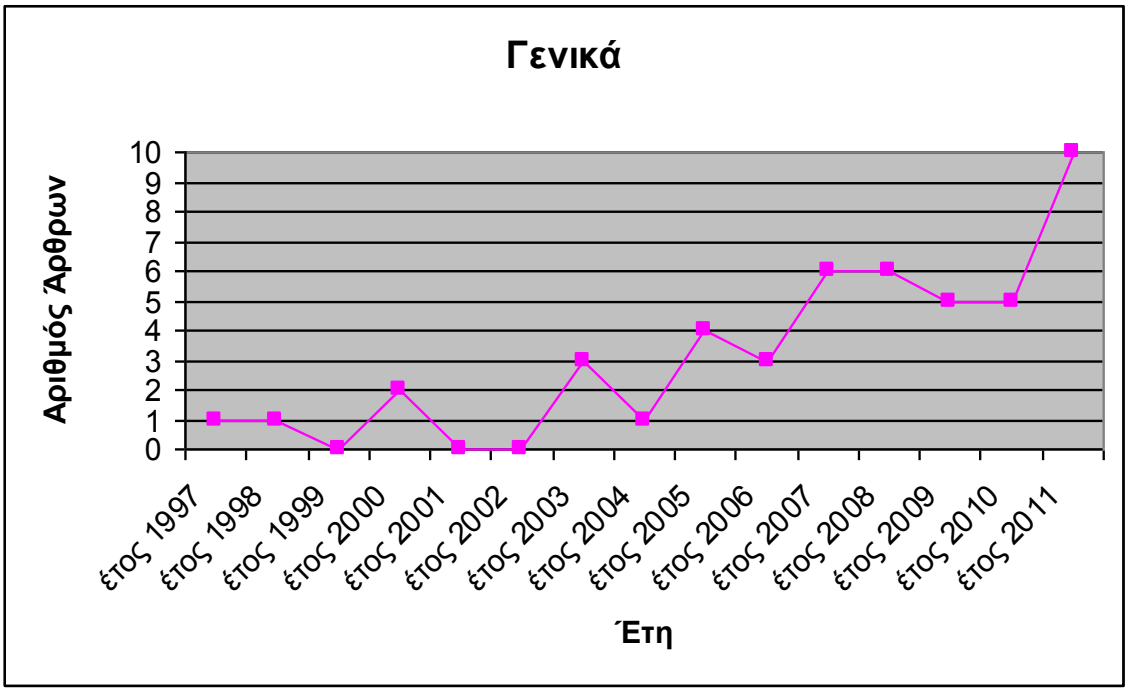
Τα αποτελέσματα του πινάκα, δείχνουν ότι ο ισχυρότερος τομέας έρευνας για τις διασυνδέσεις προγραμματισμού εφαρμογών (APIs), είναι εκείνος που εξετάζει θέματα ανάλυσης των διασυνδέσεων προγραμματισμού εφαρμογών, ενώ

ακολουθούν έρευνες για τη μελέτη διαφόρων θεμάτων των APIs και θέματα εφαρμογής τους.

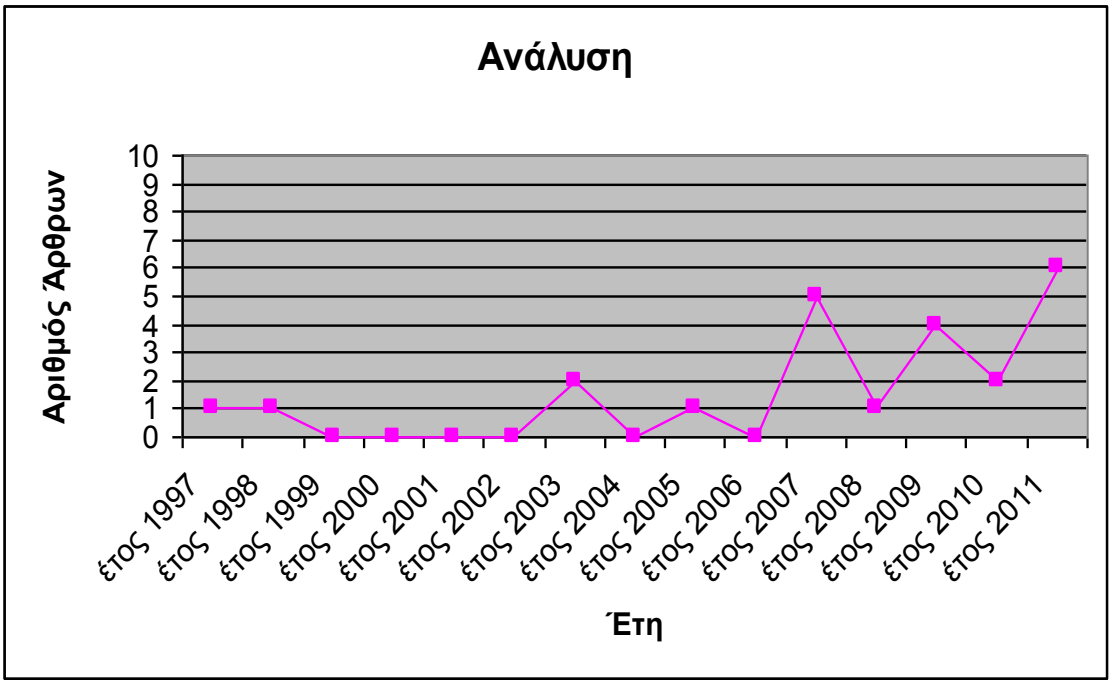
Πίνακας 2. Ερευνητικά Θέματα

A/A	Όνομα	Δημοσιεύσεις		Μελέτες
1	Ποιότητα	6	12,76%	[S01],[S03],[S22],[S23],[S36],[S47]
2	Εφαρμογή	7	14.89%	[S02],[S11],[S12],[S18],[S19],[S27],[S37]
3	Διάφορα Θέματα	11	23,40%	[S09],[S13],[S14],[S15],[S29],[S31],[S32],[S33],[S34], [S35],[S38]
4	Ανάλυση	23	48,93%	[S04],[S05],[S06],[S07],[S08],[S10],[S16],[S17],[S20], [S21],[S24],[S25],[S26],[S28],[S30],[S39],[S40],[S41], [S42],[S43],[S44],[S45],[S46]

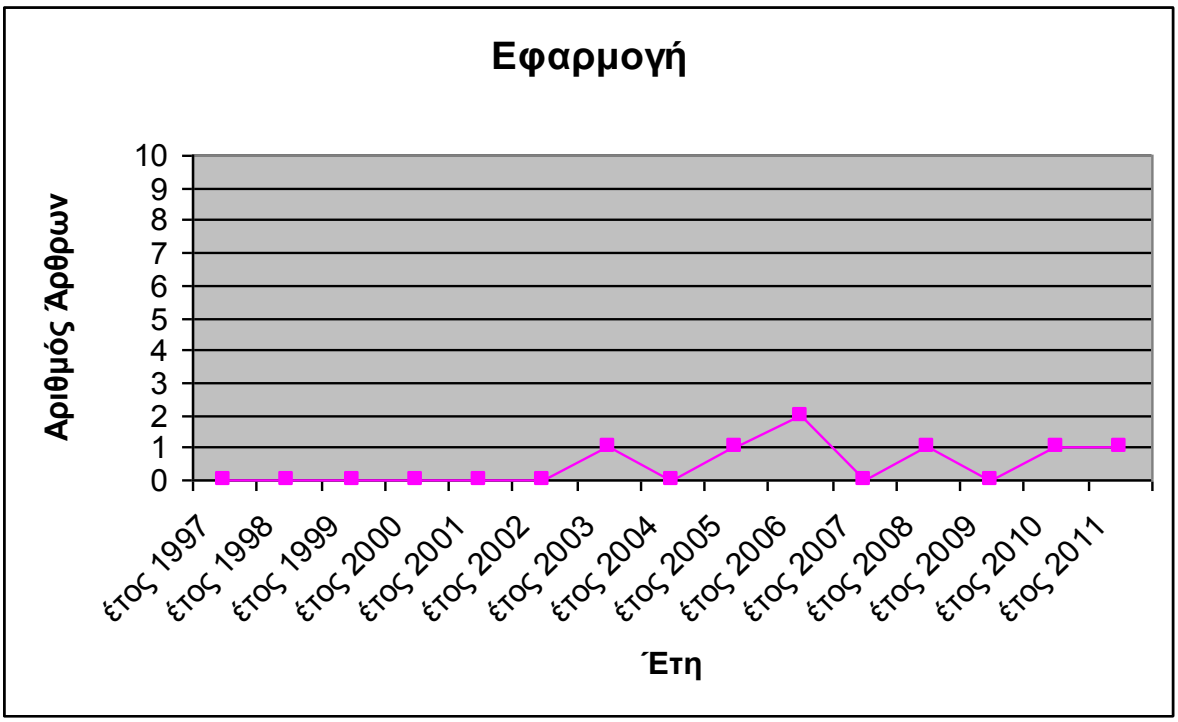
Τα σχήματα 1-5 απεικονίζουν την τάση της ερευνητικής δραστηριότητας για κάθε ερευνητικό θέμα κατά την πάροδο του χρόνου.



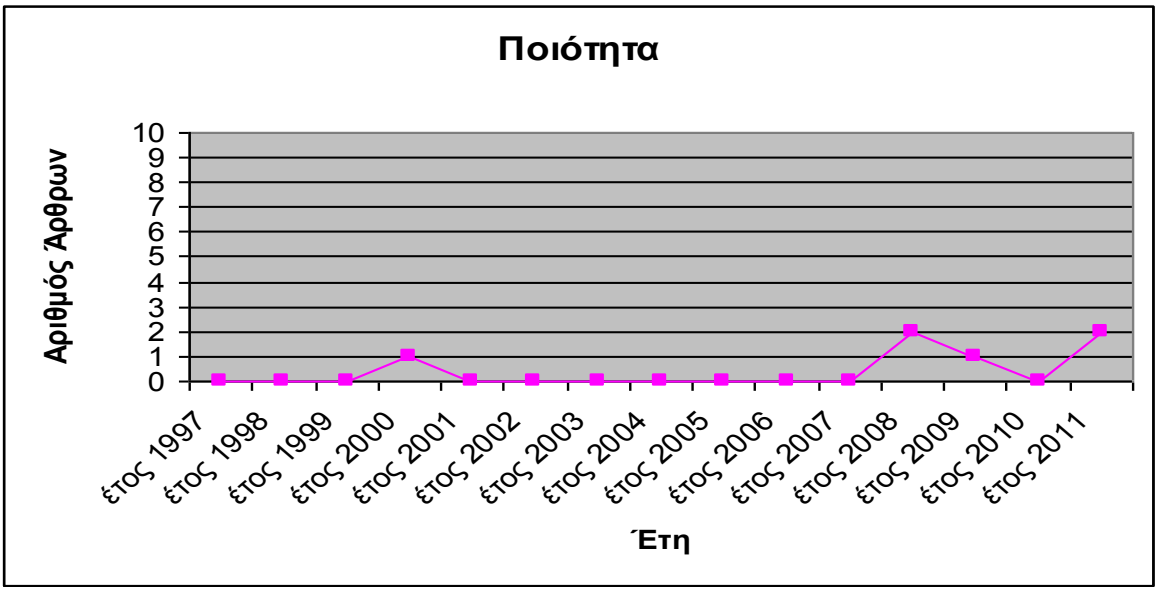
Σχήμα 1. Γενική έρευνα κατά την πάροδο του χρόνου



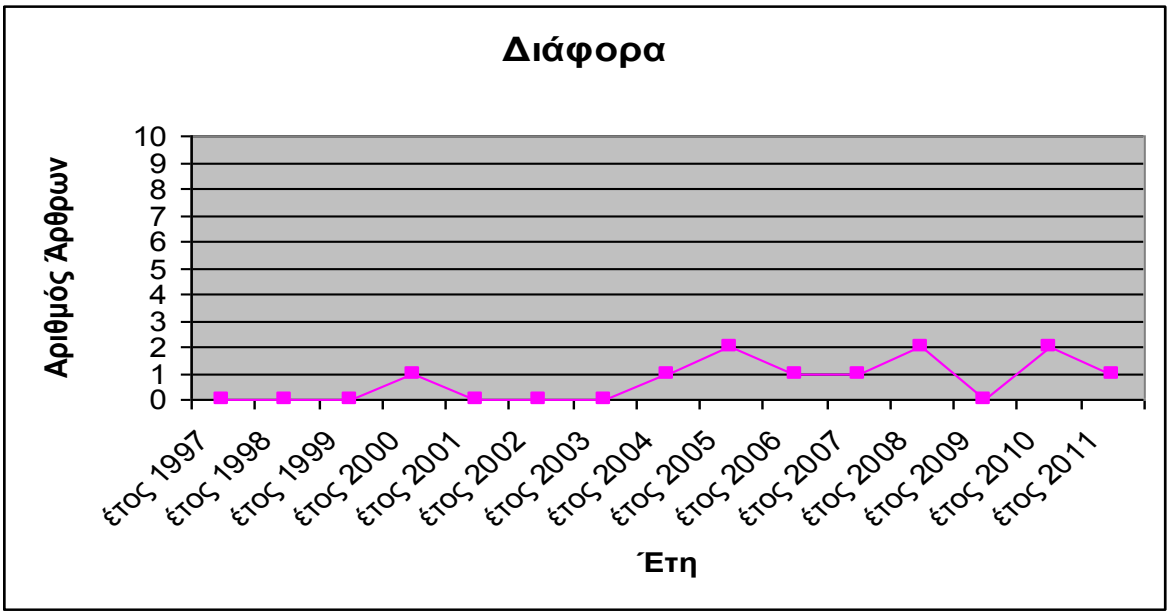
Σχήμα 2. Έρευνα κατά την πάροδο του χρόνου (Ανάλυση APIs)



Σχήμα 3. Έρευνα κατά την πάροδο του χρόνου (Εφαρμογή APIs)



Σχήμα 4. Έρευνα κατά την πάροδο του χρόνου (Ποιότητα APIs)



Σχήμα 5. Έρευνα κατά την πάροδο του χρόνου (Διάφορα APIs)

Τα παραπάνω στοιχεία δείχνουν ότι, λαμβάνοντας υπόψη τη συνολική έρευνα στην πάροδο του χρόνου, παρατηρούμε ότι ο όγκος της έρευνας αυξάνεται κατά τη διάρκεια των χρόνων. Ωστόσο, υπάρχουν χρόνια, όπου ο αριθμός των δημοσιευμένων άρθρων στις βιβλιοθήκες των APIs μειώθηκε σε σχέση με το προηγούμενο έτος. Πιο συγκεκριμένα, η έρευνα σχετικά με την ανάλυση των APIs είναι πιο έντονη το 2007(21,73% του αριθμού των δημοσιεύσεων που αναφέρονται στην ανάλυση των APIs) και το 2011(26,08% του αριθμού των

δημοσιεύσεων που αναφέρονται στην ανάλυση των APIs). Η έρευνα για την ποιότητα και την εφαρμογή των APIs ήταν περιορισμένη με δημοσιεύσεις ενός μικρού αριθμού άρθρων μεταξύ των ετών 2006, 2008 και 2011. Τέλος, η έρευνα σχετικά με διάφορα θέματα των APIs ήταν αντίστοιχα περιορισμένη με δημοσιεύσεις ενός μικρού αριθμού άρθρων μεταξύ των ετών 2005, 2008 και 2010.

Τέλος, στον Πίνακα 3, παρουσιάζουμε τη συχνότητα εφαρμογής κάθε εμπειρικής μεθόδου, δηλαδή βιβλιογραφική ανασκόπηση, εννοιολογική ανάλυση, μελέτη αξιολόγησης, μελέτης περίπτωσης, πείραμα και έρευνα, που χρησιμοποιούνται για την αξιολόγηση των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών. Παρατηρείται ότι η κυρίαρχη μεθοδολογία είναι η «έρευνα», ακολουθούν τα «πειράματα», η «μελέτη περίπτωσης», η «μελέτη αξιολόγησης», η «εννοιολογική ανάλυση» και τέλος η «βιβλιογραφική ανασκόπηση».

Πίνακας 3. Εμπειρικές Μέθοδοι

A/A	Όνομα	Δημοσιεύσεις	
1	Βιβλιογραφική Ανασκόπηση	1	2,12%
2	Εννοιολογική Ανάλυση	4	8,51%
3	Μελέτη Αξιολόγησης	8	17,02%
4	Μελέτη Περίπτωσης	10	21,27%
5	Πείραμα	11	23,40%
6	Έρευνα	13	27,65%

2.3 ΜΕΛΕΤΗ ΤΩΝ ΕΡΕΥΝΗΤΙΚΩΝ ΕΡΩΤΗΣΕΩΝ

Στο κεφάλαιο αυτό, συζητάμε τα αποτελέσματα της ανασκόπησής μας, σύμφωνα με τα ερωτήματα της έρευνας που έχουν διατυπωθεί. Στην ενότητα 2.3.1 παρουσιάζουμε την μέχρι τώρα ερευνητική δραστηριότητα σχετικά με τις βιβλιοθήκες διασύνδεσης προγραμματισμού εφαρμογών (APIs). Στην ενότητα 2.3.2 συζητάμε τα αποτελέσματα των χαρακτηριστικών ποιότητας των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών. Στην ενότητα 2.3.2.1 μελετούνται οι μετρικές λοιμικού που αναφέραμε στην ενότητα 2.3.2. Στην ενότητα 2.3.3 συζητάμε για τον τρόπο που οι επανασχεδιάσεις σε επίπεδο API βοηθούν στις διορθώσεις των σφαλμάτων. Στην ενότητα 2.3.4 παρουσιάζουμε τα εμπόδια

εκμάθησης του API και στην ενότητα 2.3.5 παρουσιάζουμε τους παράγοντες που επηρεάζουν την χρηστικότητα του API.

2.3.1 Η ερευνητική δραστηριότητα μέχρι τώρα

Η ερευνητική δραστηριότητα μέχρι σήμερα, σχετικά με τις διασυνδέσεις προγραμματισμού εφαρμογών, χωρίζεται σε τέσσερις ερευνητικούς τομείς, όπως προέκυψε από την ανάλυση των δεδομένων μας. Ο πρώτος τομέας, ονομάζεται Εφαρμογή των Διασυνδέσεων Προγραμματισμού Εφαρμογών, περιλαμβάνει άρθρα που παρουσιάζουν τεχνικές, δοκιμές, υλοποίηση, μεθόδους και εργαλεία που αυτοματοποιούν ή βοηθούν την εφαρμογή των APIs. Ο δεύτερος τομέας, ονομάζεται Ποιότητα των Διασυνδέσεων Προγραμματισμού Εφαρμογών και περιλαμβάνει μελέτες που ασχολούνται με την επίδραση των ποιοτικών χαρακτηριστικών στις βιβλιοθήκες Διασύνδεσης Προγραμματισμού Εφαρμογών. Τρίτος, έρχεται ο τομέας Ανάλυσης των Διασυνδέσεων Προγραμματισμού Εφαρμογών, που ασχολείται με άρθρα που ερευνούν αρχές σχεδιασμού, αντικειμενοστρεφή λογική, προδιαγραφές κτλ. Ο τελευταίος τομέας έρευνας, αποτελείται από έρευνες που δεν μπορούν να κατηγοριοποιηθούν σε κάποια από τις άλλες κατηγορίες και για αυτό ονομάζεται Γενικά Ζητήματα, σχετικά με τις διασυνδέσεις προγραμματισμού εφαρμογών. Ακολουθεί μια αναλυτική περιγραφή της δραστηριότητας καθενός από τους τομείς αυτούς.

2.3.1.1 Εφαρμογή των Διασυνδέσεων Προγραμματισμού Εφαρμογών (APIs)

Στην πρώτη κατηγορία ανήκουν 7 μελέτες οι οποίες σχετίζονται με θέματα εργαλείων και τεχνικών, τον έλεγχο, την επίδοση, τις πλατφόρμες δοκιμών, τους χρόνους εκτέλεσης κ.α των APIs. Τα άρθρα (Pai et al., 2003, Singh et al., 2005, Sudarsan και Ribbens, 2010, Carvalho και Cachopo, 2011, Gotlieb και Bernard, 2006) περιγράφουν τον σχεδιασμό, την εφαρμογή και την επίδοση των διασυνδέσεων προγραμματισμού εφαρμογών.

Το (Pai et al., 2003) περιγράφει τον σχεδιασμό, την υλοποίηση και την απόδοση μιας απλής αλλά και ισχυρής διασύνδεσης προγραμματισμού εφαρμογών (Application Programming Interface - API) για την παροχή εκτεταμένων υπηρεσιών σε μια μεσολάβηση κρυφής μνήμης (cache). Αυτό το API διευκολύνει την ανάπτυξη του εξατομικευμένου περιεχομένου προσαρμογής, τη

διαχείριση περιεχομένου, καθώς και εξειδικευμένες δυνατότητες διαχείρισης. Αναπτύχθηκαν διάφορες μονάδες που εκμεταλλεύονται αυτό το API για να εκτελέσουν διάφορες εργασίες στο πλαίσιο της μεσολάβησης, συμπεριλαμβανομένης μιας μονάδας για να υποστηρίξει το πρωτόκολλο διαδικτύου προσαρμογής περιεχομένου (ICAP - Internet Content Adaptation Protocol), χωρίς καθόλου αλλαγές στον πυρήνα μεσολάβησης. Ο σχεδιασμός API παραλληλίζει εκείνους τους υψηλούς επίδοσης διακομιστές (servers), επιτρέποντας την εφαρμογή τους έχοντας την ελάχιστη επιβάρυνση σε μια υψηλής απόδοσης κρυφή μνήμη (cache). Ταυτόχρονα, θα παρέχει την υποδομή που απαιτείται για την επεξεργασία των αιτήσεων HTTP και των αποκρίσεων σε υψηλό επίπεδο, προστατεύοντας τους προγραμματιστές από το χαμηλό επίπεδο HTTP και τις λεπτομέρειες υποδοχής και επιτρέποντας σε μονάδες να εκτελούν ενδιαφέρουσες εργασίες χωρίς σημαντικές ποσότητες κώδικα. Έχει εφαρμοστεί αυτό το API στον φορητό και υψηλής απόδοσης iMimic DataReactor μεσολαβητή κρυφής μνήμης.

Στο (Singh et al., 2005) παρουσιάζεται το ORBIT πλαίσιο μέτρησης και η βιβλιοθήκη (OML), το οποίο είναι ένα κατανεμημένο πλαίσιο λογισμικού που επιτρέπει σε πραγματικό χρόνο τη συλλογή των δεδομένων και το πλαίσιο οργάνωσης σε ένα μεγάλο κατανεμημένο περιβάλλον. Επικεντρώνεται στην παροχή κινήτρων, στις απαιτήσεις, στο σχεδιασμό, την εφαρμογή και την πραγματική χρήση του OML που είναι σχεδιασμένο για να προσφέρει έναν επεκτάσιμο, ελεγχόμενο και εύχρηστο μηχανισμό για τους ερευνητές να συλλέγουν χρήσιμα αποτελέσματα από τα πειράματα που πραγματοποιούνται για την πλατφόρμα δοκιμών ORBIT. Το πλαίσιο OML βασίζεται σε αρχιτεκτονική client / server και χρησιμοποιεί το πρωτόκολλο IP πολλαπλής διανομής για τον πελάτη για να υποβάλει μια αναφορά σχετικά με τα δεδομένα που συλλέγονται στο διακομιστή σε πραγματικό χρόνο. Παρουσιάζει την απόδοση της εφαρμογής, καθώς και την εμπειρία που αποκτήθηκε μέσω της OML χρήσης για την ORBIT πλατφόρμα δοκιμών. Εκτός από τη μέτρηση πειραματικών δεδομένων, το OML χρησιμοποιείται για τη συλλογή δεδομένων από τρίτα ασύρματα εργαλεία παρακολούθησης του δικτύου. Το ORBIT σύστημα παρακολούθησης υλικού χρησιμοποιεί επίσης το OML για να συγκεντρώσει και να αναφέρει τις διάφορες παραμέτρους που συνδέονται με τους κόμβους πλατφόρμας δοκιμών.

Το (Sudarsan και Ribbens, 2010) περιγράφει τον σχεδιασμό και την αρχική εφαρμογή ενός πλαισίου λογισμικού για την αξιοποίηση της δυνατότητας αλλαγής μεγέθους σε κατανεμημένη μνήμη παράλληλων εφαρμογών. Με τη δυνατότητα αλλαγής μεγέθους εννοούμε τη δυνατότητα κατά το χρόνο εκτέλεσης για να διευρύνουμε ή να περιορίσουμε τον αριθμό των διεργασιών που συμμετέχουν σε μια παράλληλη εφαρμογή. Η επανασχεδίαση του πλαισίου περιλαμβάνει μια ομάδα χρονοπρογραμματιστή, μια βιβλιοθήκη με υποστηρικτικά στοιχεία ανακατανομής και χαρτογράφηση της διαδικασίας, καθώς και μια διεπαφή προγραμματισμού εφαρμογών (API) που επιτρέπει στις εφαρμογές να αλληλεπιδρούν με τον χρονοπρογραμματιστή και την αλλαγή μεγέθους της βιβλιοθήκης με μικρές μόνο τροποποιήσεις στον κώδικα. Τα πειραματικά αποτελέσματα δείχνουν ότι η επανασχεδίαση πλαισίου μπορεί να βελτιώσει σημαντικά τη συνολική απόδοση του συστήματος, ακόμη και με πολύ απλή εφαρμογή των πολιτικών προγραμματισμού. Επιπλέον, το πλαίσιο λειτουργεί ως μια βολική πλατφόρμα για την έρευνα σε πολύ πιο πολύπλοκες πολιτικές της ομάδας χρονοπρογραμματισμού και τις μεθόδους.

Στο (Carvalho και Cachopo, 2011) περιγράφεται μια θύρα για Java στο σημείο αναφοράς WormBench για να διερευνηθούν οι επιπτώσεις στις επιδόσεις της χαλάρωσης της διαφάνειας του STM(Software Transactional Memory). Η θύρα αυτή επεκτείνει το αρχικό σημείο αναφοράς με διάφορους τρόπους, καθιστώντας το πιο χρήσιμο ως πλατφόρμα δοκιμών για την αξιολόγηση των STMs. Για το σκοπό αυτό, εφαρμόζεται σε ένα καλά γνωστό STM πλαίσιο (Deuce), ένα ζευγάρι των σχολιασμών που επιτρέπει στους προγραμματιστές να διευκρινίσουν ότι ορισμένα αντικείμενα ή τα πεδία των αντικειμένων δεν πρέπει να είναι συναλλάσιμα. Προσδιορίζεται το σύνολο των χαρακτηριστικών που θα θέλαμε να έχουμε σε ένα καλό σημείο αναφοράς για ένα σύστημα STM, και εξετάζεται σε ποιο βαθμό ορισμένα από τα υπάρχοντα σημεία αναφοράς ανταποκρίνονται στις απαιτήσεις αυτές, περιγράφεται με λεπτομέρεια το σημείο αναφοράς WormBench, καθώς και οι κύριες διαφορές που εισάγονται στη θύρα, της εν λόγω αναφοράς σε Java, η οποία ονομάζεται JWormBench. Το άρθρο αναφέρει τις δοκιμασμένες διαμορφώσεις του JWormBench και παρουσιάζεται μια αξιολόγηση των επιδόσεων.

Το άρθρο (Gotlieb και Bernard, 2006) αναφέρεται στη δοκιμή των διεπαφών προγραμματισμού εφαρμογών καρτών Java (API). Το άρθρο αναφέρεται σχετικά με την εμπειρία σε μεθόδους δοκιμών των συστημάτων καρτών, το Oberthur Cosmo 32 RSA Java Card API, χρησιμοποιώντας το παράδειγμα συμμετρικού ελέγχου. Το παράδειγμα εκμεταλλεύεται τις καθορισμένες από το χρήστη ιδιότητες συμμετρίας των Java μεθόδων ως δοκιμή πρόβλεψης. Προτείνεται ένα πειραματικό περιβάλλον που συνδυάζει τυχαία δοκιμή και τον έλεγχο συμμετρίας (με κάρτα) για δοκιμή των διαφόρων μεθόδων καρτών Java API. Έχει αναπτυχθεί ένα ημι-εμπειρικό μοντέλο (ένα μοντέλο που τροφοδοτείται από πειραματικά δεδομένα) για να βοηθήσει να αποφασιστεί πότε θα σταματήσει τις δοκιμές και να αξιολογήσει την ποιότητα δοκιμής, έχει γίνει εισαγωγή ενός πλαισίου δοκιμής του λογισμικού για τον έλεγχο σε κάρτα των συμμετρικών μεθόδων της Java Card του API. Το πλαίσιο περιλαμβάνει ένα ημι-εμπειρικό μοντέλο για να βοηθήσει να αποφασιστεί πότε θα σταματήσει τις δοκιμές και πώς να αξιολογήσει την ποιότητα του ελέγχου, έγινε μια πρώτη εμπειρία στον έλεγχο μερικών μεθόδων του OCS Cosmo 32 RSA V3.4 Java Card του API χρησιμοποιώντας το συμμετρικό παράδειγμα ελέγχου.

Σε αυτό το άρθρο (Bellotti et al., 2008) μελετάμε την τεχνολογία RFID η οποία είναι όλο και πιο δημοφιλής στην ανάπτυξη εφαρμογών πληροφορικής. Στο άρθρο αναλύεται η πλήρης αξιοποίηση των δυνατοτήτων RFID που απαιτούν την μελέτη και την υλοποίηση λεπτομερειών της αλληλεπίδρασης ανθρώπου - υπολογιστή (Human Computer Interaction - HCI) και είναι σε θέση να υποστηρίξει τη δυνατότητα χρήσης. Αυτό συνεπάγεται την ανάγκη για τις μεθοδολογίες προγραμματισμού ειδικά για να υποστηρίξουν την εύκολη και αποτελεσματική προτυποποίηση των εφαρμογών για να έχουν ανατροφοδότηση από τις πρώτες δοκιμές με τους χρήστες, έχει σχεδιαστεί το oDect, μια υψηλού επιπέδου γλώσσα και μια ανεξάρτητη πλατφόρμα διεπαφής προγραμματισμού εφαρμογών (API). Επιπλέον, η προσέγγιση αναπτύσσεται σε έναν τελικό χρήστη (End User Developing - EUD), το oDect παρέχει συγκεκριμένη στήριξη για την εφαρμογή του τελικού χρήστη με τα τυπικά προβλήματα των εφαρμογών RFID για την ανίχνευση αντικειμένων. Περιγράφονται τα χαρακτηριστικά του API και εξετάζονται τα πορίσματα της δοκιμής με τέσσερις προγραμματιστές, όπου αναλύεται και αξιολογείται η χρήση του API σε τέσσερα δείγματα εφαρμογών. Επίσης,

παρουσιάζονται τα αποτελέσματα της δοκιμής του τελικού χρήστη, τα οποία διερεύνησαν δυνατά και αδύνατα σημεία της έννοιας Territorial Agenda (TA). Η TA είναι ένα RFID με βάση τον οδηγό όπου ενισχύσεις-μέσα στο χρόνο-και στη θέση-βασισμένες σε υπενθυμίσεις χρηστών στις καθημερινές τους δραστηριότητες σε μια πόλη.

Το τελευταίο άρθρο αυτής της κατηγορίας ασχολείται με την εφαρμογή των πλαισίων διαδικτυακής εφαρμογής. Το (Shan και Hua, 2006) περιγράφει τα διάφορα πλαίσια των διαδικτυακών εφαρμογών που σχετίζονται με τις νέες τεχνολογίες με το μοντέλο της Java EE από τεχνική άποψη. Ένα πλαίσιο διαδικτυακής εφαρμογής παρουσιάζεται για να δείξει πώς ένα πλαίσιο μπορεί να βελτιώσει την παραγωγικότητα ανάπτυξης εφαρμογών και την ποιότητα. Υπάρχουν πέντε μεγάλες κατηγορίες των πλαισίων διαδικτυακής εφαρμογής: το πλαίσιο βασισμένο στο αίτημα, το πλαίσιο βασισμένο στο περιεχόμενο, το Hybrid πλαίσιο, το Μετά-πλαίσιο και το πλαίσιο βασισμένο στο RIA (Rich Internet Application). Το πλαίσιο με βάση το αίτημα είναι πολύ κοντά στην αρχική CGI προδιαγραφή. Χρησιμοποιεί τους ελεγκτές και τις δράσεις που διαχειρίζονται άμεσα τα εισερχόμενα αιτήματα. Ένα πλαίσιο που βασίζεται στο περιεχόμενο αφαιρεί τα εσωτερικά του χειρισμού αιτήματος και ενσωματώνει τη λογική σε επαναχρησιμοποιήσιμα στοιχεία, συχνά ανεξάρτητα από το διαδίκτυο. Το Hybrid πλαίσιο συνδυάζει και τα δυο, το πλαίσιο με βάση το αίτημα και το πλαίσιο με βάση το περιεχόμενο, αναλαμβάνοντας τον έλεγχο του συνόλου των δεδομένων και τη λογική ροή σε ένα μοντέλο με βάση το αίτημα. Το Μετά-Πλαίσιο έχει ένα σύνολο διεπαφών πυρήνα για τις κοινές υπηρεσίες και ιδιαίτερα επεκτάσιμη στήριξη για την ενσωμάτωση των στοιχείων και των υπηρεσιών. Το πλαίσιο βασισμένο στο RIA αναφέρεται σε μια ιστοσελίδα βασισμένη στην εφαρμογή που εκτελείται σε ένα πρόγραμμα περιήγησης με πλούσια χαρακτηριστικά διεπαφής χρήστη.

2.3.1.2 Ποιότητα των Διασυνδέσεων Προγραμματισμού Εφαρμογών (APIs)

Η δεύτερη κατηγορία περιλαμβάνει άρθρα που ασχολούνται με την ποιότητα των Διασυνδέσεων Προγραμματισμού Εφαρμογών (APIs). Χαρακτηριστικά που αξιολογούν την ποιότητα των APIs είναι η ευκολία κατά τη συντήρηση και την εφαρμογή αλλαγών, η δυνατότητα επαναχρησιμοποίησης και διενέργειας ελέγχων,

η προσαρμοστικότητα, η επεκτασιμότητα, η σταθερότητα, η ευελιξία, η εύκολη κατανόηση, η χρηστικότητα κ.α. Με βάση αυτά τα ποιοτικά χαρακτηριστικά παρουσιάζονται τα 6 άρθρα αυτής της κατηγορίας. Στα άρθρα (Stylos et al., 2008, Kim et al., 2011, Stylos και Myers, 2008, Hoffman και Strooper, 2000) μελετάμε τη δυνατότητα επαναχρησιμοποίησης των APIs, τη βελτίωση της χρηστικότητας των APIs, καθώς επίσης και τον ρόλο των επανασχεδιάσεων στην ποιότητα του λογισμικού.

Το (Stylos et al., 2008) περιγράφει τη σχεδίαση με επίκεντρο τον χρήστη και τη διαδικασία αξιολόγησης που εξελίχθηκε σε επανασχεδιασμό του BRFplus(Business Rule Framework plus) του SAP - μια μηχανή επιχειρησιακών κανόνων, της οποίας το API δημιουργήθηκε για την ανάπτυξη πλατφόρμας. Το API BRFplus εφαρμόζεται στη γλώσσα προγραμματισμού ABAP. Χρησιμοποιείται αυτό το API ως το επίκεντρο της περίπτωσης μελέτης κατά την οποία θα αναπτυχθεί και θα εφαρμοσθεί μια διαδικασία για τη μελέτη και τη βελτίωση της χρηστικότητας του API για την παροχή ενός κοινού πλαισίου που τυποποιεί το πλαίσιο των επιχειρησιακών κανόνων για όλους τους SAP. Το API BRFplus επιτρέπει τη δημιουργία και την επεξεργασία των επιχειρησιακών κανόνων. Το API που εξετάστηκε είχε ήδη κυκλοφορήσει και χρησιμοποιηθεί, δεν θα μπορούσε να δημιουργηθεί απλά μια νέα έκδοση και να αγνοηθεί το παλιό API, επιλέχτηκε η σχεδίαση ενός “wrapper API,” ένα υψηλότερου επιπέδου API που εφαρμόζεται στην κορυφή του αρχικού API. Αυτό μπορεί να δώσει επίσης τη δυνατότητα στους προγραμματιστές να επιλέξουν με ποιο επίπεδο διακριτότητας θέλουν να αλληλεπιδρούν με το API ή το wrapper API.

Το (Kim et al., 2011) παρουσιάζει μια εμπειρική έρευνα για το ρόλο των επανασχεδιάσεων σε επίπεδο API κατά την εξέλιξη του λογισμικού. Η επανασχεδίαση είναι η διαδικασία αλλαγής της δομής του σχεδιασμού ενός προγράμματος, χωρίς τη μεταβολή της εξωτερικής λειτουργικής συμπεριφοράς του προκειμένου να βελτιωθεί η αναγνωσιμότητα του προγράμματος, η συντήρηση και η επεκτασιμότητα. Διαπιστώθηκε ότι ο αριθμός των διορθώσεων σφάλματος αυξάνεται μετά τις επανασχεδιάσεις ενώ ο χρόνος που απαιτείται για να επιλυθούν τα σφάλματα μειώνεται μετά τις επανασχεδιάσεις. Ποσοτική και ποιοτική μελέτη της προσέγγισης των επανασχεδιάσεων σε επίπεδο API και τις διορθώσεις

σφαλμάτων σε τρία μεγάλα έργα ανοικτού κώδικα, το Eclipse JDT, το jEdit, και το Columba συνολικού ύψους 26523 αναθεωρήσεων. Ο στόχος του άρθρου είναι να διερευνήσει συστηματικά το ρόλο της επανασχεδίασης κατά τη διάρκεια της εξέλιξης του λογισμικού, εξετάζοντας τις σχέσεις μεταξύ των επανασχεδιάσεων, τις διορθώσεις σφαλμάτων, το χρόνο για την επίλυση των σφαλμάτων και την έκδοση. Πρώτα, εφαρμόστηκε η τεχνική ανασυγκρότησης της επανασχεδίασης, συγκρίνει στοιχεία του κώδικα όσον αφορά την ονομασία τους και την ομοιότητα της δομής για να προσδιορίσει τη μετονομασία, τη μετακίνηση και τις αλλαγές στην υπογραφή ή πάνω από το επίπεδο των επικεφαλίδων της μεθόδου. Δεύτερον, εφαρμόστηκε η τεχνική της εξαγωγής σφάλματος για τον εντοπισμό των αναθεωρήσεων διόρθωσης σφαλμάτων.

Στο άρθρο (Stylos και Myers, 2008) οι μεγάλες αντικειμενοστρεφείς διασυνδέσεις προγραμματισμού εφαρμογών (APIs), τα πλαίσια, οι βιβλιοθήκες της Java JDK και της Microsoft .NET προσφέρουν την δυνατότητα για τη βελτίωση της παραγωγικότητας των προγραμματιστών, παρέχοντας πρόσβαση σε χιλιάδες κλάσεις. Πρόσφατες έρευνες έχουν αρχίσει να μελετούν τις στρατηγικές των προγραμματιστών και τη χρήση των APIs, διαπιστώθηκε ότι η τοποθέτηση της μεθόδου μπορεί να έχει μεγάλο αντίκτυπο στη χρηστικότητα των αντικειμενοστρεφών APIs. Εναλλακτικές εκδόσεις των τριών διαφορετικών APIs συγκρίθηκαν και διαπιστώθηκε ότι οι προγραμματιστές κλίνουν προς τις ίδιες κλάσεις και ήταν δραματικά πιο γρήγορες - από 2 έως 11 φορές - συνδυάζοντας πολλαπλά αντικείμενα, όταν μια μέθοδος στην αρχική κλάση αναφέρεται σε άλλη κλάση. Παρουσιάζονται δυο καταστάσεις, η κατάσταση Α αντιπροσωπεύει το API που βρίσκεται πλησιέστερα στο πραγματικό API (εάν υπάρχει), στην οποία η εργασία απαιτεί τη χρήση ενός αντικειμένου που δεν αναφερόταν από την κλάση που αναμένεται να βρεθεί ως ένα σημείο εκκίνησης. Η κατάσταση Β αντιπροσωπεύει το "σταθερό" API, στο οποίο η κλάση αναμένεται να χρησιμοποιηθεί ως αρχική κλάση που περιείχε μια μέθοδο αναφοράς στη βοηθητική κλάση.

Το (Hoffman και Strooper, 2000) αναφέρεται στην έλευση των αντικειμενοστρεφών γλωσσών και τη φορητότητα των Java APIs, η ανάπτυξη και η χρήση των επαναχρησιμοποιήσιμων στοιχείων λογισμικού γίνεται πραγματικότητα.

Σε αυτό το άρθρο, παρουσιάζεται το εργαλείο Roast και οι τεχνικές για τον έλεγχο των APIs της Java. Το εργαλείο και οι τεχνικές που φαίνονται στα δύο μη-τετριμμένα στοιχεία και τα ποσοτικά αποτελέσματα που παρουσιάζονται για να τεκμηριώσουν την πρακτικότητα και την αποτελεσματικότητα της προσέγγισης. Ενσωματώνει αυτές σε ένα απλό εργαλείο Java και περιέχει σημαντικές βελτιώσεις στην παραγωγή κώδικα και την παραγωγή των οριακών τιμών. Το άρθρο δείχνει αυτή τη προσέγγιση με ένα οδηγό δοκιμής για ένα ADT που παρέχει πρόσβαση σε τρεις ορθογώνιους πίνακες. Επίσης παρουσιάζει τις εξαρτώμενες περιοχές και μια νέα επέκταση του συστήματος παραγωγής οριακών τιμών, περιγράφει μια προσέγγιση δοκιμής που βασίζεται σε μοντελοποίηση κάθε περίπτωσης δοκιμής με μια πλειάδα και στη συνέχεια τη δημιουργία μεγάλου αριθμού των πλειάδων για να καλύψει πλήρως ένα χώρο εισόδου με πολλούς ενδιαφέροντες συνδυασμούς των τιμών.

Σε αυτό το άρθρο μελετάμε δυο μοντέλα, που υποστηρίζουν τη λειτουργία της πλατφόρμας δοκιμής. Το άρθρο (Liu et al., 2009) μελετά τα χαρακτηριστικά των πλατφόρμων δοκιμής στους ελέγχους του λογισμικού και αναλύει τη δομή τους. Μια νέα πλατφόρμα δοκιμής του πλαισίου συνεργασίας παρουσιάζεται σε αυτή τη δοκιμή του λογισμικού που βασίζεται στην υπάρχουσα πλατφόρμα δοκιμής. Υπάρχουν δύο μοντέλα που υποστηρίζουν τη λειτουργία μεταξύ των πλατφόρμων δοκιμής. Αυτά τα δύο μοντέλα είναι το μοντέλο της οντολογίας και το μοντέλο της ταξινόμησης. Το μοντέλο της οντολογίας παρέχει την επίλυση των συγκρούσεων ή διαφορετικές αναλύσεις μεταξύ των πλατφόρμων των δοκιμών, οι οποίες έχουν την ίδια λειτουργία ή σχετικές λειτουργίες. Το μοντέλο της ταξινόμησης είναι για τη βελτίωση σε επίπεδο συνεργασίας για παρόμοιες κατηγορίες λογισμικού. Το πλαίσιο μπορεί να αναλύσει τις διαφορετικές πλατφόρμες δοκιμής με το μοντέλο της οντολογίας και της ταξινόμησης, καθώς και την ενίσχυση της συνεργασίας μεταξύ των πλατφόρμων των δοκιμών σε κάποιο βαθμό.

Στο τελευταίο άρθρο αυτής της κατηγορίας μελετάμε τους παράγοντες που επηρεάζουν την χρηστικότητα του API. Το (Zibran et al., 2011) αναφέρει ότι η εξέλιξη του λογισμικού σήμερα εξαρτάται σε μεγάλο βαθμό από τη χρήση των API βιβλιοθηκών, τα πλαίσια και τα επαναχρησιμοποιήσιμα στοιχεία. Έχουν εντοπιστεί

οι 22 παράγοντες που επηρεάζουν την χρηστικότητα του API, που αποτυπώνονται στα μηνύματα σφάλματος από τους χρήστες του API και διακρίνουν τη σημασία των παραγόντων χρηστικότητας. Έγινε έρευνα των 1.513 αναφορών λάθους από τα συστήματα παρακολούθησης λαθών των πέντε διαφορετικών έργων ανοικτού πηγαίου κώδικα του Eclipse, GNOME, MySQL, Python 3.1 και Android. Οι παράγοντες ευχρηστίας του API κατατάσσονται με βάση τη ρεαλιστική σημασία τους.

2.3.1.3 Ανάλυση των Διασυνδέσεων Προγραμματισμού Εφαρμογών (APIs)

Σε αυτή τη κατηγορία ανήκουν 23 μελέτες οι οποίες σχετίζονται με την ανάλυση των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών. Στα άρθρα που ακολουθούν μελετούνται ζητήματα επανασχεδίασης, αναφορές τεκμηρίωσης, αντικειμενοστρεφή μοντέλα, μετρικές κ.α. Πιο συγκεκριμένα στα (Sarkar et al., 2007, Tselikas et al., 2007, Fujiwara et al., 2003 και Ratiu και Jurjens, 2008) παρουσιάζονται ζητήματα που σχετίζονται με τις μετρικές για τις διασυνδέσεις προγραμματισμού εφαρμογών.

Το (Sarkar et al., 2007) παρουσιάζει μια νέα σειρά από μετρικές που μετρούν την ποιότητα διαμόρφωσης ενός μη αντικειμενοστρεφούς συστήματος λογισμικού. Προτείνεται ένα σύνολο αρχών σχεδιασμού για να συλλάβουν την έννοια της διαμόρφωσης και ορίζονται μετρικές που επικεντρώνονται γύρω από αυτές τις αρχές. Οι μετρικές αυτές χαρακτηρίζουν το λογισμικό από ποικίλες οπτικές γωνίες: δομικές, αρχιτεκτονικές και έννοιες όπως η ομοιότητα του σκοπού και των κοινών στόχων. Οι υπόλοιπες μετρικές που παρουσιάζονται είναι υπέρ εκείνων που βασίζονται στο API. Μερικές από τις σημαντικές μετρικές υποστήριξης περιλαμβάνουν εκείνες που χαρακτηρίζουν κάθε μονάδα με βάση την ομοιότητα του σκοπού των υπηρεσιών που προσφέρονται από τη μονάδα. Δοκιμάστηκαν οι μετρικές σε μερικά δημοφιλή συστήματα ανοικτού πηγαίου κώδικα και ορισμένες μεγάλου κώδικα επιχειρηματικές εφαρμογές. Για να επικυρωθούν οι μετρικές, συγκρίθηκαν με τα αποτελέσματα που προέκυψαν σχετικά με τις ανθρώπινες – ρυθμιζόμενες εκδόσεις του λογισμικού με εκείνες που λαμβάνονται σε τυχαιοποιημένες εκδόσεις του κώδικα.

Το (Tselikas et al., 2007) περιγράφει την εφαρμογή της πύλης ελέγχου κλήσεων στο πλαίσιο της εφαρμογής προώθησης κλήσης. Η εφαρμογή βασίζεται

σε ανοικτές διεπαφές εκθέτοντας τις λειτουργίες δικτύου ελέγχου κλήσεων. Πρώτον, μια εφαρμογή OSA / Parlay με RMI και CORBA ως βασική αρχιτεκτονική της επικοινωνίας και στη συνέχεια μια JAIN (**J**ava **A**PIs for **I**ntegrated **N**etworks) εφαρμογή που χρησιμοποιεί RMI με βάση την εφαρμογή του API ελέγχου κλήσεων. Μια μελέτη αξιολόγησης των επιδόσεων έγινε βάσει των μετρήσεων και οι δύο υλοποιήσεις συγκρίθηκαν σε σχέση με τις μετρικές απόδοσης, όπως τη μέση απόδοση και τη μέση καθυστέρηση. Το πεδίο εφαρμογής του άρθρου είναι να ερευνήσει και να συγκρίνει την επίδραση στην απόδοση ενσωματώνοντας λειτουργίες ελέγχου κλήσεων δικτύου. Όλες οι μετρήσεις των επιδόσεων λαμβάνονται στο πλαίσιο σε πραγματικό χρόνο εκτέλεσης υπηρεσίας προώθησης κλήσεων, που μεταφράστηκε από μια αντίστοιχη εφαρμογή με βάση το VoIP SIP.

Το (Fujiwara et al., 2003) αξιολογεί τη χρησιμότητα ενός τομέα συγκεκριμένου πλαισίου εφαρμογής των επιχειρήσεων από την άποψη της εξοικονόμησης κόστους και της ποιότητας του λογισμικού σε μια εταιρεία. Διεξήχθησαν δύο μελέτες περίπτωσης. Στις μελέτες περιπτώσεων αναπτύσσονται τέσσερα είδη εφαρμογών. Καθένα από αυτά έχει αναπτυχθεί με δύο τρόπους: με βάση την επαναχρησιμοποίηση πλαισίου και με βάση την επαναχρησιμοποίηση των συμβατικών μονάδων. Στη συνέχεια, αξιολογείται η διαφορά μεταξύ τους χρησιμοποιώντας αρκετή λειτουργικότητα και μετρικές πολυπλοκότητας. Τα αποτελέσματα, με βάση την επαναχρησιμοποίηση πλαισίου θα ήταν πιο αποτελεσματικά από την μονάδα που βασίζεται στην επαναχρησιμοποίηση.

Το (Ratiu και Jurjens, 2008) αναφέρεται στις βιβλιοθήκες ως την πιο διαδεδομένη μορφή της επαναχρησιμοποίησης του λογισμικού, η χρηστικότητα των APIs επηρεάζει σημαντικά την παραγωγικότητα των προγραμματιστών σε όλες τις φάσεις της ανάπτυξης λογισμικού. Στο άρθρο αυτό έχει αναπτυχθεί ένα πλαίσιο για να περιγράψει την εφαρμογή των εννοιών σε ένα API σε δύο κατευθύνσεις: τον τρόπο με τον οποίο αναφέρονται και τον τρόπο με τον οποίο αναπαρίστανται στο API. Ορίζονται μετρικές που επιτρέπουν στον προγραμματιστή του API την αξιολόγηση της εννοιολογικής πολυπλοκότητας του API, τη μη ομοιομορφία και τις ασάφειες που εισήχθησαν με εσωτερικές αναπαραστάσεις του API των εννοιών χώρου, πράγμα που καθιστά την ανάπτυξη

και τη συντήρηση του λογισμικού που χρησιμοποιεί η βιβλιοθήκη, δύσκολη και επιρρεπή σε λάθη.

Τα άρθρα (Farooq et al., 2010, Rao και Kak, 2011, Kawrykow και Robillard, 2009) ασχολούνται με την ανίχνευση και διόρθωση σφαλμάτων, καθώς επίσης παρουσιάζονται και αλγόριθμοι εντοπισμού σφαλμάτων. Το (Farooq et al., 2010) συγκρίνει ατέλειες της χρηστικότητας που διαπιστώθηκαν μεταξύ των αναθεωρήσεων της ίδιας χρηστικότητας API και των API δοκιμών ευχρηστίας, τα API δοκιμών ευχρηστίας βρέθηκαν να εκθέτουν τα ζητήματα σχεδιασμού που σχετίζονται πραγματικά χρησιμοποιώντας ένα API ενώ οι αναθεωρήσεις ίδιας χρηστικότητας API βρέθηκαν να εκθέτουν το σκεπτικό του σχεδιασμού ενός API. Εξετάστηκε η αποτελεσματικότητα και η παραγωγικότητα της κάθε μεθόδου, κάθε δοκιμή χρηστικότητας API είναι ισοδύναμη με περίπου 16 αξιολογήσεις ίδιας χρηστικότητας API.

Το (Rao και Kak, 2011) αναφέρεται στην ανάκτηση από τις μεγάλες βιβλιοθήκες λογισμικού για τον σκοπό του εντοπισμού σφαλμάτων, συγκρίνονται πέντε μοντέλα κείμενου και ορισμένες σύνθετες παραλλαγές τους. Τα γενικά μοντέλα είναι: το Unigram Model (UM), το Vector Space Model (VSM), το Latent Semantic Analysis Model (LSA), το Latent Dirichlet Allocation Model (LDA), και το Cluster Based Document Model (CBDM). Ο στόχος είναι να εντοπιστούν τα αρχεία που σχετίζονται με ένα σφάλμα που αναφέρθηκε με τη μορφή μιας περιγραφής κειμένου από την ανάπτυξη του λογισμικού. Χρησιμοποιείται το iBUGS, ένα υποδειγματικό σύνολο δεδομένων εντοπισμού σφαλμάτων. Οι τεχνικές εντοπισμού σφαλμάτων που αναπτύχθηκαν στο παρελθόν, μπορούν να τοποθετηθούν σε δύο κατηγορίες: εκείνες που πραγματοποιούν δυναμική ανάλυση της συμπεριφοράς εκτέλεσης του προγράμματος, προκειμένου να εντοπίσουν ένα σφάλμα και εκείνων που βασίζονται σε μια στατική ανάλυση του κώδικα ενός συστήματος λογισμικού, συνήθως λειτουργεί σε επίπεδο βαθμού κατάτμησης μιας κλάσης.

Το (Kawrykow και Robillard, 2009) ερευνά έργα λογισμικού που συχνά βασίζονται σε βιβλιοθήκες που τα καθιστούν προσβάσιμα μέσω των διεπαφών προγραμματισμού εφαρμογών (APIs). Έχει αναπτύξει μια τεχνική και ένα εργαλείο υποστήριξης για την αυτόματη ανίχνευση αυτών των προτύπων από τη χρήση του

API σε έργα λογισμικού. Η κύρια υπόθεση που διέπει την τεχνική είναι ότι ο κώδικας πελάτη μιμείται τη συμπεριφορά μιας μεθόδου API χωρίς να μπορεί να χρησιμοποιεί το API αποτελεσματικά, διότι θα μπορούσε να καλέσει τη μέθοδο αντί να τη μιμείται. Η τεχνική περιλαμβάνει την ανάλυση των συστημάτων λογισμικού για τον εντοπισμό περιπτώσεων απομιμήσεων μεθόδου του API. Εκτός από την προειδοποίηση των προγραμματιστών της ενδεχόμενης επαναυλοποίησης μεθόδων του API, μπορεί επίσης να αναφερθεί πώς μπορεί να βελτιωθεί με τη χρήση του API.

Τρία άρθρα αυτής της κατηγορίας τα (Perkins, 2005, Xing και Stroulia, 2007, Gharaibeh et al., 2007) ασχολούνται με την επανασχεδίαση των APIs. Το (Perkins, 2005) αναφέρει ότι οι αλλαγές είναι συνεχείς, μεταξύ 60% έως 90% της ανάπτυξης του λογισμικού αποτελείται από τη «συντήρηση», ή την τροποποίηση του υπάρχοντος λογισμικού. Το άρθρο προτείνει μια νέα τεχνική που εκμεταλλεύεται το πλεονέκτημα της πληροφορίας, οι προγραμματιστές μπορούν να εισάγουν στον κώδικα παρά να τους αναγκάζουν να χρησιμοποιήσουν ένα διαφορετικό εργαλείο για να το εκφράσουν εκ νέου. Το άρθρο εξετάζει τις αλλαγές στις βιβλιοθήκες, πώς να ρυθμιστεί ο κώδικας του πελάτη σε τέτοιες αλλαγές χωρίς να ενοχλεί είτε τους συγγραφείς της βιβλιοθήκης ή τους πελάτες. Οι περισσότερες απαρχαιωμένες μέθοδοι πρόκειται να αντικατασταθούν από τον νέο κώδικα με παρόμοιο τρόπο και ισοδύναμη ή ανώτερη λειτουργικότητα. Η βασική ιδέα είναι να αντικατασταθούν κλήσεις προς απαρχαιωμένες μεθόδους από τα σώματά τους, όπου τα σώματα αυτά αποτελούν τον κατάλληλο κώδικα της αντικατάστασης. Οι συντηρητές της βιβλιοθήκης γράφουν την τεκμηρίωση, την περιγραφή και συχνά ένα παράδειγμα της αντικατάστασης των κλήσεων προς την απαρχαιωμένη μέθοδο από τις κλήσεις προς τις αντικαταστάσεις. Η τεχνική αυτή ικανοποιεί βασικούς στόχους που υποστηρίζει την περαιτέρω επαναχρησιμοποίηση των υπαρχόντων αντικειμένων όταν αυτό είναι δυνατόν.

Το (Xing και Stroulia, 2007) περιγράφει εφαρμογές που στηρίζονται σε επαναχρησιμοποιήσιμα στοιχεία πλαισίων που υπόκεινται σε δύο ανεξάρτητες και ενδεχομένως αλληλοσυγκρουόμενη εξέλιξη των διαδικασιών. Η εξέλιξη στοιχείου του πλαισίου οδηγείται από την ανάγκη να βελτιωθεί το πλαίσιο λειτουργικότητας και η ποιότητα, διατηρώντας παράλληλα τη γενικότητά του. Παρουσιάζεται μια

προσέγγιση για την αντιμετώπιση του προβλήματος της εξέλιξης του API στο πλαίσιο της ανάπτυξης λογισμικού με βάση την επαναχρησιμοποίηση, η οποία αναγνωρίζει αυτόματα τις αλλαγές στο API του επαναχρησιμοποιήσιμου πλαισίου και προτείνει λογικές αντικαταστάσεις για το απαρχαιωμένο API που βασίζεται σε παραδείγματα εργασίας βάσης κώδικα του πλαισίου. Αυτή η προσέγγιση έχει εφαρμοστεί στο εργαλείο Diff-CatchUp. Η προσέγγιση για την εξέλιξη του API βασίζεται στις εργασίες σχετικά με τον αλγόριθμο UMLDiff. Με βάση τις αλλαγές στο API ότι η επαναχρησιμοποίηση του πλαισίου αναγνωρίζεται αυτόματα από τον αλγόριθμο UMLDiff η προσέγγιση χρησιμοποιεί ένα σύνολο ευρετικών αξιολογήσεων για να συμπεράνει πιθανές αντικαταστάσεις για τη παραβίαση του API που προκαλεί το πρόβλημα της μετάβασης στο API και εξετάζει τη βάση κώδικα που στηρίζεται στο εξελιγμένο πλαίσιο για να επιλέξει παραδείγματα για το πώς χρησιμοποιούνται οι πιθανές αντικαταστάσεις.

Το (Gharaibeh et al., 2007) παρουσιάζει ότι η πλειοψηφία της συμβατότητας - έκτακτες αλλαγές σε ένα βασισμένο σε στοιχείο, αντικειμενοστρεφές σύστημα λογισμικού είναι οι επανασχεδιάσεις. Η διαδικασία αναβάθμισης του λογισμικού σε ένα τέτοιο σύστημα με την παρουσία σε μεγάλο βαθμό της επανασχεδιάσιμης διεπαφής προγραμματισμού εφαρμογών (API) είναι σε μεγάλο βαθμό το εγχειρίδιο και η αποδιοργάνωση στην εκτέλεση, οι κρίσιμες εφαρμογές οι οποίες αναμένεται να εκτελούνται συνεχώς χωρίς διακοπές. Προκειμένου να αντιμετωπίσει την απευθείας σύνδεση, API ενημέρωση για το ζήτημα συστημάτων κρίσιμης σημασίας, έχει αναπτυχθεί μια απευθείας ενημέρωση πλαισίου που βασίζεται σε Εικονικό Περιβάλλον Εκτέλεσης (VEE- Virtual Execution Environment), όπως η εικονική μηχανή της Java. Το πλαίσιο επεκτείνει το VEE για να αναλάβει το αρχείο καταγραφής αλλαγών API, το οποίο καταγράφει τις αλλαγές στα API στοιχεία που επιβάλλουν τις κατάλληλες ενημερώσεις για το λειτουργικό σύστημα χωρίς ανθρώπινη παρέμβαση και χωρίς διακοπή της λειτουργίας του συστήματος.

Τα άρθρα (Ellis et al., 2007, Matsuzawa και Ikeda, 1998, Ren et al., 2011) παρουσιάζουν μερικά αντικειμενοστρεφή μοντέλα. Πιο συγκεκριμένα το (Ellis et al., 2007) εξετάζει τις επιπτώσεις χρηστικότητας ενός από τα πιο γνωστά πρότυπα αντικειμενοστρεφούς σχεδίασης: το πρότυπο αφηρημένο εργοστάσιο. Το πρότυπο του αφηρημένου εργοστασίου μπορεί να χρησιμοποιηθεί για τη στενή διαχείριση

της κατανομής και για την διαδικασία αρχικοποίησης, αφού ένα αφηρημένο εργοστάσιο δεν πρέπει οπωσδήποτε να διαθέσει ένα νέο αντικείμενο κάθε φορά που ζητείται. Το άρθρο εστιάζει στην ευχρηστία των APIs που χρησιμοποιούν το πρότυπο εργοστάσιο, ενώ η τελευταία έρευνα σχετικά με το εργοστάσιο έχει ως επί το πλείστον επικεντρωθεί στα αρχιτεκτονικά πλεονεκτήματα του προτύπου εργοστασίου για σχεδιαστές του συστήματος.

Το (Matsuzawa και Ikeda, 1998) αναφέρεται σε ένα αντικειμενοστρεφές μοντέλο αναφοράς, το MVIm μοντέλο (Model-View-Interface manager) με διάφορα εργαλεία ενδιάμεσου λογισμικού, το οποίο περιλαμβάνει διάφορα αντικειμενοστρεφή προϊόντα ενδιάμεσου λογισμικού, όπως αντικειμενοστρεφή βάση δεδομένων (OODB), GUI βιβλιοθήκες, και το CORBA. Το μοντέλο MVIm, ως βασικό μοντέλο της αρχιτεκτονικής του λογισμικού εξεταζόμενο στα προβλήματα της σύνθεσης ενδιάμεσου λογισμικού. Το MVIm μοντέλο που βασίζεται στο μοντέλο MVC (Model-View-Controller) το οποίο είναι ένα από τα πιο διάσημα μοντέλα αναφοράς. Η κατηγορία Μοντέλο αντιστοιχεί σε όλες τις διατηρούμενες κατηγορίες κληρονομώντας τις OODB βιβλιοθήκες και η κατηγορία Προβολή αντιστοιχεί σε όλες τις κλάσεις κληρονομώντας τις GUI βιβλιοθήκες εκτός της κλάσης του Ελεγκτή, η οποία ανήκει στην κατηγορία Ελεγκτή.

Το (Ren et al., 2011) αναφέρει ότι η πλατφόρμα ανάπτυξης του λογισμικού βασίζεται στον ανοιχτό πηγαίο κώδικα του SSH (Spring, Struts, Hibernate) πλαισίου. Με βάση την ανάλυση των υπαρχόντων προβλημάτων της ανάπτυξης λογισμικού, μελετά τις βασικές τεχνικές της πλατφόρμας του SSH πλαισίου, συμπεριλαμβανομένου του πλαισίου Struts στο επίπεδο της παρουσίασης, το επιχειρηματικής λογικής πλαίσιο Spring, το επίπεδο διατήρησης δεδομένων του πλαισίου Hibernate και το J2EE πλαίσιο που ενσωματώθηκε στο νέο SSH. Υπάρχει κάποια εφαρμογή ανοιχτού πηγαίου κώδικα πλαισίου που βασίζεται σε J2EE, στην οποία οι βασικές τεχνολογίες του πλαισίου είναι τα Struts πλαίσια που βασίζονται στο πρότυπο MVC και το Spring πλαίσιο με βάση το πρότυπο IoC και το αντικείμενο / αντιστοίχηση των σχέσεων του πλαισίου Hibernate.

Τα άρθρα (Watson, 2009, Berglund, 2003, Rupakheti και Hou 2011, Wu et al., 2010, Hou και Yao, 2011, Dekel και Herbsleb, 2009) ασχολούνται με θέματα τεκμηρίωσης των APIs. Το (Watson, 2009) περιγράφει πώς η εφαρμογή των

τεχνικών ικανοτήτων επικοινωνίας και τα εργαλεία που συνέβαλαν στη βελτίωση της χρηστικότητας και της σαφήνειας μια νέας διεπαφής προγραμματισμού εφαρμογών (API), πραγματοποιώντας μια ανάλυση κειμένου από τα στοιχεία του API. Η μελέτη περίπτωσης παρουσιάζει τη θεωρία πάνω στην οποία η προσέγγιση αυτή στηρίζεται και περιγράφει το πώς η θεωρία αυτή εφαρμόστηκε για να αναλύσει ένα συγκεκριμένο API. Το άρθρο καταλήγει με μια ανασκόπηση του πώς αυτή η μέθοδος ανάλυσης μπορεί να μεταφερθεί σε άλλα έργα και πώς τα εργαλεία που χρησιμοποιούνται στην παρούσα ανάλυση μπορούν να εφαρμοστούν προς όφελος του σχεδιασμού, της ανάπτυξης και των διαδικασιών τεκμηρίωσης του API.

Το (Berglund, 2003) παρουσιάζει μία μελέτη του σχεδιασμού της ηλεκτρονικής αναφοράς της τεκμηρίωσης για βιβλιοθήκες στοιχείου του λογισμικού. Τα αποτελέσματα προέρχονται από μια μελέτη σε ένα βιομηχανικό περιβάλλον που βασίζεται στη χρήση μιας πειραματικής ηλεκτρονικής τεκμηρίωσης αναφοράς (που ονομάζεται Dynamic Javadoc ή DJavadoc) που χρησιμοποιούνται σε ένα πραγματικό έργο κατάστασης για 4 μήνες. Τα αποτελέσματα από τις συνεντεύξεις με τους προγραμματιστές δείχνουν ότι η ηλεκτρονική βιβλιοθήκη αναφοράς της τεκμηρίωσης δεν απαιτεί προσαρμογή ή την εξέλιξη σε ατομικό επίπεδο. Το πιο σημαντικό, η αναφορά της τεκμηρίωσης θα πρέπει να διευκολύνει τη μεταφορά του κώδικα από έγγραφα στα αρχεία προέλευσης και επίσης να υποστηρίξει την ενσωμάτωση πολλαπλών πηγών τεκμηρίωσης.

Το (Rupakheti και Hou 2011) αναφέρει ότι οι προγραμματιστές μπορούν να χρησιμοποιήσουν την αναζήτηση με βάση τα εργαλεία για τον εντοπισμό των αποσπασμάτων κώδικα ή εφαρμογές που μπορεί να σχετίζονται με τα APIs που χρησιμοποιούν. Έχει εξεταστεί ένα δυναμικό σύστημα βοήθειας που είναι ενσωματωμένο σε ένα περιβάλλον ανάπτυξης για να παρέχει σχετιζόμενες προτάσεις στους προγραμματιστές, όπως η ανάγνωση του κώδικα και η επεξεργασία σε πρόγραμμα επεξεργασίας. Η κύρια λειτουργία των εργαλείων αναζήτησης είναι να βρουν αποσπάσματα κώδικα και έγγραφα με βάση λέξεις-κλειδιά. Εξετάστηκαν τα πιθανά προβλήματα της επαναχρησιμοποίησης του λογισμικού που προκαλούνται από το πληροφοριακό χάσμα μεταξύ του μοντέλου

κατάστασης και του μοντέλου συστήματος. Το μοντέλο κατάστασης είναι το νοητικό μοντέλο ενός προγραμματιστή σχετικά με την εργασία ή το πρόβλημα που χρειάζεται να επιλυθεί, και αυτό είναι συχνά ασαφές και άτυπο. Το μοντέλο συστήματος, από την άλλη πλευρά, είναι για την πραγματική λειτουργία του συστήματος. Υπάρχουν δύο είδη των πηγών πληροφοριών: οι αρχικοί σχεδιαστές API και οι έμπειροι χρήστες των APIs, με βάση την εμπειρία τους, μπορούν να εντοπίσουν και να παράσχουν τέτοιες πληροφορίες στα εργαλεία από διάφορες πηγές, όπως τα ηλεκτρονικά φόρουμ, τα εκπαιδευτικά προγράμματα, τα βιβλία, τον σχεδιασμό και την αναφορά της τεκμηρίωσης του API.

Το (Wu et al., 2010) παρέχει το εργαλείο CoDocent για να βοηθήσει τους προγραμματιστές να εξετάσουν παραδείγματα κώδικα που διαπιστώθηκαν από τις μηχανές αναζήτησης. Για κάθε ένα παράδειγμα κώδικα, το CoDocent μπορεί αυτόματα να συνδέσει τα σχετικά έγγραφα του API για να παρέχει διαγράμματα ως αφαιρέσεις ώστε να αντικατοπτρίζει τη σημασιολογία των κλήσεων στο API. Το άρθρο εισάγει το CoDocent για να συνδέσει τα σχετικά έγγραφα στο API κατά την διάρκεια επανεξέτασης των ανακτημένων παραδειγμάτων κώδικα. Με τη διέλευση των σχετικών APIs σύμφωνα με τα APIs που αναφέρονται σε ένα παράδειγμα κώδικα, το CoDocent παρέχει δύο διαγράμματα, την κλάση cloud view και τη βοήθεια χρήσης του API, για να καθοδηγήσει τους προγραμματιστές σε διερεύνηση του παραδείγματος κώδικα.

Το (Hou και Yao, 2011) αναφέρεται στη μελέτη περίπτωσης της εξέλιξης του API που βασίζεται σε πληροφορίες που προέρχονται από την επίσημη τεκμηρίωση της Java στο API. Καταλήγει στο συμπέρασμα ότι μια σταθερή αρχιτεκτονική έχει διαδραματίσει σημαντικό ρόλο στην υποστήριξη της ομαλής εξέλιξης του API AWT / Swing. Επιπλέον, έχει δημιουργηθεί ένας κατάλογος των API λαθών σχεδιασμού και βελτιώσεις που οδηγούν στην εξέλιξη. Αυτός ο κατάλογος μπορεί να χρησιμεύσει ως διδάγματα και για τους δύο, τους σχεδιαστές του API και τους προγραμματιστές εφαρμογών.

Στο (Dekel και Herbsleb, 2009) η τεκμηρίωση των API λειτουργιών μεταφέρουν τυπικά λεπτομερείς προδιαγραφές για το όφελος των ενδιαφερόμενων αναγνωστών. Σε ορισμένες περιπτώσεις, όμως, περιέχει επίσης οδηγίες χρήσης, όπως είναι οι κανόνες ή επιφυλάξεις τις οποίες επικαλούνται οι συγγραφείς του

κώδικα που πρέπει να ενημερώνονται για αποφυγή λαθών και αδυναμιών. Υπάρχει ο κίνδυνος ότι οι οδηγίες αυτές μπορεί να "χαθούν" μέσα στο λεπτομερές κείμενο, ή ότι το κείμενο δεν θα πρέπει να διαβαστεί γιατί υπάρχουν τόσες πολλές καλούμενες λειτουργίες. Για την αντιμετώπιση αυτών των ανησυχιών για την Java, ένα Eclipse plug-in που ονομάζεται eMoose καλεί μεθόδους οι οποίες έχουν στόχους που σχετίζονται με οδηγίες. Στόχος είναι να οδηγήσει στην περαιτέρω εξέταση, η οποία ενισχύεται προβάλλοντας τις οδηγίες ετικέτας στο JavaDoc.

Δύο άρθρα (Robillard, 2009, Hou και Li, 2011) που μελετούνται σε αυτή την κατηγορία, αναλύουν τις δυσκολίες εκμάθησης των APIs. Το (Robillard, 2009) παρουσιάζει μια μελέτη των εμποδίων που αντιμετωπίζουν οι επαγγελματίες προγραμματιστές της Microsoft, όταν μαθαίνουν να χρησιμοποιούν το API. Τα APIs υποστηρίζουν επαναχρησιμοποίηση του κώδικα, την παροχή υψηλού επιπέδου αφαιρέσεων που διευκολύνουν το έργο του προγραμματισμού και βοηθούν να ενοποιηθεί η προγραμματιστική εμπειρία. Χρησιμοποιείται μια προσέγγιση εντελώς στηριζόμενη στην εμπειρία των προγραμματιστών, με την έρευνα και τις συνεντεύξεις των προγραμματιστών για τα εμπόδια που αντιμετωπίζουν μαθαίνοντας τα APIs. Τα εργαλεία λογισμικού μπορούν επίσης να βοηθήσουν τους προγραμματιστές στην προσπάθειά τους για μια καλύτερη κατανόηση των APIs. Προς το παρόν, τα εργαλεία αναζήτησης συμβάλουν στη γεφύρωση του χάσματος μεταξύ των αναγκών των χρηστών για πληροφορίες API και τους αντίστοιχους πόρους (όπως τα παραδείγματα του κώδικα).

Το (Hou και Li, 2011) αναφέρεται σε μια διερευνητική μελέτη στην οποία αναλύεται μη αυτόματα μια σειρά των ομάδων συζητήσεων σχετικά με τις συγκεκριμένες προκλήσεις που οι προγραμματιστές είχαν για ένα πλαίσιο λογισμικού. Με βάση αυτό το σύνολο των δεδομένων, εντοπίστηκαν αρκετές κατηγορίες των εμποδίων στη χρήση των APIs. Ο στόχος του άρθρου είναι να προσδιοριστούν τα σημαντικά χαρακτηριστικά των εμποδίων του API, για να μην δοκιμαστούν καθιερωμένες υποθέσεις. Περιγράφτηκαν αναλυτικά αυτά τα εμπόδια και οι πιθανές αιτίες τους. Εξετάστηκαν οι επιπτώσεις αυτών των εμποδίων για τη δημιουργία καλύτερων εργαλείων για να βοηθήσουν στην αύξηση της προσβασιμότητας στα APIs και της τεκμηρίωσης.

Μια σημαντική αναφορά γίνεται σε δυο δημοφιλή μοντέλα του λογισμικού, το μοντέλο καταρράκτη και το σπειροειδές μοντέλο. Στο άρθρο (Kernebeck, 1997) παρουσιάζονται τα βασικά στοιχεία της διαδικασίας της εκ νέου χρήσης του λογισμικού, συμπεριλαμβανομένων των βημάτων για τη δημιουργία και τη διαχείριση ενός στοιχείου μιας βιβλιοθήκης λογισμικού. Ένα μοντέλο στοιχείου παρουσιάζεται που επιτρέπει την εφαρμογή της επαναχρησιμοποίησης του πηγαίου κώδικα, τις απαιτήσεις, τον σχεδιασμό και τις πληροφορίες διασύνδεσης. Ο καταρράκτης και το σπειροειδές μοντέλο διαδικασίας του λογισμικού, χωρίς το λογισμικό της επαναχρησιμοποίησης περιλαμβάνουν τον απαιτούμενο σχεδιασμό, την εφαρμογή και τις δραστηριότητες ελέγχου. Για την επαναχρησιμοποίηση του κώδικα, αναφέρονται δύο διαφορετικές μέθοδοι, το μαύρο κουτί επαναχρησιμοποίησης και το λευκό κουτί επαναχρησιμοποίησης. Το μαύρο κουτί επαναχρησιμοποίησης δηλώνει την επαναχρησιμοποίηση του αμετάβλητου πηγαίου κώδικα. Σε αντίθεση με αυτό, το άσπρο κουτί επαναχρησιμοποίησης επιτρέπει τις αλλαγές στο τμήμα του λογισμικού που πρέπει να γίνουν.

Σε αυτό το άρθρο παρουσιάζεται η μέθοδος του εννοιολογικού χάρτη για να μελετήσει τη χρηστικότητα ενός API στη πάροδο του χρόνου. Το (Gerken et al., 2011) βασίζεται σε μια ιδέα όπου οι εννοιολογικοί χάρτες μπορούν να χρησιμοποιηθούν για να αποσπάσουν και να αξιολογήσουν τη γνώση των χρηστών έχοντας τις σύνθετες και αφηρημένες περιοχές. Αυτό επιτρέπει στον ερευνητή να αποσπάσει το νοητικό μοντέλο του προγραμματιστή κατά την εργασία με ένα API, καθιστώντας ορατή την αλληλεπίδραση και έτσι να προσδιορίσει τα θέματα ευχρηστίας και εκμάθησης των εμποδίων και την ανάπτυξή τους με τη πάροδο του χρόνου. Παρατίθενται μετρήσεις που χρησιμοποιήθηκαν στις μελέτες για να αξιολογηθεί η χρησιμότητα, υπήρξαν τόσο ποιοτικές όσο και ποσοτικές προσεγγίσεις που εισήγαγαν την έννοια του κατώτατου ορίου και του ανώτατου ορίου ως κριτήρια ποιότητας. Το κατώτατο όριο είναι το πόσο δύσκολη είναι η εκμάθηση πώς να χρησιμοποιηθεί το σύστημα και το ανώτατο όριο είναι το πως μπορεί να γίνει με τη χρήση του συστήματος.

2.3.1.4 Γενικά Ζητήματα σχετικά με τις Διασυνδέσεις Προγραμματισμού Εφαρμογών (APIs)

Στη τέταρτη κατηγορία ανήκουν 11 άρθρα γενικού περιεχομένου όσον αφορά τις Διασυνδέσεις Προγραμματισμού Εφαρμογών. Τα άρθρα (WENDYKIER και

NAGY, 2010, Koehler et al., 2008, Dezso et al., 2011, Johnsson και Mathur, 2005) αναφέρονται σε πολυδιάστατους πίνακες και επιστημονικούς υπολογισμούς, σε παράλληλα εργαλεία ανάλυσης των αποδόσεων, καθώς επίσης σε παράλληλους επεξεργαστές, στη χρήση γραφημάτων και αλγόριθμων δικτύου.

Το άρθρο (WENDYKIER και NAGY, 2010) περιγράφει το Παράλληλο Colt, μια πολυνηματική βιβλιοθήκη της Java για τους επιστημονικούς υπολογισμούς και την επεξεργασία εικόνας. Εκτός από την περιγραφή του σχεδιασμού και τη λειτουργικότητα του Παράλληλου Colt, παρουσιάζεται μια σύγκριση με το MATLAB. Δύο ImageJ συνδεόμενες μονάδες για επαναληπτική αποκατάσταση εικόνας και τη διόρθωση κίνησης του PET (Positron Emission Tomography), εικόνες του εγκεφάλου που περιγράφονται ως τυπικές εφαρμογές αυτής της βιβλιοθήκης. Περιγράφονται τα βασικά χαρακτηριστικά, η λειτουργικότητα και η απόδοση του Παράλληλου Colt, το οποίο περιλαμβάνει αποτελεσματική διαχείριση των πολυδιάστατων πινάκων (π.χ., για μια 3-διαστάσεων εικόνα), των αραιών μορφών πινάκων, πυκνό και αραιό πίνακα υπολογιστικών πυρήνων, επαναληπτικοί λύτες και προ-συνθήκες, τριγωνομετρικές μετατροπές, καθώς και μια μονάδα ελέγχου και εργαλεία μέτρησης επιδόσεων.

Το (Koehler et al., 2008) ερευνά τις προκλήσεις και παρουσιάζονται νέες τεχνικές στην αυτοματοποιημένη οργάνωση, μέτρηση του χρόνου εκτέλεσης και την απεικόνιση της συμπεριφοράς των RC (Reconfigurable Computing) εφαρμογών. Παρουσιάζονται ιδέες για την ενσωμάτωση με τα συμβατικά εργαλεία ανάλυσης της απόδοσης για τη δημιουργία ενός ενιαίου εργαλείου για RC εφαρμογές καθώς και το αρχικό πλαίσιο για FPGA οργάνωση και μέτρηση. Επίσης παρουσιάζονται έννοιες για την ενσωμάτωση αυτών των τεχνικών σε ένα υπάρχον παράλληλο εργαλείο ανάλυσης των επιδόσεων, παράλληλο οδηγό απόδοσης (PPW-Parallel Performance Wizard), με στόχο τη δημιουργία ενός ενιαίου εργαλείου για την ανάλυση της απόδοσης των RC εφαρμογών. Προκλήσεις όπως η κατανομή των πόρων, την αυτοματοποίηση της οργάνωσης και των μετρήσεων, καθώς και συμβιβασμούς μεταξύ ακριβής και σαφής μέτρησης, όλες πρέπει να αντιμετωπιστούν για την ανάλυση των επιδόσεων, για να είναι επιτυχή.

Στο άρθρο (Dezso et al., 2011) παρουσιάζεται το LEMON. Το όνομά του είναι μία συντομογραφία της Βιβλιοθήκης για Αποδοτική Μοντελοποίηση και

Βελτιστοποίηση Δικτύων (Library for Efficient Modeling and Optimization in Networks), ένα γενικό λογισμικό ανοιχτού κώδικα C++ βιβλιοθήκης που προσφέρει ευκολία στη χρήση και αποτελεσματικές υλοποιήσεις του γραφήματος και των αλγορίθμων του δικτύου και των σχετικών δομών δεδομένων. Οι βασικές έννοιες του σχεδιασμού, τα χαρακτηριστικά και οι επιδόσεις του LEMON σε σύγκριση με παρόμοια πακέτα λογισμικού, δηλαδή το BGL (Boost Graph Library) και το LEDA(Library of Efficient Data Types and Algorithms). Το LEMON κατέληξε να είναι μια βιώσιμη εναλλακτική λύση σε αυτές τις βιβλιοθήκες που χρησιμοποιούνται ευρέως.

Στο (Johnsson και Mathur, 2005) οι τεράστιοι παράλληλοι επεξεργαστές εισάγουν νέες απαιτήσεις σχετικά με τα συστήματα λογισμικού όσον αφορά την απόδοση, την επεκτασιμότητα, την ευρωστία και τη φορητότητα. Η αυξανόμενη πολυπλοκότητα των συστημάτων μνήμης και το αυξημένο εύρος του προβλήματος των μεγεθών για το οποίο ένα συγκεκριμένο κομμάτι του λογισμικού χρησιμοποιείται, θέτει σοβαρές προκλήσεις για τους προγραμματιστές λογισμικού. Η Μηχανή Σύνδεσης Επιστημονικής Βιβλιοθήκης Λογισμικού CMSSL(Connection Machine Scientific Software Library), χρησιμοποιεί διάφορες καινοτόμες τεχνικές για την αντιμετώπιση αυτών των προκλήσεων. Το CMSSL περιέχει ρουτίνες για τη διαχείριση της διανομής δεδομένων και παρέχει τα δεδομένα ανεξάρτητης κατανομής της λειτουργικότητας. Η υψηλή απόδοση επιτυγχάνεται μέσω του προγραμματισμού των αριθμητικών λειτουργιών και των δεδομένων κίνησης και με την αυτόματη επιλογή των αλγορίθμων κατά το χρόνο εκτέλεσης.

Τέσσερα άρθρα αυτής της κατηγορίας (Ferreira et al., 2007, Unalir et al., 2000, Inoue et al., 2010, Sabou, 2004) ασχολούνται με θέματα που αφορούν το κατανεμημένο περιβάλλον, τους πόρους, το HTTP πρωτόκολλο, θέματα προσομοίωσης, την αρχιτεκτονική client – server καθώς και με διαδικτυακές υπηρεσίες οι οποίες περιγράφονται από μια οντολογία.

Το (Ferreira et al., 2007) προτείνει και περιγράφει τα APIs, την αρχιτεκτονική για τη προδιαγραφή της ποιότητας των υπηρεσιών και τη διάταξη σχετικά από την πλευρά του πελάτη (με J2ME) και στην πλευρά του διακομιστή (χρησιμοποιώντας J2SE). Το άρθρο αναφέρει ζητήματα αρχιτεκτονικής και εφαρμογής. Το πρωτόκολλο διαδικτύου (IP) είναι το πρωτόκολλο επιπέδου δικτύου στις

προδιαγραφές 3GPP και η τρέχουσα τάση για την ανάπτυξη νέων τηλεπικοινωνιακών δικτύων είναι να χρησιμοποιήσει τα πρωτόκολλα του διαδικτύου. Εξετάστηκε η ποιότητα της υπηρεσίας σε 3GPP, έγινε περιγραφή της γενικής αρχιτεκτονικής του STF QoS API σε J2ME (πελάτη UE), στη συνέχεια έγινε περιγραφή της γενικής αρχιτεκτονικής του STF QoS API σε κατανεμημένους διακομιστές, και οι λειτουργικές δοκιμασίες στις οποίες διεξάγονται προσομοιώσεις. Στη συνέχεια εξετάστηκε η ενσωμάτωση του API με την υπόλοιπη STF αρχιτεκτονική. Παρουσιάστηκε μια προσομοίωση της εφαρμογής των πλαισίων PDP και μια πραγματική υλοποίηση της RSVP μέσω της UDP ενθυλάκωσης και το λειτουργικά δοκιμασμένο πρωτόκολλο RSVP.

Το άρθρο (Unalir et al., 2000) παρέχει το μηχανισμό για να διαχειριστεί σωστά τα επαναχρησιμοποιήσιμα στοιχεία και τα θέτει στη διάθεση των προγραμματιστών των λογισμικών συστημάτων. Δύο βασικές απαιτήσεις για την αγορά ενός στοιχείου άργησαν να εμφανιστούν: τα πρότυπα, τα ανταλλάξιμα μέρη και την ικανότητα των καταναλωτών να βρουν τα σωστά τμήματα για τη μη αυτόματη εργασία. Οι πρόσφατες εξελίξεις και η τεχνολογία του διαδικτύου παρέχει τα μέσα για την ικανοποίηση αυτών των απαιτήσεων. Ο στόχος του εργαλείου FedeRaL είναι να καταστήσει ολόκληρο το διαδίκτυο ως ένα χώρο αποθήκευσης της συλλογικής επαναχρησιμοποίησης για επαναχρησιμοποιήσιμα στοιχεία.

Το (Inoue et al., 2010) παρουσιάζει το WAPDB, ένα νέο σύστημα διαχείρισης βάσης δεδομένων με APIs στον παγκόσμιο Ιστό για την ταχεία ανάπτυξη των εφαρμογών Ιστού. Τα Web APIs προσφέρονται σε πολλές ιστοσελίδες για Ajax και mashup, αλλά έχουν αναπτυχθεί ανεξάρτητα εφόσον δεν υπάρχει επαναχρησιμοποιήσιμο στοιχείο βάσης δεδομένων που να ταιριάζει με εφαρμογές Ιστού. Το WAPDB έχει σχεδιαστεί με Atom, ένα σύνολο Web API πρότυπα και παρέχει πολλά χαρακτηριστικά που απαιτούνται για εφαρμογές Ιστού, συμπεριλαμβανομένου του αποτελεσματικού ελέγχου πρόσβασης, έναν εύκολο μηχανισμό επέκτασης, όπως αναζήτηση και στατιστικές δυνατότητες. Με την εισαγωγή του WAPDB, οι προγραμματιστές απελευθερώθηκαν από την ανάγκη να εφαρμόσουν αυτά τα χαρακτηριστικά γνωρίσματα καθώς και την επεξεργασία Web

API. Επιπλέον, ο σχεδιασμός ακολουθεί απόλυτα την αρχιτεκτονική REST, η οποία δίνει την ομοιομορφία και τη δυνατότητα κλιμάκωσης για εφαρμογές.

Το (Sabou, 2004) μελετά την επιτυχή απασχόληση των σημασιολογικών διαδικτυακών υπηρεσιών, εξαρτάται από τη διαθεσιμότητα της υψηλής ποιότητας οντολογιών για την περιγραφή των τομέων των υπηρεσιών αυτών. Η υπόθεση είναι ότι, εφόσον η λειτουργικότητα προσφέρεται από μια υπηρεσία Ιστού αντανακλάται από το βασικό λογισμικό, οι οντολογίες περιοχής θα μπορούσαν να κατασκευαστούν από την ανάλυση της τεκμηρίωσης του εν λόγω λογισμικού. Έχει ελεγχθεί η υπόθεση αυτή στον τομέα της RDF οντολογίας εργαλείων αποθήκευσης. Εφαρμόζεται και τελειοποιείται μια ημι-αυτόματη μέθοδος για την εξαγωγή οντολογιών τομέα από την τεκμηρίωση του λογισμικού. Η ποιότητα των εξαγόμενων οντολογιών επαληθεύτηκε κατά υψηλή ποιότητα με μη αυτόματη κατασκευή οντολογίας του ίδιου τομέα. Παρά τη χαμηλή γλωσσική ποιότητα του σώματος, η μέθοδος επιτρέπει την εξαγωγή ενός σημαντικού όγκου πληροφοριών για μια οντολογία.

Αυτό το άρθρο παρουσιάζει μια ολιστική προσέγγιση για το σχεδιασμό ενός συστήματος απεικόνισης για τα ασαφή συστήματα. Το (Pham και Brown, 2005) αναφέρει ότι τα σύνθετα ασαφή συστήματα υπάρχουν σε πολλές εφαρμογές. Αναλύονται οι απαιτήσεις για ένα τέτοιο σύστημα απεικόνισης εκφράζοντας θεμελιώδεις οντολογίες που υποστηρίζουν τη δομή και τη λειτουργία των ασαφών συστημάτων. Ένα πλαίσιο του λογισμικού χρησιμοποιεί μια πολυπρακτορική προσέγγιση με στόχο να διευκολύνει την οργάνωση και τη ροή των πολύπλοκων εργασιών, μεταξύ των σχέσεων και τις αλληλεπιδράσεις τους με τους χρήστες. Ένα πλαίσιο οπτικοποίησης βασίζεται σε πέντε κατηγορίες των πρακτόρων: πράκτορας ελέγχου, πράκτορας υπολογισμού, συμβολικός πράκτορας, πράκτορας απεικόνισης και πράκτορας προφίλ. Ο πράκτορας ελέγχου δέχεται είσοδο των χρηστών που περιλαμβάνει τις προδιαγραφές, τα ερωτήματα και τις παραμέτρους. Ο πράκτορας υπολογισμού εκτελεί όλους τους αριθμητικούς υπολογισμούς που απαιτούνται από το σύστημα λαμβάνει οδηγίες τόσο για τον πράκτορα έλεγχου και τον πράκτορα οπτικοποίησης. Ο συμβολικός πράκτορας κάνει χρήση της βάσης γνώσεων για την εκτέλεση λογικών συμπερασμάτων κανόνα. Ο πράκτορας απεικόνισης λαμβάνει οδηγίες από τον πράκτορα ελέγχου

και ζητεί πληροφορίες από τον πράκτορα υπολογισμού και τον συμβολικό πράκτορα, προκειμένου να επιλέξουν τις κατάλληλες τεχνικές απεικόνισης για να παρέχουν τις εμφανίσεις. Ο πράκτορας προφίλ καταγράφει το μοτίβο συμπεριφοράς του χρήστη όσον αφορά την επιλογή των εργασιών, τις τεχνικές απεικόνισης, τις αριθμητικές μεθόδους ή κανόνες συμπεράσματος.

Προκειμένου οι χρήστες να γράψουν αποτελεσματικά API κώδικα πελάτη έχει αναπτυχθεί μια API χρήση εξόρυξης πλαισίου που υποστηρίζει το εργαλείο που ονομάζεται MAPO (Mining APIs usages from Open source repositories - Εξόρυξη Χρήσεων του API από Ανοιχτές Πηγές Αποθήκευσης). Στο (Xie και Pei, 2006) το MAPO αξιοποιεί τις υπάρχουσες μηχανές αναζήτησης του πηγαίου κώδικα για να συλλέξει σχετικά πηγαία αρχεία και να διεξάγει εξόρυξη δεδομένων. Η εξόρυξη οδηγεί σε μια μικρή λίστα με τις συχνές χρήσεις του API για τους προγραμματιστές για να το ελέγχουν. Το MAPO αποτελείται σήμερα από πέντε συνιστώσες: μια μηχανή αναζήτησης κώδικα, έναν αναλυτή του πηγαίου κώδικα, μια ακολουθία προεπεξεργαστή, μια συχνή ακολουθία εξαγωγέα και μια συχνή ακολουθία μεταεπεξεργαστή. Η μηχανή αναζήτησης κώδικα λαμβάνει ένα ερώτημα και στη συνέχεια αναζητεί ανοιχτές πηγές αποθήκευσης για τα αρχεία προέλευσης που είναι σχετικά με το ερώτημα. Ο αναλυτής κώδικα αναλύει τα σχετικά αρχεία προέλευσης που επιστράφηκαν από τη μηχανή αναζήτησης κώδικα και παράγει μια σειρά ακολουθιών κλήσης της μεθόδου. Η ακολουθία προ-επεξεργαστή προωθεί κάποιες ακολουθίες κλήσης σε άλλους με βάση τις σχέσεις καλούντα - καλούμενου και καταργεί ορισμένες άσχετες ακολουθίες κλήσης. Η συχνή ακολουθία εξαγωγέα ανακαλύπτει συχνές ακολουθίες από τις προ-επεξεργασμένες ακολουθίες. Η συχνή ακολουθία μετά-επεξεργαστή μειώνει το σύνολο των συχνών ακολουθιών με κάποιους τρόπους.

Το τελευταίο άρθρο αυτής της κατηγορίας ερευνά έναν υπολογιστή που έχει API μηχανισμό σύνδεσης σε σύγκριση με έναν υπολογιστή χωρίς μηχανισμό προστασίας, προκείμένου να ανιχνευθεί κακόβουλο λογισμικό. Το άρθρο (Marhusin, et al., 2008) εξετάζει την ανίχνευση κακόβουλου λογισμικού, πώς με ακρίβεια και αξιοπιστία διακρίνει ένα κακόβουλο λογισμικό από ένα καλό, που εκτελείται στον ίδιο υπολογιστή. Ο χρόνος φόρτωσης συγκρίνεται με τρεις διαφορετικές ρυθμίσεις: την απλή, τον υπολογιστή με προστασία από ιούς και τον

υπολογιστή με σύνδεση API. Το σύστημα εντοπισμού κλήσης ή η διασύνδεση προγραμματισμού εφαρμογών (API), είναι μια τεχνική που εξετάζει τη χρήση των APIs, παγιδεύοντας κλήσεις που καλούν κάθε πρόγραμμα. Ο σκοπός του άρθρου είναι να αναφέρει τις επιδόσεις του υπολογιστή που έχει ένα μηχανισμό που παγιδεύει API κλήσεις που καλούν ένα πρόγραμμα.

2.3.2 Ποιοτικά χαρακτηριστικά των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών

Η ενότητα αυτή αναφέρεται στα χαρακτηριστικά ποιότητας των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών (APIs), καθώς επίσης αναφέρεται στο ερευνητικό θέμα κάθε μελέτης, την εμπειρική μέθοδο που χρησιμοποιήθηκε σε κάθε μελέτη και τέλος τις μετρικές λογισμικού που χρησιμοποιήθηκαν, όπου και αναλύονται στην επόμενη υποενότητα.

Πίνακας 4. Χαρακτηριστικά ποιότητας των APIs

Μελέτη	Ερευνητικό Θέμα	Ποιοτικά Χαρακτηριστικά (APIs)	Μετρικές Λογισμικού	Εμπειρική Μέθοδος
[S01]	Ποιότητα	χρηστικότητα		μελέτη περίπτωσης
[S02]	Εφαρμογή	ευελιξία, έλεγχος, αποδοτικότητα, επεκτασιμότητα		πείραμα
[S03]	Ποιότητα	επανασχεδίαση, εξελιξιμότητα, επεκτασιμότητα		μελέτη περίπτωσης
[S04]	Ανάλυση	ευελιξία, επαναχρησιμ/ση, χρηστικότητα		εννοιολογική ανάλυση
[S05]	Ανάλυση	κατανοησιμότητα, χρηστικότητα, αποτελεσματικ/τα		έρευνα
[S06]	Ανάλυση	ποιότητα, διαμόρφωση	σύζευξη, συνοχή, CC, MQ, MII, NC, APIU, IDI, MSHI, MSBI, SCC, LOI, MISI, TDC, CDM, CCM	πείραμα
[S07]	Ανάλυση	συντηρησιμότητα, επεκτασιμότητα, εφαρμοσιμότητα		έρευνα
[S08]	Ανάλυση	σταθερότητα, φορητότητα, ασφάλεια, επαναχρησιμ/ση	RGW, SGW	μελέτη αξιολόγησης
[S09]	Διάφορα	αποτελεσματικ/τα επαναχρησιμ/ση		μελέτη περίπτωσης

[S10]	Ανάλυση	ποιότητα, επαναχρησιμ/ση, αποτελεσματικ/τα		έρευνα
[S11]	Εφαρμογή	προσαρμοστικ/τα, ευελιξία, έλεγχος, επαναχρησιμ/ση		έρευνα
[S12]	Εφαρμογή	ευελιξία, χρηστικότητα, επεκτασιμότητα		πείραμα
[S13]	Διάφορα	ευελιξία, αποτελεσματικ/τα		εννοιολογική ανάλυση
[S14]	Διάφορα	φορητότητα, κατανοησιμότητα	HMM	μελέτη περίπτωσης
[S15]	Διάφορα	ευρωστία, συνδεσιμότητα, ποιότητα		έρευνα
[S16]	Ανάλυση	αποτελεσματικ/τα χρηστικότητα	MAP, SCORE	μελέτη αξιολόγησης
[S17]	Ανάλυση	χρηστικότητα, εφαρμοσιμότητα, κατανοησιμότητα		μελέτη αξιολόγησης
[S18]	Εφαρμογή	δοκιμή, επαλήθευση, ενσωμάτωση		πείραμα
[S19]	Εφαρμογή	συνέπεια, απόδοση, επαναχρησιμ/ση		έρευνα
[S20]	Ανάλυση	χρηστικότητα, ποιότητα		εννοιολογική ανάλυση
[S21]	Ανάλυση	χρηστικότητα, λειτουργικότητα		μελέτη αξιολόγησης
[S22]	Ποιότητα	δυνατότητα εκμάθησης, κατανοησιμότητα		έρευνα
[S23]	Ποιότητα	πρακτικότητα, πληρότητα, φορητότητα, χρηστικότητα		βιβλιογραφική ανασκόπηση
[S24]	Ανάλυση	ποιότητα, επαναχρησιμ/ση		έρευνα
[S25]	Ανάλυση	επαναχρησιμ/ση, λειτουργικότητα, παραγωγικότητα	OOFP, C&K, FP, WMC, DIT, NOC, CBO, RFC, LCOM	μελέτη περίπτωσης
[S26]	Ανάλυση	κατανοησιμότητα, αποτελεσματικ/τα		εννοιολογική ανάλυση
[S27]	Εφαρμογή	ευελιξία, επεκτασιμότητα, χρηστικότητα		πείραμα
[S28]	Ανάλυση	ποιότητα, φορητότητα, ενσωμάτωση		πείραμα
[S29]	Διάφορα	ευελιξία, αποδοτικότητα		μελέτη αξιολόγησης
[S30]	Ανάλυση	συντηρησιμότητα, επαναχρησιμ/ση		έρευνα

[S31]	Διάφορα	προσαρμοστικ/τα, επεκτασιμότητα, αποδοτικότητα		μελέτη περίπτωσης
[S32]	Διάφορα	διαλειτουργικό/τα, επεκτασιμότητα, επαναχρησιμ/ση		έρευνα
[S33]	Διάφορα	ευελιξία, λειτουργικότητα		πείραμα
[S34]	Διάφορα	διαλειτουργικό/τα, διαδραστικο/τα, επαναχρησιμ/ση, επεκτασιμότητα		έρευνα
[S35]	Διάφορα	φορητότητα, ευρωστία, λειτουργικότητα		έρευνα
[S36]	Ποιότητα	έλεγχος, επαναχρησιμ/ση		έρευνα
[S37]	Εφαρμογή	αποτελεσματικ/τα ποιότητα		πείραμα
[S38]	Διάφορα	ευελιξία, αποδοτικότητα		μελέτη αξιολόγησης
[S39]	Ανάλυση	χρηστικότητα, λειτουργικότητα		μελέτη περίπτωσης
[S40]	Ανάλυση	χρηστικότητα		μελέτη περίπτωσης
[S41]	Ανάλυση	σταθερότητα, έλεγχος		πείραμα
[S42]	Ανάλυση	ορατότητα, συνέπεια, πληρότητα	OD, OR, AD, AR, CC, ER	έρευνα
[S43]	Ανάλυση	επαναχρησιμ/ση		μελέτη περίπτωσης
[S44]	Ανάλυση	αποδοτικότητα, χρηστικότητα		πείραμα
[S45]	Ανάλυση	επαναχρησιμ/ση, αποτελεσματικ/τα		μελέτη αξιολόγησης
[S46]	Ανάλυση	σαφήνεια, συνέπεια, συνεκτικότητα		μελέτη περίπτωσης
[S47]	Ποιότητα	χρηστικότητα, κατανοησιμότητα, αποτελεσματικ/τα		μελέτη αξιολόγησης

2.3.2.1 Μετρικές Λογισμικού

Σε αυτή την ενότητα αναλύονται οι μετρικές που μετρούν την ποιότητα του λογισμικού που αναφέραμε στην ενότητα 2.3.2. Οι μετρικές μπορούν να χωριστούν σε πολλές κατηγορίες. Στην πρώτη κατηγορία μελετάμε τις δομικές μετρικές με βάση τη σύζευξη, που παρέχουν διάφορα μέτρα της κλήσης της συνάρτησης μέσω των API μονάδων σε σχέση με την συνολική κλήση της συνάρτησης. Τέτοιες μετρικές είναι (1) η μετρική σύζευξης - συνοχής, μετρά τα πρότυπα σύζεύξης - συνοχής μεταξύ των μονάδων βάσει των δομικών

εξαρτήσεων, (2) η μετρική με βάση μη δομημένες πληροφορίες, μετρά τη συνεκτικότητα και τη σύζευξη των οντοτήτων με βάση τις αμοιβαίες πληροφορίες κατά την έννοια της θεωρίας πληροφοριών, (3) η MII (Module Interaction Index), υπολογίζει πόσο αποτελεσματικά οι API λειτουργίες μιας μονάδας χρησιμοποιούνται από τις άλλες μονάδες του συστήματος, (4) η NC (Non - API Function Closedness Index), μετρά την έκταση της κίνησης της λειτουργίας κλήσεων των μη API λειτουργιών της μονάδας που δεν πρέπει να εκθέτουν τον εαυτό τους με τον έξω κόσμο. (5) η APIU (API Function Usage Index), προσδιορίζει ποιο τμήμα των λειτουργιών API εκτέθηκε από μια μονάδα που χρησιμοποιείται από τις άλλες μονάδες. (6) η IDI (Implicit Dependency Index), δημιουργείται από την καταγραφή, για κάθε μονάδα του αριθμού των λειτουργιών που γράφουν σε καθολικές οντότητες (όπως μεταβλητές, αρχεία, βάσεις δεδομένων), με την προϋπόθεση ότι τέτοιες καθολικές οντότητες είναι προσβάσιμες από λειτουργίες σε άλλες μονάδες.

Στη δεύτερη κατηγορία μελετάμε τις μετρικές σε επίπεδο κώδικα, βασικό πλεονέκτημά τους είναι ότι παρέχουν στους προγραμματιστές διορατικότητα στο εσωτερικό του κώδικα που αναπτύσσουν και τους βοηθούν να κατανοήσουν ποια κομμάτια κώδικα πρέπει να ανακατασκευαστούν ή να ελεγχθούν σχολαστικά. Βοηθούν στην αναγνώριση ενδεχόμενων κινδύνων, στην κατανόηση της εκάστοτε κατάστασης ενός έργου και στην καταγραφή της προόδου κατά την ανάπτυξη του λογισμικού. Τέτοιες είναι (1) η MSUI (Module Size Uniformity Index), κατευθύνει τον αλγόριθμο διαμόρφωσης προς την παραγωγή μονάδων των οποίων τα μεγέθη είναι σε αποδεκτά όρια, τοποθετώντας όλο το κώδικα σε μια ενιαία μονάδα είναι τεχνικά μια σωστή ρύθμιση, αλλά προφανώς δεν είναι αποδεκτή. Ο περιορισμός όσον αφορά την ομοιομορφία του μεγέθους μπορεί να εκφραστεί με όρους του μέσου όρου και της τυπικής απόκλισης που σχετίζονται με τα μεγέθη της μονάδας. (2) η MSBI (Module Size Boundedness Index), μετρά το βαθμό στον οποίο τα μεγέθη των μονάδων διαφέρουν από κάποια επιθυμητή τιμή, (3) η WMPC (Weighted Methods Per Class), που υπολογίζει την πολυπλοκότητα μιας κλάσης, μετρώντας τον αριθμό των μεθόδων που την αποτελούν, (4) η DIT (Depth of Inheritance Tree), που συσχετίζει το βάθος μιας κλάσης στην ιεραρχία με την πολυπλοκότητά της, θεωρώντας ότι όσο μεγαλύτερο το βάθος στην ιεραρχία, τόσο πιο πιθανή η κληρονομικότητα κλάσεων υψηλότερων στην ιεραρχία, που έχει ως

αποτέλεσμα την αύξηση του αριθμού των μεθόδων στην κλάση και συνεπώς την αύξηση της πολυπλοκότητας. (5) η NOC (Number Of Children) που μετράει πόσες υποκλάσεις πρόκειται να κληρονομήσουν τις μεθόδους των κλάσεων γονέων, (6) η CBO (Coupling Between Object Classes) που υπολογίζει το κατά πόσο οι κλάσεις εξαρτώνται από άλλες κλάσεις για να λειτουργήσουν ή είναι αυτόνομες, γεγονός που επηρεάζει πολύ την δυνατότητα επαναχρησιμοποίησης, (7) η RFC (Response For a Class), που μετράει τον αριθμό των μεθόδων που ανταποκρίνονται κατά τη λήψη ενός μηνύματος από μια κλάση. Όσο μεγαλύτερος ο αριθμός αυτός, τόσο πιο σύνθετη αναμένεται να είναι η διαδικασία της αποσφαλμάτωσης και του ελέγχου, αφού για τη λειτουργία της κλάσης εμπλέκονται και μέθοδοι που καλούνται από εξωτερικές κλάσεις, αυξάνοντας έτσι την πολυπλοκότητα. (8) η LCOM (Lack of Cohesion in Methods) που μετράει τη συνοχή των μεθόδων μιας κλάσης, και είναι επιθυμητή αφού προωθεί την ενθυλάκωση και μειώνει τη πολυπλοκότητα και την πιθανότητα λαθών κατά τη διαδικασία της ανάπτυξης.

Στην επόμενη κατηγορία ανήκουν μετρικές με βάση την αρχιτεκτονική τους. Τέτοιες είναι (1) η CDI (Cyclic Dependency Index), μετρά την έκταση των κυκλικών εξαρτήσεων μεταξύ των μονάδων του συστήματος. Η μετρική καλύπτει προφανώς την αρχή της ελαχιστοποίησης των κυκλικών εξαρτήσεων μεταξύ των μονάδων. (2) η LOI (Layer Organization Index), παρέχει μια σειρά υπηρεσιών που μπορούν να χρησιμοποιηθούν από στρώματα πάνω από αυτό. Ένα στρώμα είναι μόνο γνώση (οι κλήσεις συνάρτησης μόνο για τα κατώτερα στρώματα) του στρώματος κάτω από αυτό και δεν είναι ενήμερη για το στρώμα πάνω από αυτό. (3) η MISI (Module Interaction Stability Index), μετρά την σταθερότητα των μονάδων στο χαμηλότερο στρώμα σε σχέση με τις μονάδες στα ανώτερα στρώματα. (4) η TDC (Testability Dependency Count), μετρά το βαθμό στον οποίο οι μονάδες εξαρτώνται από άλλες όσον αφορά την ελεγξιμότητα. Αυτή η μετρική χαρακτηρίζει το λογισμικό σύμφωνα με την αρχή της μεγιστοποίησης της δυνατότητας δοκιμών των μονάδων.

Στην τέταρτη κατηγορία ανήκουν μετρικές με βάση την ομοιότητα του σκοπού. Τέτοιες είναι (1) η CDM (Concept Domination Metric), μετρά τη μη ομοιομορφία της κατανομής πιθανοτήτων των εννοιών. Όλες οι έννοιες θα πρέπει να κατανέμονται ομοιόμορφα και η τιμή της μετρικής αναμένεται να πάει στο

μηδέν. (2) η CCM (Concept Coherency Metric), βασίζεται στην ιδέα της θεωρίας των πληροφοριών ότι αν μια έννοια είναι κεντρικής σημασίας για τις υπηρεσίες που προσφέρονται από μια μονάδα, τότε η κοινή πληροφορία μεταξύ της μονάδας και της έννοιας θα πρέπει να είναι υψηλή.

Στην πέμπτη κατηγορία ανήκουν μετρικές, για το χαρακτηρισμό της αναφοράς και την αναπαράσταση των εννοιών στα APIs. Τέτοιες μετρικές είναι (1) η ER (Reference Explicitness Ratio), υπολογίζει την αναλογία των στοιχείων του προγράμματος που αφορούν μόνο έννοιες. Η ER παίρνει τιμές μεταξύ μηδέν και ένα - όσο υψηλότερο είναι το ER (πιο κοντά στο ένα), τόσο πιο σαφές είναι το API στο σύνολό του και ως εκ τούτου οι έννοιες έχουν καλή ορατότητα. (2) η CC (Conceptual Complexity), πολυπλοκότητα ενός ολόκληρου API είναι ο μέσος όρος του αριθμού των εννοιών που αναφέρονται από τα στοιχεία του προγράμματος. (3) η OD (Overloading Degree), ποσοτικοποιεί τον αριθμό των εννοιών που αναπαρίστανται μέσω του ίδιου τύπου, (4) η OR (Overloading Ratio), μετρά τον μέσο όρο του βαθμού υπερφόρτωσης για ένα ολόκληρο API. Η OR μπορεί να πάρει τιμές μόνο μεγαλύτερες ή ίσες με ένα. (5) η AD (Ambiguity Degree), ορίζει τον αριθμό των διακριτών παραστάσεων που χρησιμοποιούνται για μια ενιαία έννοια, (6) η AR (Ambiguity Ratio), για ένα API είναι ο μέσος όρος των βαθμών ασάφειας όλων των εννοιών που αναπαρίστανται στο API.

Οι μετρικές που αναφέρονται σε αυτή τη παράγραφο δεν ομαδοποιούνται ως προς μια ειδική κατηγορία, όπως έγινε με τις πέντε προηγούμενες κατηγορίες, αλλά γενικά μελετούνται μετρικές των APIs. Οι μετρικές που μελετάμε είναι (1) η RGW (Mean Serving Rate) του OSA / Parlay της πύλης ελέγχου κλήσεων, ορίζεται ως ο λόγος των αιτήσεων που εξυπηρετούνται από τη πύλη ως προς τα αιτήματα που έφθασαν προς την πύλη, σε μια συγκεκριμένη χρονική περίοδο. (2) η SGW (Mean Throughput), η κύρια ρυθμοαπόδοση της πύλης, η οποία ορίζεται ως ο λόγος των επιτυχών αιτήσεων υπηρεσιών μέσω της πύλης ελέγχου κλήσεων για να φτάσει τις συνολικές αιτήσεις σε αυτό, στη μονάδα του χρόνου. (3) η HMM (Hardware Measurement Module), είναι υπεύθυνη για την εφαρμογή όλων των προφίλ, ανίχνευσης, και τις δυνατότητες δειγματοληψίας, καθώς και τη τοποθέτηση των δεδομένων για την ανάκτηση από το λογισμικό. Η HMM επιτρέπει τη γρήγορη προσαρμογή, (και εύκολη πρόσβαση) σε όλους αυτούς τους πόρους,

εξαλείφοντας τη χρονοβόρα και επιρρεπή σε λάθη διαδικασία της μη αυτόματης μέτρησης των επιδόσεων. (4) η MAP (Mean Average Precision), εξάγει από το iBUGS έγγραφο τα ονόματα των αρχείων που σχετίζονται με ένα σφάλμα. Για τον υπολογισμό της MAP, αναφερόμαστε σε αυτό το σύνολο ως ένα σχετιζόμενο σύνολο. (5) η SCORE, μπορεί να χρησιμοποιηθεί για να δηλώσει το ποσοστό του προγράμματος που πρέπει να εξεταστεί προκειμένου να εντοπίσει μια εσφαλμένη δήλωση. (6) η OOFP (Object-Oriented Function Point), μετρά το μέγεθος της εφαρμογής λαμβάνοντας υπόψη τη λειτουργικότητα, για να καταστεί δυνατή η μέτρηση της αντικειμενοστρεφούς ανάλυσης και οι προδιαγραφές σχεδιασμού. Η OOFP θα μπορούσε να χρησιμοποιηθεί για την έμμεση αξιολόγηση της προσπάθειας ανάπτυξης εφαρμογών, χρησιμοποιήθηκε η OOFP για τη μέτρηση της παραγωγικότητας. (7) οι C και K μετρικές (Chidamber και Kemerer), είναι μια από τις πιο διάσημες μετρικές για την αξιολόγηση της πολυπλοκότητας του αντικειμενοστρεφούς λογισμικού. Οι C και K μετρικές έδειξαν ότι είναι καλύτερης πρόβλεψης της αποτυχίας της κλάσης από τις παραδοσιακές μετρικές κώδικα. Οι C και K μετρικές χρησιμοποιούνται για την έμμεση μέτρηση της ποιότητας των εφαρμογών.

2.3.3 Η χρήση επανασχεδιάσεων σε επίπεδο API για την επίλυση σφαλμάτων

Στην υποενότητα αυτή συζητάμε για τον ρόλο των επανασχεδιάσεων στην επίλυση σφαλμάτων (Kim et al., 2011). Ευρέως πιστεύεται ότι η επανασχεδίαση βελτιώνει την ποιότητα του λογισμικού και την παραγωγικότητα του προγραμματιστή, καθιστώντας ευκολότερο να διατηρήσει και να κατανοήσει τα συστήματα λογισμικού. Η επανασχεδίαση είναι η διαδικασία αλλαγής της δομής του σχεδιασμού ενός προγράμματος, χωρίς τη μεταβολή της εξωτερικής λειτουργικής συμπεριφοράς του προκειμένου να βελτιωθεί η αναγνωσιμότητα του προγράμματος, η συντήρηση και η επεκτασιμότητα. Μελετήσαμε ποσοτικά και ποιοτικά τις επανασχεδιάσεις σε επίπεδο API και τις διορθώσεις σφαλμάτων σε τρία μεγάλα έργα ανοικτού κώδικα, στο Eclipse JDT, στο jEdit και στο Columba.

Βρήκαμε ότι το 29,1% στο 44,4% των αναθεωρήσεων επανασχεδίασης περιλαμβάνουν επίσης διορθώσεις σφάλματος στην ίδια αναθεώρηση. Επιπλέον, το 32% των αναθεωρήσεων επανασχεδίασης σχετίζεται με τον καθορισμό αναθεωρήσεων που ακολουθούν μέσα σε 20 αναθεωρήσεις, σε αντίθεση με το

14% των αναθεωρήσεων μη επανασχεδίασης που σχετίζονται με αργότερες διορθώσεις σφαλμάτων μέσα σε 20 αναθεωρήσεις. Αυτό σημαίνει ότι, σε πολλές περιπτώσεις, είτε οι επανασχεδιάσεις δημιούργησαν νέα σφάλματα ή οι επανασχεδιάσεις μπορεί να έχουν εφαρμοστεί για τη διευκόλυνση διορθώσεων που ήταν δύσκολο να εφαρμοστούν χωρίς αυτές. Παρατηρήσαμε ότι κατά την επίλυση σφαλμάτων είχαμε μείωση του χρόνου μετά τις επανασχεδιάσεις. Οι προγραμματιστές ανακαλύπτουν σφάλματα κατά τη διάρκεια των επανασχεδιάσεων προσπαθώντας για γρήγορη επίλυση αυτών, κάνουν αλλαγές στον κώδικα ή καθορίζουν γρήγορα ελλιπείς επανασχεδιάσεις. Οι επανασχεδιάσεις σε συνδυασμό με τις διορθώσεις σφαλμάτων επιβαρύνονται συχνά με συμπληρωματικές διορθώσεις, οι οποίες είναι συνήθως μικρότερες και ευκολότερο να εφαρμοστούν από τις κύριες διορθώσεις σφαλμάτων. Όταν πρόκειται για επίλυση σφαλμάτων που εισάγονται κοντά στο χρόνο των επανασχεδιάσεων ο μέσος χρόνος επίλυσης τείνει να μειώνεται μετά τις επανασχεδιάσεις.

Για να εξετάσουμε πόσες επανασχεδιάσεις έγιναν ως μέρος μιας διόρθωσης σφάλματος, πρώτα μετρήσαμε την έκταση των αναθεωρήσεων που περιλαμβάνουν τόσο τις επανασχεδιάσεις όσο και τις διορθώσεις σφαλμάτων. Στον Πίνακα 5, η P(F) είναι η πιθανότητα της αναθεώρησης για να συμπεριλάβει μια διόρθωση σφάλματος, η P(R) είναι η πιθανότητα της αναθεώρησης για να συμπεριλάβει μια επανασχεδίαση σε επίπεδο API, η P(R | F) είναι μια δεσμευμένη πιθανότητα των συμπεριλαμβανομένων μιας επανασχεδίασης δεδομένης της αναθεώρησης, αντιμετώπισης του συγκεκριμένου σφάλματος. Η P(R|¬ F) είναι μια δεσμευμένη πιθανότητα συμπεριλαμβανομένης μιας επανασχεδίασης δεδομένου ότι δεν είναι μια αναθεώρηση επίλυσης. Στο Eclipse JDT, βρήκαμε ότι περισσότερο από το 41,5% αναθεωρήσεων επανασχεδίασης συνδέθηκαν με διορθώσεις σφαλμάτων (29,1% στο jEdit και 44,4% στο Columba). Επιπλέον, η πιθανότητα να συμπεριληφθεί μια επανασχεδίαση δεδομένης μιας επίλυσης επανασχεδίασης είναι πολύ υψηλότερη από ό,τι η πιθανότητα να συμπεριληφθεί η επανασχεδίαση δεδομένης μιας μη επίλυσης της αναθεώρησης (12,0% έναντι 5,7% στο Eclipse, 11,5% έναντι 3,0% στο jEdit, και 10,7% έναντι 7,4% στο Columba). Η πιθανότητα να συμπεριληφθεί μια λύση δεδομένης μιας αναθεώρησης της επανασχεδίασης είναι 62,8%, ενώ η πιθανότητα να

συμπεριληφθεί μια λύση δίνεται με μη αναθεώρηση της επανασχεδίασης, 51,7% στο Eclipse. Είναι ενδιαφέρον, ότι το jEdit και το Columba δείχνουν τις αντίθετες τάσεις: 33,0% έναντι 46,9% στο jEdit, και 24,3% έναντι 36.3% στο Columba.

Τα αποτελέσματα σε επίπεδο μεθόδου είναι ελαφρώς διαφορετικά από την αναθεώρηση σε επίπεδο ανάλυσης, επειδή η επανασχεδίαση τείνει να περιλαμβάνει οριζόντιες αλλαγές σε περισσότερες από μία μεθόδους, καθιστώντας την πιθανότητα μιας μεθόδου να περιλαμβάνει μια επανασχεδίαση πολύ μεγαλύτερη από την πιθανότητα μιας αναθεώρησης που περιλαμβάνει μια επανασχεδίαση. Για να εξεταστεί κατά πόσο οι διορθώσεις σφάλματος επιλύθηκαν μετά τις επανασχεδιάσεις, μετρήσαμε το ποσοστό των αναθεωρήσεων επανασχεδίασης που έχουν τουλάχιστον μια επίλυση σφάλματος που εφαρμόζονται στην ίδια θέση επανασχεδίασης σε επίπεδο μεθόδου μέσα σε 20 αναθεωρήσεις. Ως ομάδα ελέγχου, μετρήθηκε το ποσοστό των αναθεωρήσεων μη επανασχεδιάσεων που έχει τουλάχιστον μια επίλυση σφάλματος που εφαρμόζεται στην ίδια θέση αλλαγής μέσα σε 20 αναθεωρήσεις από την αλλαγή. Το αποτέλεσμα αυτό σημαίνει ότι είναι πιο πιθανό οι αναθεωρήσεις επανασχεδίασης να ακολουθούνται από σχετικές διορθώσεις σφάλματος, από τις αναθεωρήσεις μη επανασχεδίασης που πρέπει να ακολουθούνται από σχετικές διορθώσεις.

Πίνακας 5. Πιθανότητες επίλυσης σφαλμάτων και επανασχεδιάσεις σε επίπεδο αναθεώρησης και σε επίπεδο μεθόδου

επίπεδο αναθεώρησης									
Project	# αναθεωρήσεις	|F|	|R|	P(F)	P(R)	P(R|F)	P(R|¬F)	P(F|R)	P(F|¬R)
Eclipse JDT	15,000	3752	1089	0,25	0,073	0,120	0,057	0,415	0,237
JEdit	11,102	1073	423	0,097	0,038	0,115	0,030	0,291	0,089
Columba	421	150	36	0,357	0,086	0,107	0,074	0,444	0,348
Total	26,523	4975	1548	0,188	0,058	0,119	0,044	0,382	0,176
επίπεδο μεθόδου									
Project	# αναθεωρήσεις	|F|	|R|	P(F)	P(R)	P(R|F)	P(R|¬F)	P(F|R)	P(F|¬R)
Eclipse JDT	21,938	12049	6278	0,549	0,286	0,327	0,236	0,628	0,517
JEdit	8,938	3704	3529	0,414	0,395	0,315	0,451	0,330	0,469
Columba	2,191	745	424	0,34	0,194	0,138	0,222	0,243	0,363
Total	33,067	16498	10231	0,499	0,309	0,316	0,303	0,510	0,492

2.3.4 Τα εμπόδια που συντελούν στη δυσκολία εκμάθησης του API

Σε αυτή τη ενότητα μελετάμε τα εμπόδια που δυσκολεύουν την εκμάθηση του API. Τα εμπόδια χωρίζονται σε πέντε κατηγορίες. Οι κύριες κατηγορίες των εμποδίων εκμάθησης του API είναι οι πόροι, η δομή, το υπόβαθρο, το τεχνικό περιβάλλον και τα θέματα διαδικασίας καθεμία από αυτές αποτελείται από υποκατηγορίες. Η κατηγορία των πόρων αναφέρεται σε εμπόδια που προκαλούνται από ανεπαρκείς ή απόντες πόρους για την εκμάθηση του API (για παράδειγμα, η τεκμηρίωση) ακόμη αποτελείται από τις εξής υποκατηγορίες: τα παραδείγματα, το περιεχόμενο, την εργασία, τη μορφή και τη σχεδίαση. Τα παραδείγματα αναφέρονται σε ανεπαρκή παραδείγματα, το περιεχόμενο αναφέρεται σε ένα ειδικό κομμάτι του περιεχομένου που λείπει ή δεν μπορεί να παρουσιαστεί στην τεκμηρίωση (για παράδειγμα, πληροφορίες για όλες τις εξαιρέσεις που ρίχνονται), η εργασία αναφέρει ότι δεν υπάρχει αναφορά πώς να χρησιμοποιηθεί το API για να ολοκληρωθεί μια συγκεκριμένη εργασία, η μορφή αναφέρει ότι οι πόροι δεν είναι διαθέσιμοι στην επιθυμητή μορφή και η σχεδίαση την ανεπαρκή τεκμηρίωση στο υψηλό επίπεδο του API όπως η σχεδίαση ή η λογική. Η κατηγορία της δομής αναφέρεται σε εμπόδια που σχετίζονται με την δομή ή την σχεδίαση του API, αποτελείται από τις εξής υποκατηγορίες: την σχεδίαση που αναφέρεται σε θέματα δομικής σχεδίασης του API, τον έλεγχο και την αποσφαλμάτωση που αναφέρονται σε θέματα που σχετίζονται με τον έλεγχο και την αποσφαλμάτωση του API και τη συμπεριφορά στο χρόνο εκτέλεσης. Η κατηγορία του υπόβαθρου σχετίζεται με εμπόδια που προκαλούνται από το υπόβαθρο των συμμετεχόντων και την προηγούμενη εμπειρία τους. Η κατηγορία του τεχνικού περιβάλλοντος αναφέρεται σε εμπόδια που προκαλούνται από το τεχνικό περιβάλλον στο οποίο χρησιμοποιήθηκε το API (για παράδειγμα, ετερογενές σύστημα, υλικό). Η κατηγορία της διαδικασίας σχετίζεται με εμπόδια που αναφέρονται σε θέματα διαδικασιών (για παράδειγμα, χρόνος, διακοπές).

2.3.5 Παράγοντες που επηρεάζουν τη χρηστικότητα του API

Στην ενότητα αυτή παρουσιάζουμε τους 22 παράγοντες που επηρεάζουν την χρηστικότητα του API σύμφωνα με το (Zibran et al., 2011). Ο πίνακας περιέχει αριθμημένους τους παράγοντες από το 1-22, καθώς επίσης το όνομα κάθε παράγοντα και μια σύντομη περιγραφή.

Πίνακας 6. Παράγοντες που επηρεάζουν τη χρηστικότητα του API

Δείκτης	Παράγοντας Χρηστικότητας	Περιγραφή
F - 01	Πολυπλοκότητα	Αύξηση του μεγέθους και της πολυπλοκότητας των εξαγόμενων χαρακτηριστικών, της έννοιας και τη μείωση της χρηστικότητας.
F - 02	Ονομασία	Σύμβαση που ακολούθησε την ονομασία των συναρτήσεων σε επίπεδο διασύνδεσης και τις μεταβλητές. Τα περιγραφικά ονόματα είναι προτιμότερα για τις συντομεύσεις των ονομάτων.
F - 03	Διατήρηση του καλούντος	Σαφήνεια πώς ο καλών θα κινήσει τις συναρτήσεις ή τα χαρακτηριστικά που πρέπει να είναι σαφή / διαισθητικά προς τον χρήστη για καλύτερη χρηστικότητα.
F - 04	Τεκμηρίωση	Πλήρη, σαφή και μέχρι την ημερομηνία τεκμηρίωση και παραδείγματα χρήσης αύξησης της χρηστικότητας
F - 05	Συνέπεια	Συνέπεια στο σχεδιασμό και την τήρηση με κοινές συμβάσεις αύξησης της χρηστικότητας.
F - 06	Εννοιολογική ορθότητα	Εννοιολογική ορθότητα στο σχεδιασμό και την ονομασία των χαρακτηριστικών είναι σημαντικό για τη χρηστικότητα.
F - 07	Παράμετρος και επιστροφή	Ο αριθμός και ο τύπος των παραμέτρων για τις συναρτήσεις και οι τύποι επιστροφής έχουν σημαντικό αντίκτυπο στη χρηστικότητα. Πάρα πολλές παράμετροι μειώνουν τη χρηστικότητα.
F - 08	Δομητής παραμέτρου	Η προεπιλογή (parameterless) του δομητή είναι συχνά ευκολότερη από τον παραμετροποιημένο δομητή για να αρχικοποιεί αντικείμενα, ειδικά για τους αρχάριους και τους προχωρημένους προγραμματιστές.
F - 09	Πρότυπο Εργοστασίου έναντι δομητή	Οι προγραμματιστές φυσικά αναμένουν τον κατασκευαστή να αρχικοποιήσει το αντικείμενο, παρά τις μεθόδους του εργοστάσιου. Αρχικοποιώντας τα αντικείμενα μέσω των μεθόδων του εργοστάσιου μπορεί να προκαλέσει μερικές φορές δυσκολία.
F - 10	Τύποι δεδομένων	Οι τύποι των εξαγόμενων αντικειμένων και χαρακτηριστικών. Οι τύποι δεδομένων πρέπει να επιλέγονται σωστά για να αποφευχθεί το άσκοπο type-casting, η κατανάλωση των πόρων και η απώλεια της ακρίβειας.
F - 11	Χρήση των χαρακτηριστικών	Διασπορά και λειτουργικές εξαρτήσεις των χαρακτηριστικών. Συνεκτική εφαρμογή της λειτουργικότητας που αυξάνει τη χρηστικότητα.
F - 12	Συγχρονισμός	Η ορθή εφαρμογή του συγχρονισμού και της εξαγωγής ευμετάβλητων στοιχείων. Η περιττή εξαγωγή μεταβλητών στοιχείων μπορεί να αυξήσουν τα θέματα ασφάλειας νήματος και να αυξήσουν τις παγίδες για την κακή χρήση.
F - 13	Χειρισμός λάθους	Μηχανισμός για την πρόληψη σφάλματος από την απόκρυψη πληροφοριών, καθώς και τον κατάλληλο χειρισμό των συνθηκών σφάλματος με πληροφορίες διάγνωσης και μηχανισμό για την ανάκαμψη.

F - 14	Συνδρομητικότητα για τον πελάτη	Διαθεσιμότητα της έτοιμης εφαρμογής των όσων οι χρήστες μπορεί να χρειαστούν, για να μειώσουν την επιβάρυνση των χρηστών.
F - 15	Πολλαπλοί τρόποι να κάνει ένα πράγμα	Διαθεσιμότητα πολλών τρόπων (π.χ., διάφορες μεθόδους που προσφέρουν την ίδια λειτουργικότητα) να κάνουν το ίδιο πράγμα μπορεί να μπερδέψει τους χρήστες να επιλέξουν από τις εναλλακτικές λύσεις.
F - 16	Αλυσίδα αναφοράς	Μεγάλη αλυσίδα των κλήσεων μεθόδου ή της ιεραρχίας κληρονομικότητας είναι δύσκολο να εντοπιστούν και να μειώσουν τη χρηστικότητα.
F - 17	Εφαρμογή έναντι διεπαφή εξάρτησης	Διασύνδεση εξαρτήσεων μεταξύ των στοιχείων που παρέχουν μεγαλύτερη ευελιξία και έτσι αυτά είναι που προτείνονται για την εφαρμογή των εξαρτήσεων.
F - 18	Διαχείριση μνήμης	Διαχείριση μνήμης (κατανομή και deallocation της μνήμης) ευθύνες που επαφίονται στο χρήστη να μειώσει τη χρηστικότητα του API.
F - 19	Τεχνική αναντιστοιχία	Συμβατότητα με την πλατφόρμα και άλλες τεχνολογίες στο λειτουργικό περιβάλλον είναι σημαντική για τη χρηστικότητα.
F - 20	Αλλαγή API	Συμβατότητα με προηγούμενα είναι απαραίτητη για τη χρηστικότητα, ενώ η υποτίμηση των κοινών χαρακτηριστικών μπορεί να εκπλήξει τους χρήστες.
F - 21	API aging	API aging όταν συμβαίνουν αλλαγές στην πλατφόρμα στόχος, αλλά το API δεν συμβαδίζει με την εξέλιξη της πλατφόρμας και κατά συνέπεια, γίνεται ακατάλληλο προς χρήση το API.
F - 22	Αναγνωρισιμότητα κώδικα	Η αναγνωσιμότητα κώδικα του πελάτη επηρεάζει την συντηρησιμότητα.

Ακολουθήσαμε τη μελέτη (Zibran et al., 2011) και καθορίσαμε την προτεινόμενη σειρά των API παραγόντων χρηστικότητας προς την πληρότητα και την ορθότητα. Δεν υπάρχει ποσοτική ένδειξη για το πόσο σημαντικός είναι ο παράγοντας χρηστικότητας (π.χ. τεκμηρίωση) είναι πάνω από ένα άλλο (π.χ., εξάρτηση), μέχρι αυτή τη μελέτη, όπου καταλάβαμε αυτούς τους παράγοντες, σύμφωνα με μια ρεαλιστική σημασία τους με βάση το πόσο συχνά οι παράγοντες αντανακλώνται στα μηνύματα λάθους. Γενικά, ο τεράστιος αριθμός των σφαλμάτων, καθιστά δύσκολο για τους προγραμματιστές API να ανταποκριθούν γρήγορα σε αυτά. Είναι ενδιαφέρον, ότι το 37,14% των εκθέσεων σφαλμάτων στη μελέτη μας βρέθηκε να σχετίζεται με τη χρηστικότητα API, μεταξύ των οποίων ένα σημαντικό μέρος ασχολείται με τα πιο συχνά θέματα ευχρηστίας. Ως εκ τούτου, η σωστή μελέτη αυτών των θεμάτων, μπορεί να μειώσει σημαντικά τον αριθμό των καθημερινών σφαλμάτων, η οποία κατά συνέπεια, θα καταστήσει συνεπώς την εργασία συντήρησης του API ευκολότερη.

2.4 ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην ενότητα αυτή συζητάμε τα συμπεράσματα που προκύπτουν από την ανασκόπηση της βιβλιογραφίας και τους πιθανούς κινδύνους εγκυρότητας της έρευνάς μας. Όσον αφορά τη διαδικασία αναζήτησης, οποιαδήποτε μελέτη δεν ανέφερε τη λέξη «API», «Library», «Framework» στον τίτλο του άρθρου, δε συμπεριλήφθηκε στο σύνολο των πρωταρχικών ερευνών. Συνεπώς, μπορεί να έχει παραλειφθεί, ένας μικρός αριθμός σχετικών άρθρων, αν και πιστεύουμε ότι τα άρθρα που ασχολούνται με τα APIs το πιθανότερο είναι ότι θα το αναφέρουν και στον τίτλο τους. Παρόλα αυτά, εμείς θεωρούμε ότι συμπεριλαμβάνοντας στην ανασκόπηση μόνο κορυφαία περιοδικά, συνέδρια και workshops, αυξάνονται τα ποιοτικά κριτήρια των πρωταρχικών μελετών και έτσι εξασφαλίζεται η ποιότητα της συστηματικής μας ανασκόπησης.

ΚΕΦΑΛΑΙΟ 3: ΜΕΛΕΤΗ ΚΑΙ ΚΑΤΑΓΡΑΦΗ ΤΩΝ ΠΛΕΟΝΕΚΤΗΜΑΤΩΝ ΚΑΙ ΜΕΙΟΝΕΚΤΗΜΑΤΩΝ ΤΩΝ APIs

Το κεφάλαιο αυτό στοχεύει στη διερεύνηση των πλεονεκτημάτων και μειονεκτημάτων των APIs. Προκειμένου να επιτευχθεί αυτό, το κεφάλαιο χωρίζεται σε δυο ενότητες, η ενότητα 3.1 μελετά τα πλεονεκτήματα των APIs με βάση τα άρθρα που συλλέχθηκαν και αντίστοιχα η ενότητα 3.2 τα μειονεκτήματα.

3.1 ΠΛΕΟΝΕΚΤΗΜΑΤΑ ΤΩΝ ΔΙΑΣΥΝΔΕΣΕΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΕΦΑΡΜΟΓΩΝ (APIs)

Στην ενότητα αυτή μελετάμε τα πλεονεκτήματα των διασυνδέσεων προγραμματισμού εφαρμογών (APIs), σύμφωνα με τους τέσσερις ερευνητικούς τομείς όπως προέκυψαν στην ενότητα 2.3.1.

3.1.1 Μελέτη των πλεονεκτημάτων του τομέα εφαρμογής των APIs

Στην πρώτη κατηγορία αναλύουμε τα πλεονεκτήματα των 7 μελετών που αφορούν τον τομέα εφαρμογής των APIs. Μια αρχική προσέγγιση των μελετών έγινε στην υποενότητα 2.3.1.1. Τα άρθρα (Pai et al., 2003, Singh et al., 2005, Sudarsan και Ribbens, 2010, Carvalho και Cachopo, 2011, Gotlieb και Bernard, 2006) περιγράφουν τα πλεονεκτήματα σχετικά με τον σχεδιασμό, την εφαρμογή και την επίδοση των διασυνδέσεων προγραμματισμού εφαρμογών.

Τα πλεονεκτήματα του άρθρου (Pai et al., 2003) είναι ότι μια διεπαφή προγραμματισμού εφαρμογών (API) επιτρέπει στο χρήστη, οι μονάδες να φορτώνονται απευθείας στον πυρήνα της μνήμης cache και οι υπηρεσίες να εκτελούνται είτε με το σύστημα της cache ή σε ένα ξεχωριστό διακομιστή. Παρουσιάζεται ένα API που επιτρέπει τον προγραμματισμό του μεσολαβητή της μνήμης cache μετά την εγκατάσταση. Ταυτόχρονα, το API επιτρέπει την εξαιρετικά γρήγορη επικοινωνία μεταξύ της κρυφής μνήμης και των μονάδων των χρηστών χωρίς την ανάγκη για συνδέσεις TCP ή ένα τυποποιημένο σύστημα ανάλυσης. Συγκεκριμένα, το API παρέχει την υποδομή για την επεξεργασία των αιτήσεων HTTP και των απαντήσεων σε υψηλό επίπεδο, προστατεύοντας τους προγραμματιστές από τις χαμηλού επιπέδου λεπτομέρειες της υποδοχής προγραμματισμού, τις HTTP αλληλεπιδράσεις και τη διαχείριση της μνήμης. Το API μπορεί να επιτύχει υψηλή επεκτασιμότητα με τη ελάχιστη επίδραση στην απόδοση στον μεσολαβητή. Η προσέγγιση αυτή απαλλάσσει από τα πολλαπλά

περιβαλλοντικά νήματα ή τις πολλαπλές διεργασίες, επιτρέποντας την επεκτασιμότητα για το λογισμικό του διακομιστή.

Το API περιλαμβάνει ρουτίνες επεξεργασίας κεφαλίδας που επιτρέπει την αναζήτηση για συγκεκριμένες HTTP κεφαλίδες, προσθέτοντας νέες κεφαλίδες, διαγράφοντας υπάρχουσες κεφαλίδες και δηλώνοντας ότι το τρέχον σύνολο των επικεφαλίδων έχει ολοκληρωθεί και μπορεί να σταλεί. Αυτές οι ρουτίνες παρέχουν μια απλή και ευέλικτη διεπαφή για τη μονάδα να προσαρμόζει το HTTP επίπεδο συμπεριφοράς του αιτήματος, μεμονωμένα ή σε συνδυασμό με τη μετατροπή του περιεχομένου του σώματος. Το API παρέχει υπηρεσίες για να ωθήσουν, να ρωτήσουν, να ακυρώσουν ή να τροποποιήσουν δεδομένα, ενώ βρίσκονται στην κρυφή μνήμη και επίσης παρέχει δυνατότητες για την παρακολούθηση σε πραγματικό χρόνο και με την ενσωμάτωση των στατιστικών. Όσον αφορά την κάλυψη, το API έχει σχεδιαστεί ειδικά για την προσωρινή αποθήκευση των διακομιστών μεσολάβησης, περιέχει διαχείριση του περιεχομένου και των λειτουργιών χρηστικότητας που δεν υπάρχουν σε άλλα APIs. Χρησιμοποιώντας αυτό το API, η λειτουργικότητα μπορεί να προστεθεί στον διαμεσολαβητή της cache αφού έχει αναπτυχθεί, χωρίς να χρειάζονται τροποποιήσεις του πηγαίου κώδικα που μπορεί να είναι δύσκολο να διατηρηθεί ή να χρειάζεται εξωτερικούς διακομιστές για την εκτέλεση αυτών των υπηρεσιών, ακόμη και αν η CPU cache δεν αξιοποιείται πλήρως.

Στο (Singh et al., 2005) το OML παρέχει έναν ευέλικτο και δυναμικό τρόπο με τον οποίο τα δεδομένα συλλέγονται και διατίθενται σε πραγματικό χρόνο για την πρόσβαση στους ερευνητές. Το πλεονέκτημα της συλλογής δεδομένων σε πραγματικό χρόνο είναι ότι επιτρέπει διαδραστικά πειράματα στα οποία οι χρήστες μπορούν να αντιδράσουν με τη δυναμική του πειράματος αμέσως, εξοικονομώντας πολύτιμους πόρους. Μπορεί να μειώσει το βάρος της συλλογής των μετρήσεων για τους ερευνητές ώστε να μπορούν να επικεντρωθούν στο πρωτόκολλο και την ανάπτυξη της εφαρμογής χωρίς να ανησυχούν για την πολυπλοκότητα και τις λεπτομέρειες για τη συλλογή, τη μεταφορά και την αποθήκευση των δεδομένων του πειράματος. Είναι φιλικό προς το χρήστη και γενικά τα APIs, μπορούν να ενσωματωθούν εύκολα σε εφαρμογές του χρήστη για την ανάπτυξη εφαρμογών για τη συλλογή και τη μεταφορά των δεδομένων.

Αυτό περιλαμβάνει τον χειρισμό ζητημάτων νημάτωσης που σχετίζονται με τη συλλογή δεδομένων, τα δεδομένα τύπου για την ασφάλεια και την ελάχιστη διαμόρφωση και την πολυπλοκότητα συγκεκριμενοποίησης από την πλευρά του κατασκευαστή της εφαρμογής. Το OML υποστηρίζει πολλαπλά κανάλια πολλαπλής διανομής. Το OML μπορεί να χρησιμοποιηθεί σε διάφορες ενσύρματες και ασύρματες πλατφόρμες δοκιμών δικτύωσης και κατανεμημένα συστήματα για τη συλλογή δεδομένων. Αυτό το πλαίσιο μειώνει το βάρος της συλλογής των δεδομένων σχετικά με τους προγραμματιστές εφαρμογών, παρέχοντας απλά APIs για τη μεταφορά των δεδομένων με αξιόπιστο τρόπο. Η χρηστικότητα του πλαισίου ενισχύεται σημαντικά με τη χρήση τεχνολογιών όπως η SQL, γεγονός το οποίο επιτρέπει τη χρήση των τυποποιημένων εργαλείων για ανάλυση δεδομένων.

Στο (Sudarsan και Ribbens, 2010) οι κύριες συνεισφορές του πλαισίου περιλαμβάνει: (α) έναν χρονοπρογραμματιστή ο οποίος διαχειρίζεται δυναμικά τους πόρους για την εκτέλεση παράλληλων εφαρμογών σε μια ομοιογενή ομάδα, (β) μια αποτελεσματική βιβλιοθήκη χρόνου εκτέλεσης για τον επεξεργαστή αντιστοίχισης και την αναδιανομή των δεδομένων, (γ) ένα απλό μοντέλο προγραμματισμού και εύκολο στη χρήση API για την τροποποίηση των υπαρχόντων επιστημονικών εφαρμογών για την εκμετάλλευση της αλλαγής μεγέθους. Η επανασχεδίαση είναι σε θέση να αλλάζει δυναμικά το μέγεθος του διαμερίσματος μιας εφαρμογής, λαμβάνοντας υπόψη το πλαίσιο που περιγράφεται. Ένα απλό API επιτρέπει οι κώδικες του χρήστη να προσπελάσουν το πλαίσιο επανασχεδίασης και τη βιβλιοθήκη.

Αυτές οι λειτουργίες παρέχουν πρόσβαση στη βασική λειτουργικότητα της βιβλιοθήκης αλλαγής μεγέθους, επικοινωνώντας με τον χρονοπρογραμματιστή, επαναθέτοντας τους επεξεργαστές, μετά την επέκταση ή τη συρρίκνωση και την ανακατανομή των δεδομένων. Η επανασχεδίαση του πλαισίου ευνοεί επίσης της μεγάλης εκτέλεσης εργασίες, καθώς μπορούν να χρησιμοποιήσουν τους πόρους πέρα από την αρχική κατανομή τους, μόλις αυτοί καταστούν διαθέσιμοι. Η επανασχεδίαση πλαισίου επιτρέπει οι επαναληπτικές εφαρμογές να επεκταθούν σε περισσότερους επεξεργαστές.

Στο (Carvalho και Cachopo, 2011) ένα από τα βασικά σημεία των συστημάτων λογισμικού μνήμης συναλλαγών (STM) είναι ότι αυτά απλοποιούν την ανάπτυξη των ανταγωνιστικών προγραμμάτων, επειδή οι προγραμματιστές δεν χρειάζεται να ασχολούνται με ποια αντικείμενα έχουν πρόσβαση ταυτόχρονα. Μία από τις συνέπειες των αποφάσεων στο STM είναι η πλήρη διαφάνεια για τον προγραμματιστή, είναι ότι μπορεί να αναλάβει σε μεγάλο βαθμό την επιβάρυνση. Η θύρα JWormBench, σχεδιάστηκε για να είναι εύκολα επεκτάσιμη και να επιτρέπει την εύκολη ενσωμάτωση με διαφορετικά STMs. Ένα καλό σημείο αναφοράς είναι αρκετά ευέλικτο για να επιτρέπει την ενσωμάτωση των νέων μηχανισμών συγχρονισμού χωρίς να απαιτούνται αλλαγές στον πηγαίο κώδικά του και να παρέχει έναν μηχανισμό συγχρονισμού που βασίζεται σε μια προσέγγιση, η οποία μπορεί να παρουσιάζει μια καλή επίδοση. Το σημείο αναφοράς WormBench έχει ένα μοντέλο χαμηλής πολυπλοκότητας τομέα καθιστώντας το εύκολο να το καταλάβουμε και έχει ένα απλό API. Το JWormBench παρέχει ένα απλό API, εύκολο να ενσωματωθεί σε οποιαδήποτε εφαρμογή STM σε Java. Έτσι, ο καθένας μπορεί να προσθέσει ένα νέο μηχανισμό συγχρονισμού (με βάση τα STM ή άλλα), την εφαρμογή των κατάλληλων αφηρημένων τύπων και την παροχή αυτών των εφαρμογών σε JWormBench μέσω μιας μονάδας διαμόρφωσης. Η βασική μηχανή του JWormBench σημείου αναφοράς έχει αναπτυχθεί σε μια ξεχωριστή και ανεξάρτητη βιβλιοθήκη, της οποίας τα στοιχεία μπορούν να επεκταθούν με άλλες βιβλιοθήκες.

Στο άρθρο (Gotlieb και Bernard, 2006) με τη χρήση διαφόρων εργαλείων, έχουμε σχεδιάσει ένα πειραματικό περιβάλλον για τη δημιουργία του ημι-εμπειρικού μοντέλου και εφαρμόζονται οι συμμετρικές δοκιμές σε καταστάσεις όσο το δυνατόν πλησιέστερα στις πραγματικές καταστάσεις. Η αρχή της συμμετρικής δοκιμής έχει ως στόχο την εξεύρεση πλήθος παραδειγμάτων (που ονομάζονται παραβιάσεις της συμμετρίας) για τη σχέση συμμετρίας που ένα πρόγραμμα πρέπει να ικανοποιήσει. Η διαδικασία επικύρωσης της Java Card των API συνήθως αποτελείται από δύο διακριτές φάσεις: πρώτον, η Java κάρτα ελέγχει βοηθητικές εφαρμογές που αναπτύχθηκαν σε έναν κεντρικό υπολογιστή, χρησιμοποιώντας τις βιβλιοθήκες προσομοίωσης. Δεύτερον, οι δοκιμές που εφαρμόζονται σε μια προσομοίωση κώδικα που τρέχει σε μια κάρτα εξομοιωτή και η εκτέλεση δοκιμής που διεξάγεται από πολλαπλές δοκιμές στην κάρτα Java.

Στο άρθρο (Bellotti et al., 2008) αναλύονται τα πλεονεκτήματα της τεχνολογίας RFID. Το oDect στοχεύει στο να επιτρέπει στους προγραμματιστές εφαρμογής να δημιουργήσουν τα πρωτότυπα τους, εστιάζοντας στις ανάγκες των τελικών χρηστών, χωρίς να νοιάζονται για το χαμηλού επιπέδου λογισμικό που αλληλεπιδρά με το υλικό του RFID. Το EUD έχει σκοπό να επιτρέψει στους τελικούς χρήστες να εφαρμόσουν ή / και προσαρμόσουν εφαρμογές λογισμικού. Υπάρχουν δύο κύριες προσεγγίσεις για την ανάπτυξη αυτού του είδους τα συστήματα: το πρώτο στοχεύει στο να κάνει τους υπολογιστές να είναι σε θέση να συμπεράνουν τον κώδικα εφαρμογής μαθαίνοντας από τα παραδείγματα των χρηστών, το δεύτερο σκοπεύει να μειώσει την καμπύλη εκμάθησης των γλωσσών προγραμματισμού βασιζόμενη σε εύκολες στην κατανόηση, επαναχρησιμοποιήσιμα στοιχεία και γραφικές μεταφορές. Το oDect API, στην πραγματικότητα, έχει θεωρηθεί ως ένα εργαλείο προγραμματισμού, που εκθέτει τις λειτουργίες που μπορούν να ενεργοποιηθούν εύκολα και να παραμετροποιηθούν από έναν τελικό χρήστη μέσω μιας απλής αλληλεπίδρασης.

Αυτή η ένα-προς-ένα αντιστοιχία μεταξύ RFID των APIs και των εντολών της διεπαφής χρήστη επιτρέπει στους ίδιους τους χρήστες να ελέγχουν άμεσα και να προσαρμόζουν τις παρεχόμενες λειτουργίες και να κάνουν το έργο της ανάπτυξης εφαρμογών πολύ απλούστερο. Το oDect των APIs εφαρμόζεται στις Μικροπολυμεσικές Γλωσσικές Υπηρεσίες (MSL- Micromultimedia Services Language). Η MSL προσφέρει υψηλού επιπέδου αντικείμενα (στοιχεία) τα οποία ενσωματώνουν προηγμένες υπηρεσίες. Αυτά τα στοιχεία μπορούν εύκολα να ενσωματωθούν σε εφαρμογές πολυμέσων. Η MSL μπορεί να επεκταθεί σταδιακά με την προσθήκη στοιχείων που παρέχουν νέες υπηρεσίες και λειτουργίες. Η προτεινόμενη RFID API υποστηρίζει την ανάπτυξη των πλήρως λειτουργικά πρωτότυπων σε σύντομο χρονικό διάστημα της ανάπτυξης. Τα APIs υποστηρίζουν αποτελεσματική πρωτοτύποποιήση για κοινές κλάσεις της RFID-με βάση εφαρμογές.

Το τελευταίο άρθρο αυτής της κατηγορίας (Shan και Hua, 2006) μελετά τα πλεονεκτήματα των πλαισίων διαδικτυακής εφαρμογής. Ένα Πλαίσιο Διαδικτυακής Εφαρμογής (WAF) είναι μια επαναχρησιμοποιήσιμη, ημι-πλήρης αρθρωτή πλατφόρμα που είναι εξειδικευμένη να παράγει εξειδικευμένες εφαρμογές Ιστού

που εξυπηρετούν συνήθως τους φυλλομετρητές Ιστού μέσω του http πρωτοκόλλου. Τα πλαίσια λογισμικού μειώνουν σημαντικά τον χρόνο, τη προσπάθεια και τους πόρους που απαιτούνται για την ανάπτυξη και τη συντήρηση εφαρμογών Ιστού. Το πλαίσιο είναι μια ανοικτή αρχιτεκτονική που βασίζεται σε κοινώς αποδεκτά πρότυπα και τεχνολογίες επιτρέποντας σε κάθε έμπειρο προγραμματιστή την ταχεία ανάπτυξη και την υποστήριξη του συστήματος, χωρίς μια απότομη καμπύλη μάθησης. Η προσέγγιση αυτή μειώνει τον κίνδυνο δεδομένου ότι τα βιομηχανικά πρότυπα πλαισίων ανοικτής πηγής είναι ενεργά διατηρημένα και ενισχυμένα από τους πολύ καλά εξειδικευμένους επαγγελματίες στον κόσμο. Με τη χρήση τυποποιημένων τεχνολογιών ένα πλαίσιο μπορεί εύκολα να αναπτυχθεί με την υπάρχουσα υποδομή αξιοποιώντας το υπάρχον υλικό, λογισμικό, τις διαδικασίες και τους ανθρώπους. Ένα πλαίσιο εφαρμογής Ιστού μπορεί επίσης να περιλαμβάνει ένα στοιχείο που βασίζεται στην παρουσίαση της απόδοσης του πλαισίου, που επιτρέπει στους προγραμματιστές να επεκτείνουν τα υπάρχοντα στοιχεία της διεπαφής χρήστη ή την κατασκευή νέων στοιχείων που μπορούν να επαναχρησιμοποιηθούν σε όλη την εφαρμογή.

3.1.2 Μελέτη των πλεονεκτημάτων του τομέα ποιότητας των APIs

Η δεύτερη κατηγορία μελετά τα πλεονεκτήματα των ποιοτικών χαρακτηριστικών των APIs. Αυτός ο τομέας περιλαμβάνει τις 6 μελέτες που διερευνήθηκαν στην υποενότητα 2.3.1.2. Στα άρθρα (Stylos et al., 2008, Kim et al., 2011, Stylos και Myers, 2008, Hoffman και Strooper, 2000) μελετάμε τα πλεονεκτήματα της δυνατότητας επαναχρησιμοποίησης των APIs, της βελτίωσης της χρηστικότητας των APIs, καθώς επίσης και των επανασχεδιάσεων στην ποιότητα του λογισμικού.

Στα πλεονεκτήματα του άρθρου (Stylos et al., 2008) συγκαταλέγεται ότι τα APIs και άλλοι κώδικες προορίζονται για επαναχρησιμοποίηση. Ένα από τα βασικά στοιχεία της παραγωγικότητας των προγραμματιστών είναι ότι, είναι σε θέση γρήγορα και αποτελεσματικά να επαναχρησιμοποιήσουν κώδικα που έχουν ήδη γράψει οι συνάδελφοι τους. Τα APIs επιτρέπουν κώδικα που θα πρέπει να επαναχρησιμοποιείται από άλλους προγραμματιστές. Η αξιολόγηση της ευχρηστίας των προτεινόμενων αλλαγών για το API, δείχνει ότι η διαδικασία σχεδιασμού με επίκεντρο τον χρήστη ήταν επιτυχής, συμβάλλοντας στη

δημιουργία ενός API που βελτιώνει σημαντικά την παραγωγικότητα των χρηστών και ταιριάζει καλύτερα με τις διαφορετικές ανάγκες των χρηστών. Το BRFplus API επιτρέπει πρόσθετη λειτουργικότητα που θα καθοριστεί από επαγγελματίες χρήστες, οι οποίοι γνωρίζουν τις λεπτομέρειες του τι πρέπει να κάνει το λογισμικό, αντί των προγραμματιστών λογισμικού. Αυτό παρέχει τη δυνατότητα για μια ισχυρή προσαρμογή του λογισμικού στα χέρια του επαγγελματία χρήστη, που ενεργοποιείται από το BRFplus. Κάνοντας το BRFplus πιο εύκολο στη χρήση, προσφέροντας μεγαλύτερη ευελιξία και αξία στους πελάτες του SAP.

Στο (Kim et al., 2011) η επανασχεδίαση βελτιώνει την ποιότητα του λογισμικού καθιστώντας ευκολότερη τη διατήρηση και τη κατανόηση των συστημάτων του λογισμικού. Οι επανασχεδιάσεις σε επίπεδο API βελτιώνουν την παραγωγικότητα για την ανάπτυξη. Ο χρόνος που απαιτείται για να διορθωθούν αυτά τα σφάλματα μειώνεται περίπου από 35,0% στο 63,1% μετά τις επανασχεδιάσεις. Η αναθεώρηση της διόρθωσης των δεδομένων είναι αρκετά ακριβής για να στηρίξει την έρευνα και τα δεδομένα της αναθεώρησης, να έχουν μια υψηλή ακρίβεια και μια ανάκληση σχετικά με τις επανασχεδιάσεις σε επίπεδο API. Οι προγραμματιστές ανακαλύπτουν σφάλματα κατά τη διάρκεια των επανασχεδιάσεων και τα επιλύουν γρήγορα, κάνουν αλλαγές στον κώδικα που εμπλέκεται ή αναγνωρίζουν γρήγορα ελλιπείς επανασχεδιάσεις. Οι επανασχεδιάσεις σε συνδυασμό με διορθώσεις σφαλμάτων επιβαρύνονται συχνά με συμπληρωματικές διορθώσεις, οι οποίες είναι συνήθως μικρότερες και ευκολότερο να εφαρμοστούν από τις κύριες διορθώσεις των σφαλμάτων. Όταν πρόκειται για επίλυση σφαλμάτων που εισάγονται κοντά στο χρόνο των επανασχεδιάσεων ο μέσος χρόνος επίλυσης τείνει να μειώνεται μετά τις επανασχεδιάσεις.

Στο (Stylos και Myers, 2008) οι προγραμματιστές πράγματι κλίνουν προς τις ίδιες κλάσεις έναρξης, χρησιμοποιώντας τις μεθόδους αυτής της αρχικής κλάσης για να εξερευνήσουν το API που ήταν σημαντικά ταχύτερες μεταξύ 2 και 11 φορές πιο γρήγορες στο τμήμα της εργασίας, το οποίο απαιτεί το συνδυασμό των αντικειμένων χρησιμοποιώντας τα APIs, όπου η αρχική κλάση περιείχε αναφορά σε μεθόδους των βοηθητικών κλάσεων (και όχι το αντίστροφο). Οι μέθοδοι μπορεί να μην τοποθετούνται στην πιο ανακαλύψιμη κλάση έτσι ώστε να διαφυλαχθεί η

απόκρυψη των πληροφοριών. Τα APIs για κάθε κατάσταση ήταν πλήρως λειτουργικά, έτσι ώστε τα προγράμματα των συμμετεχόντων θα μπορούσαν να μεταγλωττιστούν και να λειτουργήσουν πραγματικά. Οι προγραμματιστές ήταν πιο γρήγοροι χρησιμοποιώντας τα APIs σε αυτές τις κλάσεις από τις οποίες άρχισαν την εξερεύνηση τους περιλαμβάνοντας αναφορές στις άλλες κλάσεις που χρειάζονται.

Στο άρθρο (Hoffman και Strooper, 2000) οι βιβλιοθήκες επαναχρησιμοποιήσιμων στοιχείων υπάρχουν τώρα για πολλές αντικειμενοστρεφείς γλώσσες και η Java έχει αυξήσει τη φορητότητα των στοιχείων και ως εκ τούτου, τη δυνατότητα για επαναχρησιμοποίηση. Δεδομένου ότι η εξασφάλιση της αξιοπιστίας του λογισμικού εξαρτάται εν μέρει από τη δοκιμή, η αποτελεσματική επαναχρησιμοποίηση στοιχείου εξαρτάται από πρακτικές προσεγγίσεις για τη δοκιμή στοιχείων. Ο αποτελεσματικός έλεγχος των Java APIs είναι εφικτός. Το Roast παράγει οδηγούς ελέγχου από σενάρια δοκιμής με ενσωματωμένες περιπτώσεις δοκιμών και παρέχουν υποστήριξη για τη δημιουργία οριακών τιμών και τους συνδυασμούς αυτών των οριακών τιμών. Με την παροχή των διαφόρων αλγορίθμων για το συνδυασμό οριακών τιμών, ο ελεγκτής μπορεί να αναπτύξει γρήγορα τους οδηγούς δοκιμών και να πειραματιστούν με μια ποικιλία περιπτώσεων δοκιμών, εξισορροπώντας τις απαιτήσεις δοκιμών για το περιεχόμενο, τον χρόνο εκτέλεσης και τον διαθέσιμο χώρο.

Στο άρθρο (Liu et al., 2009) οι δοκιμές του λογισμικού είναι ένας από τους βασικούς παράγοντες για την ανάπτυξη του λογισμικού και τη συντήρηση. Υπάρχουν πολλοί παράγοντες για τη βελτίωση της ποιότητας του λογισμικού, όπως η αναθεώρηση και ο έλεγχος. Σε ορισμένες κρίσιμες εφαρμογές και περιοχές, όπως σε πραγματικό χρόνο ελέγχου του συστήματος, η διαδικασία της δοκιμής του λογισμικού είναι ανώτερη από τη διαδικασία της ανάπτυξης. Επίσης, η σημασία του ελέγχου της ποιότητας του λογισμικού είναι πιο σημαντική από τις δοκιμές του λογισμικού και είναι το πλέον αποτελεσματικό μέσο για τον έλεγχο της ποιότητας λογισμικού. Ένα μοντέλο συνεργασίας μπορεί να επαναχρησιμοποιήσει υπάρχουσες πλατφόρμες δοκιμών. Το μοντέλο της οντολογίας επιλύει τη σημασιολογική περιγραφή για τις πλατφόρμες δοκιμής. Οι μηχανικοί μπορούν να βρουν τις καταλληλότερες λέξεις-κλειδιά μέσα από την περιγραφική πληροφορία.

Το μοντέλο της ταξινόμησης βελτιώνει το επίπεδο συνεργασίας μεταξύ των πλατφόρμων δοκιμής. Οι λειτουργίες στις παρόμοιες περιοχές εφαρμογής μπορεί να είναι συνεργατικές μεταξύ των πλατφόρμων δοκιμής τους, ιδιαίτερα σε σενάρια δοκιμών και διαδικασίες δοκιμής.

Στο τελευταίο άρθρο αυτής της κατηγορίας (Zibran et al., 2011) ένα API μπορεί να θεωρηθεί χρήσιμο, όταν παρέχει σωστά την επιθυμητή λειτουργικότητα και την αποτελεσματικότητα όσον αφορά την επίδοση (όσον αφορά την κατανάλωση των πόρων). Οι σχεδιαστές του API λαμβάνουν υπόψη τα κριτήρια του σχεδιασμού, όπως την επαναχρησιμοποίηση και τη συνεκτίμηση που ωφελούν κυρίως αυτούς που εμπλέκονται στην ανάπτυξη και τη συντήρηση των APIs. Οι σχεδιαστές του API και οι προγραμματιστές χρειάζονται μια καλή κατανόηση σχετικά με τη χρηστικότητα του API και να την εφαρμόσουν στις φάσεις του σχεδιασμού και της ανάπτυξης, έτσι ώστε να μπορούν να ελαχιστοποιήσουν τις δυσκολίες συντήρησης που προκαλούνται από τα θέματα που σχετίζονται με τη χρηστικότητα τέτοιων API.

3.1.3 Μελέτη των πλεονεκτημάτων του τομέα ανάλυσης των APIs

Σε αυτό το τομέα μελετάμε τα πλεονεκτήματα που σχετίζονται με την ανάλυση των APIs. Στην ενότητα αυτή, ανήκουν τα 23 άρθρα που αναλύθηκαν στην υποενότητα 2.3.1.3. Πιο συγκεκριμένα στα (Sarkar et al., 2007, Tselikas et al., 2007, Fujiwara et al., 2003 και Ratiu και Jurjens, 2008) παρουσιάζονται ζητήματα που σχετίζονται με τα πλεονεκτήματα των μετρικών στις διεπαφές προγραμματισμού εφαρμογών.

Στο (Sarkar et al.,2007) η χαμηλή σύζευξη της ενδομονάδας, η υψηλή συνοχή της ενδομονάδας και η χαμηλή πολυπλοκότητα ανέκαθεν θεωρούνται σημαντικά χαρακτηριστικά οποιουδήποτε διαμορφωμένου λογισμικού. Η προώθηση της ενθυλάκωσης μονάδων (π.χ., απόκρυψη πληροφοριών), χωρίζοντας την διεπαφή της μονάδας από την εφαρμογή. Σε ένα καλά οργανωμένο σύστημα, μόνο τα στοιχεία της διεπαφής είναι ορατά σε άλλες μονάδες. Η εφαρμογή περιλαμβάνει το λειτουργικό κώδικα που αντιστοιχεί στα στοιχεία που δηλώνονται στη διεπαφή, μια τέτοια διασύνδεση της μονάδας που είναι γνωστή ως API. Οι διάφορες διαστάσεις κατά μήκος των οποίων η ποιότητα του λογισμικού έχει βελτιωθεί με την ενθυλάκωση που παρέχονται από τη

διαμόρφωση περιλαμβάνουν, την ελεγξιμότητα, την εναλλαξιμότητα, την αναλυτικότητα και την συντηρησιμότητα. Όταν οι μονάδες μπορούν να μεταγλωττιστούν ανεξάρτητα η μια από την άλλη, τότε, για όσο διάστημα η μονάδα των API δεν αλλάζει, οι άλλες μονάδες μπορούν να αγνοούν την εξέλιξη των εσωτερικών λεπτομερειών της κάθε μονάδας. Μια συνεκτική μονάδα που είναι απλά χαλαρά συνδεδεμένη με άλλες μονάδες παρουσιάζει λιγότερη εξάρτηση από τις άλλες μονάδες. Κατά συνέπεια, η μεταγλώττιση, η επέκταση και ο έλεγχος θα έχουν πολύ μικρότερο αντίκτυπο στις άλλες μονάδες. Οι μετρικές με βάση το μέγεθος, συχνά δεν υποβαθμίζονται όταν πάμε από την ανθρώπινη διαμόρφωση που παρέχονται στο τυχαίο σενάριο. Οι δομικές μετρικές καθοδηγούνται από την έννοια του API, μια έννοια κεντρικής σημασίας για τη σύγχρονη ανάπτυξη λογισμικού, βασίζονται σε έννοιες, όπως οριοθέτηση μεγέθους, ομοιομορφία μεγέθους, λειτουργική αποτελεσματικότητα κατά τις πολυεπίπεδες αρχιτεκτονικές και ομοιότητα του σκοπού διαδραματίζουν σημαντικό ρόλο υποστήριξης.

Στο (Tselikas et al., 2007) η ζήτηση για υπηρεσίες προστιθέμενης αξίας δημιουργεί σημαντικές ευκαιρίες για τους φορείς που αποφασίζουν να ανοίξουν τα δίκτυά τους σε εξωτερικούς παρόχους υπηρεσιών. Αυτή η διαδικασία είναι πρακτικά εφικτή μόνο εάν χρησιμοποιούνται τυποποιημένες διεπαφές για τις λειτουργίες του δικτύου πυρήνα. Με αυτό τον τρόπο το κόστος και η πολυπλοκότητα της ανάπτυξης και η διαδικασία ολοκλήρωσης μειώνεται, καθώς η επιβάρυνση διαχείρισης της υπηρεσίας, η οποία κατανέμεται μεταξύ των παρόχων υπηρεσιών δεν επιβαρύνει τον χειριστή του δικτύου. Οι περισσότερες υπάρχουσες λύσεις για την τυποποιημένη πρόσβαση στις λειτουργίες του δικτύου πυρήνα ακολουθούν το παράδειγμα ενδιάμεσου λογισμικού που ενσωματώνει τις λειτουργίες του δικτύου και παρέχει ανοικτές διασυνδέσεις με εξωτερικούς φορείς παροχής υπηρεσιών. Οι διασυνδέσεις αυτές προβάλλουν ένα απλουστευμένο μοντέλο για τις λειτουργίες δικτύου πυρήνα και είναι τυποποιημένες ώστε να επιτρέπουν την επαναχρησιμοποίηση και την ανεξαρτησία από τις βασικές αρχιτεκτονικές δικτύου. Αυτή η νέα προσέγγιση κρύβει τις βασικές ιδιαιτερότητες του δικτύου στο επίπεδο παροχής υπηρεσιών με την έκθεση μόνο της πραγματικά απαιτούμενης λειτουργικότητας για την δημιουργία και την παροχή νέων υπηρεσιών. Τα ανοιχτά APIs, όπως OSA / Parlay και Java APIs για ολοκληρωμένα δίκτυα (JAIN) προτιμούνται σε πολλές εφαρμογές. Οι

προδιαγραφές του Parlay API έχουν αναπτυχθεί και καθορισθεί, αφήνοντας τις υπηρεσίες και τις εφαρμογές να έχουν πρόσβαση με διαφάνεια στη λειτουργικότητα του δικτύου πυρήνα. Ο στόχος του JAIN είναι να παρέχει φορητότητα υπηρεσιών και την ασφαλή πρόσβαση σε τέτοια ολοκληρωμένα δίκτυα. Τέτοιες πλατφόρμες ή ακόμη και εφαρμογές ενδιάμεσου λογισμικού αναπτύχθηκαν από το μηδέν με βάση ανοικτά APIs όπως η OSA / Parlay ή JAIN, που δημιουργούν ένα οικονομικά αποδοτικό και εξαιρετικά ευέλικτο IP βασισμένο στην υποδομή για την παροχή υπηρεσιών. Η χρήση των τυποποιημένων διεπαφών που αλληλεπιδρούν με το δίκτυο είναι ένας σημαντικός παράγοντας που συμβάλλει στη διαφάνεια, τη διαμόρφωση, την επαναχρησιμοποίηση και τη σαφή κατανομή των διαχειριστικών αρμοδιοτήτων για τη διαδικασία της ανάπτυξης υπηρεσιών.

Στο άρθρο (Fujiwara et al., 2003) η αντικειμενοστρεφής ανάπτυξη κατασκεύασε το λογισμικό με την επαναχρησιμοποίηση διάφορων τμημάτων λογισμικού τα οποία έχουν υψηλή συνοχή και είναι εύκολο να συνδυαστούν. Με την επαναχρησιμοποίηση των στοιχείων, που έχουν υψηλή ποιότητα, η βελτίωση της ποιότητας του λογισμικού θα πρέπει να επιτευχθεί και η περίοδος ανάπτυξης θα μειωθεί επίσης. Από την άποψη της εξοικονόμησης κόστους και τη βελτίωση της παραγωγικότητας, είναι σαφές ότι το πλαίσιο με βάση την επαναχρησιμοποίηση είναι πιο κατάλληλο από τη μονάδα με βάση την επαναχρησιμοποίηση. Τα ακόλουθα πλεονεκτήματα στην ανάπτυξη εφαρμογών που χρησιμοποιούν το πλαίσιο είναι: (α) μείωση της αναπτυξιακής προσπάθειας της μονάδας ελέγχου, από τη λογική του ελέγχου έχει τεθεί σε εφαρμογή στο πλαίσιο, η προσπάθεια πρέπει να μειωθεί, (β) βελτίωση της δυνατότητας επαναχρησιμοποίησης EJB στοιχεία. Από τα στοιχεία EJB με τη διεπαφή που προβλέπεται στο πλαίσιο, την επαναχρησιμοποίηση / ανταλλαγή μεταξύ των στοιχείων γίνεται εύκολη, (γ) μείωση της προσπάθειας αλλαγής των απαιτήσεων για τη μετάβαση στην οθόνη. Οι C και K μετρικές έδειξαν ότι είναι καλύτερης πρόβλεψης της αποτυχίας της κλάσης από τις παραδοσιακές μετρικές κώδικα. Από την άποψη της OOFP (Object-Oriented Function Point), το πλαίσιο με βάση την επαναχρησιμοποίηση είναι περίπου 2,5 φορές πιο αποτελεσματικό από τη συμβατική επαναχρησιμοποίηση. Όσον αφορά την πολυπλοκότητα, η τιμή στο πλαίσιο που βασίζεται στην επαναχρησιμοποίηση είναι υψηλότερη από την

μονάδα με βάση την επαναχρησιμοποίηση. Έτσι, αν η ποιότητα του πλαισίου είναι καλή, η πολυπλοκότητα δεν επηρεάζει την ποιότητα του συνόλου των προγραμμάτων εφαρμογής.

Στο (Ratiu και Jurjens, 2008) οι βιβλιοθήκες διαδραματίζουν σημαντικό ρόλο κατά την ανάπτυξη του λογισμικού, από τον σχεδιασμό και την ανάπτυξη μέχρι την εξέλιξη και τις φάσεις συντήρησης. Εκτός από τη λειτουργία τους ως μια διεπαφή μεταξύ των ανθρώπων και των μηχανών, οι βιβλιοθήκες διαδραματίζουν σημαντικό ρόλο στην επικοινωνία ανάμεσα στους προγραμματιστές. Ένα χρήσιμο API παρουσιάζει: καλή ορατότητα - οι χρήστες μπορούν εύκολα να δουν τι μπορεί να χρησιμοποιηθεί. Καλό εννοιολογικό μοντέλο - τη χρήση των χρήσιμων, συνεπείς και πλήρεις αφαιρέσεις που αντιπροσωπεύουν έννοιες ότι οι χρήστες είναι εξοικειωμένοι με ότι τους βοηθάει να δημιουργήσουν ένα κατάλληλο νοητικό μοντέλο του συστήματος και μια καλή χαρτογράφηση – τη χρήση των φυσικών αναλογίων για την αναπαράσταση των εννοιών, των δράσεων και των αποτελεσμάτων. Προσδιορίζεται ένα εκτεταμένο σύνολο των επιθυμητών ιδιοτήτων των βιβλιοθηκών λογισμικού, όπως: οι βιβλιοθήκες είναι συνεπής (δηλαδή κάθε πτυχή της βιβλιοθήκης ακολουθεί μια ενιαία προσέγγιση), εύκολες στην εκμάθηση για τους αρχάριους χρήστες, εύκολες στη χρήση (δηλαδή τις πληροφορίες και τον κώδικα είναι εύκολο να τα βρει κανείς), με δυνατότητα επέκτασης, διαισθητική (δηλαδή ο σχεδιασμός αντιστοιχεί στην διαίσθηση ενός ειδικού τομέα).

Τα άρθρα (Farooq et al., 2010, Rao και Kak, 2011, Kawrykow και Robillard, 2009) παρουσιάζουν τα πλεονεκτήματα όσον αφορά τους αλγόριθμους εντοπισμού σφαλμάτων καθώς επίσης, και την ανίχνευση και διόρθωση σφαλμάτων. Στο (Farooq et al., 2010) οι αναθεωρήσεις ίδιας χρηστικότητας API παρέχουν μια χαμηλού κόστους, αποτελεσματική και με κλιμακούμενο τρόπο για ομάδες προϊόντων λήψη χρηστικότητας εισόδου στα API. Η μέθοδος εκπαιδεύει την ομάδα του προϊόντος, (α) να ευαισθητοποιηθεί στη χρηστικότητα των βέλτιστων πρακτικών και (β) να ενσωματώσει μια τέτοια γνώση (χρηστικότητα βέλτιστων πρακτικών), ως αναπόσπαστο μέρος της πρακτικής του σχεδιασμού τους. Το αποτέλεσμα της αναθεώρησης της ίδιας χρηστικότητας API είναι το ίδιο με τις παραδοσιακές μεθόδους ελέγχου ευχρηστίας. Δεδομένου του στόχου για την

έρευνα χρηστικότητας βρέθηκαν νέα σφάλματα χρηστικότητας για να ενισχύσουν τη χρηστικότητα του API - το αποτέλεσμα αυτό δείχνει ότι αυτές οι αξιολογήσεις από ίδιας χρηστικότητας API ήταν μια χρήσιμη μέθοδος. Οι API αξιολογήσεις ίδιας χρηστικότητας είναι πιο αποτελεσματικές και έτσι έχουν ένα πλεονέκτημα παραγωγικότητας πάνω από τις API δοκιμές ευχρηστίας. Οι API αξιολογήσεις ίδιας χρηστικότητας είναι σημαντικά πιο αποτελεσματικές από ό,τι οι API δοκιμές ευχρηστίας και έτσι παρέχουν μια πιο κλιμακωτή μέθοδο για την αξιολόγηση των APIs. Οι αξιολογήσεις ίδιας API χρηστικότητας έχει αποδειχθεί ότι είναι μια αποτελεσματική μέθοδος για τον εντοπισμό νέων σφαλμάτων χρηστικότητας.

Στο (Rao και Kak, 2011) οι προσεγγίσεις ανάκτησης πληροφοριών (Information Retrieval - IR) βασίζονται σε εντοπισμό σφαλμάτων που δεν είναι μόνο ανεξάρτητες από τη γλώσσα προγραμματισμού και τις έννοιες των επιχειρήσεων του λογισμικού συστήματος, είναι επίσης εξελίξιμες και επεκτάσιμες σε μεγάλα συστήματα λογισμικού. Οι προσεγγίσεις που βασίζονται σε IR επίσης τείνουν να είναι πιο γενικές και μπορεί να χρησιμοποιηθούν για τη διάγνωση μετά από σφάλματα, την έλλειψη συμπεριφοράς εκτέλεσης του προγράμματος που δεν είναι διαθέσιμη. Επομένως, δεν εκπλήσσει το γεγονός ότι οι IR με βάση τις μεθόδους εφαρμόζονται σε άλλους τομείς της συντήρησης λογισμικού και την κατανόηση του προγράμματος. Τα απλά μοντέλα κείμενου όπως το UM και το VSM είναι πιο αποτελεσματικά στη σωστή ανάκτηση των σχετικών αρχείων από μια βιβλιοθήκη σε σύγκριση με τα πιο εξελιγμένα μοντέλα όπως το LDA. Το VSM είναι ο απλούστερος τρόπος για να παρουσιαστεί ο σκοπός της ανάκτησης πληροφοριών. Το πλεονέκτημα του μοντέλου VSM είναι η απλότητα των υπολογισμών στην κατασκευή του μοντέλου και η ευκολία με την οποία ένα ερώτημα μπορεί να συγκριθεί με τα έγγραφα. Το μοντέλο LDA είναι σε θέση να συγκεντρώσει μαζί τα αρχεία που ανήκουν σε διαφορετικά έργα κάτω από ξεχωριστά θέματα. Η προσέγγιση IR με βάση τις τεχνικές εντοπισμού σφαλμάτων είναι τουλάχιστον εξίσου αποτελεσματική με τα στατικά και τα δυναμικά εργαλεία εντοπισμού σφαλμάτων που αναπτύχθηκαν στο παρελθόν.

Στο (Kawrykow και Robillard, 2009) η λειτουργικότητα που παρέχεται από τις βιβλιοθήκες καθίσταται διαθέσιμη μέσω των διασυνδέσεων προγραμματισμού εφαρμογών, ή API. Η κύρια υπόθεση που διέπει την τεχνική είναι ότι ο κώδικας

πελάτη μιμείται τη λειτουργικότητα που παρέχεται με τη μέθοδο του API, μπορεί ενδεχομένως να βελτιωθεί με την κλήση της ίδιας της μεθόδου, αντί της μίμησης της. Υλοποιήσαμε την υποστήριξη για την προσέγγιση με τη μορφή του iMaus, ένα εργαλείο για τη βελτίωση της χρήσης του API. Το iMaus επιτρέπει στους χρήστες να εντοπίζουν πιθανές απομιμήσεις μεθόδων του API στα έργα Eclipse και να διερευνήσει τη δομή και την ποιότητα των αναφερόμενων απομιμήσεων. Οι χρήστες μπορούν να αξιοποιήσουν το iMaus για να ανακαλύψουν νέες απομιμήσεις στο πλαίσιο των έργων ή για τον εντοπισμό περιπτώσεων των προηγουμένων επικυρωμένων προτύπων. Το iMaus μπορεί να χρησιμοποιηθεί για τη σάρωση είτε ενός ενιαίου αρχείου, ή ένα ολόκληρο πακέτο ή έργο. Το iMaus υλοποιεί τη στατική ανάλυση που απαιτείται από τις ευρετικές ανίχνευσης με τη μεταγλώττιση και την ανάλυση πηγαίου κώδικα της Java. Η τρέχουσα έκδοση της προσέγγισης είναι το αποτέλεσμα της εκτεταμένης επαναληπτικής ανάπτυξης με στόχο τη μεγιστοποίηση του αριθμού των διαπιστωμένων απομιμήσεων, ενώ περιορίζει τον αριθμό των ψευδώς θετικών αποτελεσμάτων (μέσω φιλτραρίσματος ευρετικών). Σε κάθε περίπτωση, η προσέγγιση θα είναι επωφελής για πολύ μεγάλα συστήματα που είχαν μια σημαντική εξέλιξη.

Τρία άρθρα αυτής της κατηγορίας τα (Perkins, 2005, Xing και Stroulia, 2007, Gharaibeh et al., 2007) ασχολούνται με τα πλεονεκτήματα της επανασχεδίασης των APIs. Το (Perkins, 2005) έχει πολλά πλεονεκτήματα. Δεν απαιτεί καμία αλλαγή στην πρακτική ανάπτυξης των βιβλιοθηκών, καθώς οι προγραμματιστές έχουν ήδη προσαρμόσει το σώμα της μεθόδου για να γράψουν τα παραδείγματα κώδικα και δεν υπάρχουν νέα εργαλεία ή γλώσσες για να μάθουν. Δεν απαιτεί τη διανομή των νέων αντικειμένων, καθώς ένα εργαλείο για να εφαρμοστεί μπορεί να είναι ελαφρύ. Έγινε αξιολόγηση της δυνατότητας εφαρμογής της προσέγγισης σε μια σειρά από βιβλιοθήκες και διαπιστώθηκε ότι είναι εφαρμόσιμη σε περισσότερες από το 75% των περιπτώσεων. Είναι μια τεχνική που επιτρέπει στις επανασχεδιάσεις να δημιουργούνται αυτόματα για τους πιο κοινούς τύπους αλλαγών των βιβλιοθηκών του API, χωρίς καμία επιπλέον εργασία από την πλευρά του κατασκευαστή της βιβλιοθήκης. Ένα εργαλείο μπορεί να διαβάσει εύκολα το σώμα του bytecode της απαρχαιωμένης μεθόδου και να το μετατρέψει σε κώδικα Java. Το αποτέλεσμα είναι πιθανό να είναι κατανοητό, με λίγες ή καθόλου μεταβλητές. Επειδή ένα εργαλείο που βασίζεται σε αυτή τη τεχνική

μπορεί να είναι απλό και αυτόματο, είναι εύκολο για τον συντηρητή-πελάτη να το τρέξει και μπορεί να ενσωματωθεί με τη διεπαφή του χρήστη ή ένα ολοκληρωμένο περιβάλλον ανάπτυξης για την εμφάνιση των αλλαγών ή ρωτώντας τον χρήστη. Η τεχνική αυτή εφαρμόζεται μόνο στις μεθόδους των οποίων το σώμα μπορεί να αποτελεί τις κλήσεις προς την αντικατάσταση (εις) για τη μέθοδο. Αυτή είναι μια λογική πρακτική που κάνει τον κώδικα μικρότερο, πιο εύκολο να διαβαστεί. Το σύστημα είναι απλό και ελαφρύ για να εφαρμοστούν και να κατανοηθούν. Παρά την απλότητά του, είναι σε θέση να εκτελέσει όλες τις επανασχεδιάσεις κάποιας προηγούμενης εργασίας και τις πιο σημαντικές επανασχεδιάσεις άλλων των προηγούμενων εργασιών. Η μεθοδολογία αυτή επιτρέπει τον έλεγχο της συμβατότητας με νέες εκδόσεις του κώδικα πριν επανασχεδιαστεί, μέσω μιας εύκολης επιλογής μιας έκδοσης του κώδικα σε μια κλάση από μια βασική κλάση. Αυτή η τεχνική αποδεικνύει ότι ο επανασχεδιασμός σε απάντηση στις αλλαγές της βιβλιοθήκης του API μπορεί να επωφεληθεί εξίσου από τα υπάρχοντα αντικείμενα του κώδικα αντί να στηρίζεται σε νέα που απαιτούν πρόσθετες προσπάθειες για την κατασκευή και είναι ενδεχομένως ασυνεπή με τον κώδικα.

Στο (Xing και Stroulia, 2007) ο στόχος της επανασχεδίασης είναι η βελτίωση της ποιότητας του λογισμικού συστήματος, όπως η κατανόηση, η επεκτασιμότητα και η συντηρησιμότητα, χωρίς να επηρεάζεται η συνολική λειτουργικότητα και η συμπεριφορά του. Λόγω αυτού του ευεργετικού αντίκτυπου στο σχεδιασμό του λογισμικού, ορισμένα σύγχρονα ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDEs), όπως το Eclipse, παρέχουν ημιαυτόματη υποστήριξη για την εφαρμογή των πιο συχνά χρησιμοποιούμενων χαμηλού επιπέδου επανασχεδιάσεων. Στο πλαίσιο της επαναχρησιμοποίησης του λογισμικού, των εφαρμογών πελάτη και τα στοιχεία των πλαισίων που επαναχρησιμοποιούν είναι συνήθως πολύ πιο χαλαρά συνδεδεμένα. Η προσέγγιση δεν θέτει κανένα περιορισμό για τις αντιστοιχίσεις μεταξύ των αλλαγμένων API και των πιθανών αντικαταστάσεων τους. Η προσέγγιση DiffCatchUp επιτρέπει την απευθείας διαδραστική σύλληψη μέχρι την εξέλιξη του API ενός στοιχείου του πλαισίου κατά την προσαρμογή των εφαρμογών πελάτη που εξαρτώνται από αυτήν. Τα σημαντικότερα πλεονεκτήματα της προσέγγισης είναι ότι: (α) Δεν απαιτεί καμία πρόσθετη εργασία από τους προγραμματιστές του επαναχρησιμοποιήσιμου στοιχείου του πλαισίου. (β) Δεν επικεντρώνονται σε μεμονωμένες τοπικές αλλαγές, αλλά αντίθετα, έχει ως στόχο

να συγκεντρώσει όλα τα στοιχεία του σχεδιασμού που σχετίζονται με κάποιο συγκεκριμένο πρόβλημα. (γ) Όχι μόνο δεν θα διατυπώσει υποθέσεις για το πώς το αλλαγμένο API θα μπορούσε να αντικατασταθεί, αλλά συλλέγει επίσης συγκεκριμένα παραδείγματα των υποτιθέμενων αντικαταστάσεων που έχουν χρησιμοποιηθεί για την παροχή στους προγραμματιστές εφαρμογών με συναφείς πληροφορίες βάσει των οποίων να αξιολογήσουν τις προτάσεις. (δ) Αυτό συμπληρώνεται με ένα διαδραστικό πλαίσιο οπτικοποίησης μέσω του οποίου οι προγραμματιστές εφαρμογών μπορούν να ερευνήσουν τις προτεινόμενες εναλλακτικές λύσεις και το σχετιζόμενο παράδειγμα κώδικα.

Στο (Gharaibeh et al., 2007) το πλαίσιο προτείνει τη χρήση της καταγραφής αλλαγής που παρέχεται από τους προγραμματιστές API, καταγράφοντας τις αλλαγές στα API συμπεριλαμβανομένης των επανασχεδιάσεων. Η τεχνική αυτή γίνεται δημοφιλής, όπως χρησιμοποιείται στην επανασχεδίαση, εγγραφή και επανάληψη για την ενημέρωση του πηγαίου κώδικα. Κατά τη διάρκεια της διαδικασίας ανάπτυξης, οι προγραμματιστές μπορούν να κάνουν χρήση ορισμένων API κλάσεων μέσω της κληρονομικότητας. Σε αυτές τις περιπτώσεις, ο προγραμματιστής μπορεί να καλέσει τις μεθόδους που προβλέπονται ήδη από το API ή μπορεί να τις αντικαταστήσει. Η μετονομασία μιας κλάσης δεν επηρεάζει τη δομή κληρονομικότητας των πεδίων και των μεθόδων, καθώς διατηρούνται οι σχέσεις κληρονομικότητας. Η JVM έχει πρόσβαση σε κλάσεις χρησιμοποιώντας αναφορές στις οντότητες τους. Το κίνητρο για τη κατάσταση μετατροπής είναι ότι οι τροποποιήσεις του προγράμματος μπορεί να γίνουν αποτελεσματικά χωρίς να χρειάζεται να ενημερώσει ολόκληρη την κατάσταση του προγράμματος. Ένα άλλο πλεονέκτημα της κατάστασης μετατροπής είναι η ικανότητά της να λειτουργεί χωρίς να απαιτείται ειδικό υλικό ή λειτουργικό σύστημα. Έτσι, με βάση το VEE το απευθείας σύνδεση πλαίσιο ενημέρωσης είναι μια καλή λύση για κρίσιμες εφαρμογές για την προσωρινή αντιμετώπιση με αλλαγές στο API.

Τα άρθρα (Ellis et al., 2007, Matsuzawa και Ikeda, 1998, Ren et al., 2011) παρουσιάζουν μερικά από τα πλεονεκτήματα των αντικειμενοστρεφών μοντέλων. Στο (Ellis et al., 2007) το πρότυπο αφηρημένο εργοστάσιο παρέχει μια διεπαφή με την οποία ο πελάτης μπορεί να αποκτήσει παρουσίες των κλάσεων σύμφωνα με ένα συγκεκριμένο περιβάλλον ή το πρωτόκολλο, χωρίς να γνωρίζει ακριβώς ποια

τάξη αποκτά. Αυτό έχει το πλεονέκτημα της ενθυλάκωσης και την επαναχρησιμοποίηση κώδικα, δεδομένου ότι εφαρμογές μπορούν να τροποποιηθούν χωρίς να απαιτείται καμία αλλαγή στον κώδικα του πελάτη. Το πρότυπο "μέθοδος εργοστάσιου" είναι σχετικό, αλλά απλό, όπως το αφηρημένο πρότυπο εργοστάσιο, η μέθοδος του προτύπου εργοστάσιο επιτρέπει σε έναν πελάτη να λάβει αντικείμενα από μια άγνωστη κλάση που εφαρμόζει μια συγκεκριμένη διεπαφή. Όσον αφορά τα οφέλη, το πρότυπο εργοστάσιο επιβάλλει την αρχή της αντίστροφης εξάρτησης, οι εξαρτήσεις του πελάτη είναι μόνο για αφηρημένες κλάσεις και τις διασυνδέσεις και ποτέ για τις συγκεκριμένες υποκατηγορίες που τους μεταβιβάζονται. Αποσυνδέει το συγκεκριμένο πρότυπο εργοστάσιο και στιγμιότυπα από τα πάντα, αλλά και το σημείο της αρχικοποίησης τους. Αυτό σημαίνει, στην περίπτωση του αφηρημένου εργοστάσιου, που το εργοστάσιο μπορεί να ανταλλαχθεί μέσα και έξω απλά αλλάζοντας το πιο συγκεκριμένο εργοστάσιο και χωρίς να αγγίξει οποιονδήποτε άλλο κώδικα. Το πρότυπο εργοστάσιο διευκολύνει τη δημιουργία ποιοτικών αντικειμένων (δεδομένου ότι κατά πάσα πιθανότητα όλα αρχικοποιούνται χρησιμοποιώντας το ίδιο εργοστάσιο).

Στο (Matsuzawa και Ikeda, 1998) οι αντικειμενοστρεφείς τεχνολογίες είναι μια εφαρμόσιμη λύση για να φέρουν αξιόπιστο και ευέλικτο λογισμικό. Ο διαχειριστής της διασύνδεσης συνδυάζεται έμμεσα μεταξύ του ενδιάμεσου λογισμικού και μπορεί να χρησιμοποιήσει αποτελεσματικά το ενδιάμεσο λογισμικό χωρίς κακή παρεμβολή. Με αυτή τη βελτίωση, δεν υπάρχει κλάση που εξαρτάται τόσο από το περιβάλλον GUI και τις κλάσεις των βιβλιοθηκών OODB, είναι εύκολο να διαιρεθεί το υποσύστημα βασισμένο σε έναν ρόλο κάθε κλάσης. Ο ρόλος της κάθε κλάσης του διαχειριστή διασύνδεσης είναι σαφής γιατί ο ορισμός της κλάσης του διαχειριστή διασύνδεσης βασίζεται στη ροή των μηνυμάτων μεταξύ των κλάσεων. Το σύστημα χρησιμοποιώντας το μοντέλο MVIm είναι πολύ ευέλικτο για τροποποιήσεις των προδιαγραφών του συστήματος. Όταν το GUI ή το OODB σε ένα σύστημα πρέπει να ανταλλάσσονται, ή όταν ένα εξελιγμένο σύστημα πρέπει να συνδυαστεί με το υπάρχον GUI ή OODB, το μοντέλο αυτό μπορεί να μειώσει το κόστος της τροποποίησης στο ελάχιστο με την επαναχρησιμοποίηση της αφηρημένης κλάσης του διαχειριστή διασύνδεσης. Το μοντέλο MVIm είναι η έννοια

του μοντέλου αναφοράς για να κρατήσει την ανεξαρτησία μεταξύ των ενδιάμεσων λογισμικών και για να σημειώσει πρόοδο στην επαναχρησιμοποίηση.

Στο (Ren et al., 2011) ο τομέας της ανάπτυξης Web εφαρμογών, σύμφωνα με τις αρχές και τις μεθόδους της μηχανικής συστημάτων, η ολοκληρωμένη χρήση της μηχανικής λογισμικού, τα συστήματα βάσεων δεδομένων, τα δίκτυα υπολογιστών, η αντικειμενοστρεφή τεχνολογία και συνδυάζοντας με το πιο δημοφιλές, ανοικτού κώδικα πλαίσιο SSH (Spring, Struts, Hibernate), μελετά το λογισμικό της αυτόματης δημιουργίας πλατφόρμας που στηρίζεται στη B / S δομή πλαισίου της Java, για να βοηθήσει τους προγραμματιστές να κατασκευάσουν ένα σύστημα Web εφαρμογής με σαφή δομή, καλή επαναχρησιμοποίηση και εύκολη συντήρηση σε σύντομο χρονικό διάστημα. Ο κώδικας της αυτόματης ανάπτυξης λογισμικού ολοκληρώνεται μέσω ηλεκτρονικών υπολογιστών, οι προγραμματιστές δεν χρειάζεται να έχουν τον έλεγχο στην Java και η τεχνολογία της βάσης δεδομένων θα είναι σε θέση να αναπτύξει καλύτερα συστήματα λογισμικού. Με το SSH πλαίσιο, όχι μόνο επιτυγχάνεται ο πλήρης διαχωρισμός της προβολής, του ελεγκτή και του μοντέλου, αλλά και πραγματοποιεί τον διαχωρισμό του επίπεδου της επιχειρηματικής λογικής και του στρώματος διατήρησης των δεδομένων. Δεν έχει σημασία τι αλλάζει στο πρώτο, το στρώμα του μοντέλου με μικρές αλλαγές και οι αλλαγές στη βάση δεδομένων δεν θα επηρεάσουν τα προηγούμενα και βελτιώνει τη δυνατότητα επαναχρησιμοποίησης του συστήματος.

Τα άρθρα (Watson, 2009, Berglund, 2003, Rupakheti και Hou 2011, Wu et al., 2010, Hou και Yao, 2011, Dekel και Herbsleb, 2009) ασχολούνται με τα πλεονεκτήματα των θεμάτων τεκμηρίωσης των APIs. Στο άρθρο (Watson, 2009) η συνοπτική και η πλήρης τεκμηρίωση είναι τόσο σημαντικά όσο τα αυτονόητα μοντέλα αντικείμενου. Η διαδικασία της τεκμηρίωσης, η βελτίωση της συνεκτικότητας κάνει επίσης το API πιο εύχρηστο για τους προγραμματιστές που μιλούν άλλες γλώσσες. Τα API στοιχεία είναι ευκολότερο να αναλυθούν και να εξάγονται από ένα δομημένο αρχείο απ’ ότι από ένα μη δομημένο αρχείο. Σε μια στατική τυποποιημένη γλώσσα, ο τύπος δεδομένων μιας παραμέτρου μπορεί να είναι τόσο κατατοπιστικός όσο το όνομα, αν όχι περισσότερο. Αν υπάρχει σύγκρουση μεταξύ του ονόματος της παραμέτρου και του είδους των δεδομένων, ο τύπος δεδομένων είναι πιο αξιόπιστος, διότι το όνομα της παραμέτρου είναι

μόνο σημαντικό για τον προγραμματιστή, ενώ ο τύπος δεδομένων είναι σημαντικός τόσο για τον προγραμματιστή όσο και για τον μεταγλωττιστή. Αυτή η ισορροπία και η συνέπεια το καθιστά να είναι ευκολότερο για έναν προγραμματιστή για να ανακαλύψει και τις δύο μεθόδους όταν υπάρχουν και τα δύο και να μην χάνεται χρόνος ψάχνοντας για μια άλλη μέθοδο υποστήριξης της ιδιότητας μόνο μία από τις δύο λειτουργίες. Η ανάλυση παρέχει μια μοναδική άποψη του API που το καθιστά εύκολο να βρεθούν γρήγορα ασυνέπειες και αρκετά νωρίς στη διαδικασία σχεδιασμού για να διορθωθούν(αν μη σκόπιμα) ή να κατανοηθούν καλύτερα (αν σκόπιμα). Η ενιαία προέλευση αναφοράς θεμάτων με τον τρόπο αυτό ελαχιστοποιεί τη συνολική επιβάρυνση της τεκμηρίωσης και του φόρτου εργασίας. Έχοντας την τεκμηρίωση αναφοράς που είναι διαθέσιμη στη διαδικασία σχεδιασμού θα καταστήσει δυνατή τη διεξαγωγή δοκιμών ευχρηστίας νωρίτερα, ίσως πριν από κάθε κώδικα που έχει γραφτεί. Αυτό θα καθιστούσε δυνατό να βρει προβλήματα χρηστικότητας, ενώ υπάρχει ακόμη χρόνος για την αντιμετώπισή τους στο σχεδιασμό.

Στο (Berglund, 2003) η ηλεκτρονική τεκμηρίωση έχει τη δυνατότητα να προσαρμόσει το περιεχόμενο σε σχέση με τις μεταβλητές περιβάλλοντος. Η βιβλιοθήκη τεκμηρίωσης αναφοράς παρέχει τα εξής: (α) σύντομη περιγραφή των στοιχείων και των σχέσεων των στοιχείων, (β) δείκτες πλοήγησης (πληροφορίες που βοηθούν τους αναγνώστες να έχουν πρόσβαση στην τεκμηρίωση, συχνά σε αλφαβητική οργάνωση) και (γ) συντακτικά χαρακτηριστικά (που βοηθηθούν τους προγραμματιστές να γράφουν συντακτικά σωστό κώδικα). Η βιβλιοθήκη εγγράφων αναφοράς της DJavadoc προβλέπει τα εξής (α) επανασχεδιασμό του πληροφοριακού περιεχομένου, (β) προσωρινή χειραγώγηση του επανασχεδιασμένου περιεχομένου και (γ) εξέλιξη δείκτη σε μια μόδα σελιδοδείκτη για μενομωμένη περιήγηση και γρήγορη πρόσβαση. Με την προσθήκη των γνώσεων σχετικά με τον τομέα και σχετικά με τις εργασίες των προγραμματιστών στην ηλεκτρονική τεκμηρίωση, την ανάγνωση μπορεί να γίνει πιο αποτελεσματική. Ο επανασχεδιασμός χρησιμοποιείται για να μειώσει τη χρονοβόρα, επαναλαμβανόμενη αναζήτηση του εγχειριδίου για σαφώς καθορισμένους τύπους πληροφοριών. Η τεκμηρίωση ενσωματώνει ενεργές εργασίες πέραν της ανάγνωσης, στην περίπτωση αυτής της μετακίνησης κώδικα από την τεκμηρίωση για τα αρχεία προέλευσης. Αυτοματοποιημένα συστήματα τεκμηρίωσης, όπως η

Javadoc, έχουν δείξει ότι είναι δυνατόν να αυτοματοποιηθούν μεγάλα τμήματα της διαδικασίας τεκμηρίωσης για την αναφορά τεκμηρίωσης της βιβλιοθήκης, βοήθα να τελειοποιήσει την αυτοματοποίηση για να παράγει τον σωστό τύπο της ηλεκτρονικής τεκμηρίωσης.

Στα πλεονεκτήματα του άρθρου (Rupakheti και Hou 2011) αναφέρεται ότι (α) μπορούν να εξηγήσουν την επίδραση των APIs στον προγραμματιστή. Ενώ τα IDEs παρέχουν την τεκμηρίωση για μεμονωμένα APIs, δεν εξηγούν την επίδραση του συνδυασμού πολλαπλών APIs. (β) συνιστούν APIs και λύσεις που μπορεί να χρειαστούν στη συνέχεια, με βάση τη σχετικότητα τους με αυτά που χρησιμοποιούνται ήδη στον κώδικα. (γ) Επικρίνουν την ακατάλληλη χρήση των APIs, θέματα προειδοποίησης και μηνύματα λάθους. Η τεχνική βάση των εργαλείων θα είναι οι αλγόριθμοι ανάλυσης του προγράμματος. Αυτά τα εργαλεία είναι εφικτά, γενικά, και μπορεί να βελτιώσουν την παραγωγικότητα και την ποιότητα του λογισμικού, κάνοντας τα APIs ευκολότερα για να εργαστούν με λάθη. Στοχεύουν να κάνουν σχετικές τις πληροφορίες στο API να είναι εύκολα προσβάσιμες για τον προγραμματιστή αντιδρώντας ενεργά στις προθέσεις του, όπως αυτές εκφράζονται μέσα από τον κώδικα του. Με τον τρόπο αυτό, θα συμβάλει στη μείωση του χάσματος της πληροφόρησης μεταξύ των παρόχων της βιβλιοθήκης και των προγραμματιστών των εφαρμογών.

Στο (Wu et al., 2010) το CoDocent αυτόματα συνδέει σχετιζόμενα APIs από τη δομή του πλαισίου και τη σχέση μεταξύ των API εγγράφων. Το CoDocent παρέχει διαγράμματα ως αφαιρέσεις ώστε να αντικατοπτρίζει τη σημασιολογία των API κλήσεων που αναφέρεται στο παράδειγμα κώδικα. Τα διαγράμματα παρέχουν επίσης καθοδήγηση για τους προγραμματιστές να διερευνήσουν τα σχετικά API έγγραφα. Με το CoDocent, οι προγραμματιστές μπορούν να επιβεβαιώσουν τη σημασιολογία των κλήσεων API και να προσαρμόσουν τις κλήσεις του API στο νέο προγραμματισμό εργασιών. Τα παραδείγματα κώδικα είναι ένας ευρέως χρησιμοποιούμενος πόρος για να υποστηρίξει τη χρήση του API στην ανάπτυξη λογισμικού. Η βοήθεια χρήσης του API μπορεί να καθοδηγήσει τους προγραμματιστές να διερευνήσουν τα εν λόγω έγγραφα API, επιβεβαιώνοντας για τη σημασιολογία των κλήσεων API. Με την βοήθεια χρήσης του API, οι

προγραμματιστές μπορούν να βρουν τον καλύτερο τρόπο να προσαρμόσουν τις κλήσεις API στο πλαίσιο του προγραμματισμού τους.

Στο (Hou και Yao, 2011) η επαναχρησιμοποίηση έχει βελτιώσει σημαντικά την παραγωγικότητα και την ποιότητα του λογισμικού. Μια εφαρμογή αλληλεπιδρά με ένα σύστημα επαναχρησιμοποίησης μέσω των διασυνδέσεων προγραμματισμού εφαρμογών (API). Τα πλαίσια λογισμικού και οι βιβλιοθήκες παρέχουν δύναμη σε μεγάλο βαθμό, επειδή χρησιμοποιούνται από πολλές εφαρμογές μέσω του API. Ένα μεγάλο ποσοστό των API μεθόδων ξεπεράστηκαν, προκειμένου να συμμορφώνονται με μια σειρά από συμβάσεις ονομασίας. Πιο συγκεκριμένα ονόματα. Η ακριβής μέθοδος ονομασίας θα βοηθήσει να μεταφερθεί η ακριβή έννοια του όρου και η συμπεριφορά ότι η μέθοδος αυτή αποσκοπεί στην εφαρμογή. Πιο συνοπτικά ονόματα. Μια πιο σύντομη ονομασία θα βοηθήσει να μεταφέρει την ίδια σημασιολογία, αλλά με λιγότερες λέξεις. Απλούστευση, μερικές μέθοδοι API έχουν καταργηθεί για να μειώσουν τον όγκο του API. Αυτό το είδος της απλούστευσης θα μπορούσε να βελτιώσει τη χρηστικότητα του API. Εισαγωγή των νέων εννοιών και των κλάσεων. Οι υπάρχουσες μέθοδοι API μπορούν να καταργηθούν και να αντικατασταθούν, λόγω της εισαγωγής των νέων αντιλήψεων ή κλάσεων. Οι μέθοδοι API αντικαθίστανται ή προστίθενται για να βελτιώσουν τη χρηστικότητα και την απόδοση (λιγότερες μέθοδοι API για να μάθουν, ή μικρότεροι κώδικες πελατών). Για τους προγραμματιστές εφαρμογών, γνωρίζοντας αυτές τις κλάσεις θα τους επιτρέψουν να κατανοήσουν αλλαγές στο API πιο αποτελεσματικά.

Στο (Dekel και Herbsleb, 2009) οι κλήσεις της μεθόδου είναι ένας αποτελεσματικός τρόπος για να προειδοποιήσει τους αναγνώστες για ενδεχόμενες σημαντικές πληροφορίες που σχετίζονται με αυτούς τους στόχους και χωρίς σημαντική υπερφόρτωση. Ένα αντικείμενο γνώσης (KI - Knowledge Items) είναι ένα ατομικό και συνοπτικό στοιχείο που πρόκειται να συλληφθεί γρήγορα και αποτελεσματικά από άποψη κόστους ως μία μόνο φράση ή γραμμή κειμένου μεταφέροντας μια ιδέα. Δεδομένου ότι ο κώδικας πρέπει να είναι διαθέσιμος και τροποποιήσιμος, τα KIs είναι οι πιο κατάλληλα για αντικείμενα του έργου και τις βιβλιοθήκες. Ο μηχανισμός αυτός έχει ως στόχο να επιτρέψει μία κοινότητα χρηστών API για να προσθέσουν και να βελτιώσουν τα σχόλια στη πάροδο του

χρόνου, ακόμη και χωρίς υποστήριξη από τον προμηθευτή ή την πρόσβαση στον πηγαίο κώδικα. Τα KIs δεν απεικονίζουν το κείμενο τεκμηρίωσης, μπορούν να χρησιμεύσουν ως μια αρχική ή ελαφριά εναλλακτική λύση για την τεκμηρίωση. Αυξάνοντας το JavaDoc ήταν ιδιαίτερα αποτελεσματικό και βοήθησε τους συμμετέχοντες να βρουν οδηγίες που απείχαν πολύ από κάποιους ελέγχους που διάβαζαν μόνο στο κείμενο. Αυτό σημαίνει ότι οποιαδήποτε τεχνική βοηθά να προσελκύσει την προσοχή των οδηγιών μπορεί να είναι χρήσιμη, ακόμα κι αν είναι περιορισμένη για τη διάρθρωση και τη μορφοποίηση του κειμένου.

Τα δυο άρθρα (Robillard, 2009, Hou και Li, 2011) διερευνούν τις δυσκολίες εκμάθησης των διεπαφών προγραμματισμού εφαρμογών (APIs), θα μελετηθούν αναλυτικότερα στην ενότητα 3.2.3 που αναφέρεται στα μειονεκτήματα του τομέα ανάλυσης των APIs.

Στο άρθρο (Kernebeck, 1997) το στοιχείο της διαχείρισης του χώρου αποθήκευσης έχει στόχο την μείωση του κόστους για συστηματική επαναχρησιμοποίηση του λογισμικού. Για παράδειγμα, η οργάνωση των στοιχείων λογισμικού σε ένα χώρο αποθήκευσης θα μειώσει τις προσπάθειες αναζήτησης για ένα στοιχείο λογισμικού και αυξάνει την πιθανότητα ανεύρεσης ενός κατάλληλου στοιχείου στη βιβλιοθήκη. Το σημαντικότερο μέτρο είναι να κάνει εξοικειωμένο το προσωπικό ανάπτυξης λογισμικού με τις δυνατότητες της επαναχρησιμοποίησης λογισμικού. Ως εκ τούτου, η διαδικασία ανάπτυξης λογισμικού θα ενσωματώσει τις δραστηριότητες επαναχρησιμοποίησης, έτσι ώστε να αναζητήσει το στοιχείο στη βιβλιοθήκη του λογισμικού που γίνεται κανονικά, όπως αρχίζει ένα έργο λογισμικού με μια ανάλυση απαιτήσεων. Η βιβλιοθήκη στοιχείου του λογισμικού είναι δυναμική, υπό την έννοια ότι η χρήση ενός στοιχείου μετριέται. Στοιχεία που δεν είναι ή σπάνια χρησιμοποιούνται θα μπορούσαν να διαγραφούν. Τα στοιχεία του λογισμικού μπορούν να αποθηκευτούν σε μια βάση δεδομένων, έτσι ώστε η διαδικασία αναζήτησης μπορεί να αυτοματοποιηθεί. Το λογισμικό διασύνδεσης από μόνο του είναι πλήρης, έτσι ώστε μόνο μερικές σχετικές απαιτήσεις και πρότυπα πηγαίου κώδικα μπορούν να αποθηκευτούν.

Σε αυτό το άρθρο (Gerken et al., 2011) παρουσιάζονται τα πλεονεκτήματα της μεθόδου του εννοιολογικού χάρτη για να μελετήσουμε τη χρηστικότητα ενός

API στη πάροδο του χρόνου. Ένα API είναι διαφορετικό από μια γραφική διεπαφή χρήστη, για το οποίο αυτές οι μέθοδοι έχουν σχεδιαστεί για να πιστεύουμε ότι υπάρχει ένα τεράστιο δυναμικό για τις μεθόδους αξιολόγησης που έχουν σχεδιαστεί ειδικά για να αντιμετωπίσουν τις ιδιαιτερότητες του API. Το κύριο πλεονέκτημα είναι ότι δεν χρειάζεται οι πραγματικοί χρήστες που μπορεί να διευκολύνουν τον έλεγχο ως ομάδα-στόχο του API, συχνά να εξαπλώνονται σε όλο τον κόσμο. Η μέθοδος αποδείχθηκε ότι είναι εξαιρετικά επεκτάσιμη και παρέχει μια πολύ καλή σχέση οφέλους-κόστους για την αναλογία. Ο πίνακας επιτρέπει στους περισσότερους ανθρώπους την τοποθέτηση των ίδιων γύρω από τον χάρτη, ο κάθετος πίνακας έχει το πλεονέκτημα ότι επιτρέπει στο χρήστη να εντείνει και για να αποκτήσει μια γενική εικόνα, η οποία θεωρείται ως ένα σημαντικό πλεονέκτημα της εν λόγω ρύθμισης. Επεκτείνοντας και τροποποιώντας τους χάρτες στη πάροδο του χρόνου η μεθοδολογία μπορεί να είναι πιο αποτελεσματική, αν υπάρχει χρόνος, για τέτοια διαχρονικά δεδομένα συλλογής σχεδιασμού. Ένα κύριο πλεονέκτημα της μεθόδου εννοιολογικών χαρτών που συγκρίθηκε με υπάρχουσες προσεγγίσεις είναι η δυνατότητα να συλλάβει τη δυναμική των χρήσεων, το οποίο επίσης αναφέρεται στην εκμάθηση του API και βοηθά στην αποφυγή ‘ψεύτικων θετικών αποτελεσμάτων’.

3.1.4 Μελέτη των πλεονεκτημάτων του τομέα γενικά ζητήματα των APIs

Στη τέταρτη κατηγορία ανήκουν τα 11 άρθρα γενικού περιεχομένου όσον αφορά τις Διασυνδέσεις Προγραμματισμού Εφαρμογών που μελετήσαμε στην ενότητα 2.3.1.4. Τα άρθρα (WENDYKIER και NAGY, 2010, Koehler et al., 2008, Dezso et al., 2011, Johnsson και Mathur, 2005) αναφέρονται στα πλεονεκτήματα των πολυδιάστατων πινάκων και των επιστημονικών υπολογισμών, στα παράλληλα εργαλεία ανάλυσης των αποδόσεων, καθώς επίσης στους παράλληλους επεξεργαστές, στη χρήση γραφημάτων και αλγόριθμων δικτύου.

Στο (WENDYKIER και NAGY, 2010) το πλεονέκτημα είναι η αξιοποίηση των σύγχρονων αρχιτεκτονικών υπολογιστών στον τομέα του επιστημονικού προγραμματισμού με πολυνηματικό προγραμματισμό σε Java για εφαρμογές επεξεργασίας εικόνας. Στόχος είναι να παρέχεται λογισμικό που να είναι αποτελεσματικό, ευέλικτο και εύκολο στη χρήση στους επιτραπέζιους και φορητούς υπολογιστές. Παρόλο που η Java δεν σχεδιάστηκε για να είναι μια

επιστημονική γλώσσα προγραμματισμού, έχει πολλά μοναδικά χαρακτηριστικά τα οποία είναι ελκυστικά για υψηλής απόδοσης επιστημονικό προγραμματισμό. Η Java είναι μια εξαιρετικά φορητή γλώσσα προγραμματισμού. Η Java έχει γίνει ένα έργο ανοικτού πηγαίου κώδικα, επιτρέπει σε οποιονδήποτε να την τροποποιήσει και να την προσαρμόσει στις ανάγκες του. Η Java παρέχει υποστήριξη για πολυνημάτωση. Το Παράλληλο Colt παρέχει αποτελεσματικότητα και χρησιμοποιήσιμες δομές δεδομένων και αλγορίθμους για την ανάλυση δεδομένων, τη γραμμική άλγεβρα, πολυδιάστατους πίνακες, στατιστικά στοιχεία, ιστογράμματα και ταυτόχρονο προγραμματισμό. Το Colt διαθέτει υποστήριξη για ενιαίους, ευέλικτους και αποτελεσματικούς πολυδιάστατους πίνακες. Η μονή ακρίβεια έχει δύο πλεονεκτήματα σε σχέση με τη διπλή ακρίβεια, οι αριθμητικές πράξεις είναι γρηγορότερες με μονούς αριθμούς ακρίβειας και απαιτούν μόνο κατά το ήμισυ την αποθήκευση των αριθμών διπλής ακρίβειας.

Στο (Koehler et al., 2008) οι εφαρμογές RC έχουν τη δυνατότητα να επιτύχουν κύκλους μεγέθους απόδοσης, ενώ χρησιμοποιείται λιγότερη ενέργεια και τους πόρους του υλικού από τις συμβατικές εφαρμογές λογισμικού. Οι βασικοί στόχοι των εργαλείων ανάλυσης της απόδοσης είναι οι εξής: (α) Η συμπεριφορά της αρχικής εφαρμογής διαταράσσεται όσο το δυνατόν λιγότερο (ελαχιστοποίηση των επιπτώσεων), (β) Καταγραφή επαρκούς λεπτομέρειας και της δομής για να ανακατασκευάσει με ακρίβεια τη συμπεριφορά των εφαρμογών, (γ) Επιτρέπει την ευελιξία για να παρακολουθεί διάφορες εφαρμογές και συστήματα (μεγιστοποίηση προσαρμοστικότητας και φορητότητας), (δ) Απαιτείται λίγη προσπάθεια από το σχεδιαστή ως πιθανή (ελαχιστοποίηση δυσχέρειας), (ε) Εμφάνιση μόνο ό,τι είναι απαραίτητο για να συλλάβει τη συμπεριφορά της εφαρμογής και των σημείων συμφόρησης (να είναι συνοπτική), (στ) Η μορφή των δεδομένων για να καταστεί δυνατή η ταχεία κατανόηση της συμπεριφοράς της εφαρμογής (είναι διαισθητική). Οι μετρητές κατανομής και οι απομονωτές του χώρου αποθήκευσης μπορούν να λειτουργήσουν ανεξάρτητα από την εφαρμογή από κάθε άλλη στο υλικό. Η ανάλυση των επιδόσεων του υλικού που παράγεται από μια υψηλού επιπέδου γλώσσα είναι εξαιρετικά σημαντική, δεδομένου ότι ένα άλλο στρώμα της αφαίρεσης προστίθεται, περαιτέρω αλλάζοντας την πορεία συμπεριφοράς της εφαρμογής.

Στο άρθρο (Dezso et al., 2011) ο στόχος της βιβλιοθήκης είναι να παρέχει υψηλή απόδοση, ευκολία στη χρήση και καλή συνεργασία των στοιχείων του λογισμικού, το οποίο βοηθά στην επίλυση πολύπλοκων προβλημάτων βελτιστοποίησης. Αυτά τα στοιχεία περιλαμβάνουν εφαρμογές γραφικών παραστάσεων και των σχετικών δομών δεδομένων, τους θεμελιώδεις αλγόριθμους του γραφήματος και διάφορα βοηθητικά εργαλεία. Το LEMON ενσωματώνεται καλά με διάφορα ολοκληρωμένα περιβάλλοντα ανάπτυξης. Το κύριο πλεονέκτημα αυτού του σχεδιασμού είναι ότι όλοι οι αλγόριθμοι του κατευθυνόμενου γραφήματος αυτόματα εργάζονται για μη κατευθυνόμενα γραφήματα. Το LEMON διαθέτει μόνο εξωτερική ιδιότητα χαρτών που αποθηκεύονται ξεχωριστά από το σχετικό γράφημα δομής δεδομένων, αλλά θα ενημερώνονται αυτόματα για τις αλλαγές του γραφήματος. Το κύριο πλεονέκτημα των εξωτερικών χαρτών είναι η μεγάλη ευελιξία τους. Μπορούν να κατασκευαστούν και να καταστραφούν ελεύθερα, έτσι η διάρκεια ζωής τους δεν δεσμεύεται για τη διάρκεια ζωής του γραφήματος. Το κύριο πλεονέκτημα της εσωτερικής αποθήκευσης είναι ότι η χωρητικότητα του προσαρμόζεται αυτόματα αν ο γράφος είναι τροποποιημένος. Μια συνάρτηση διεπαφής έχει το σημαντικό πλεονέκτημα ότι τα προσωρινά αντικείμενα μπορούν να περαστούν ως παράμετροι αναφοράς. Το πλεονέκτημα αυτού του σχεδιασμού είναι διπλό. Το LEMON εφαρμόζει μια αντικειμενοστρεφή προσέγγιση, η οποία είναι αρκετά παρόμοια με την τεχνολογία ILOG Concert. Η προσέγγιση αυτή καθιστά τη διεπαφή του LEMON πιο ευέλικτη από τις διεπαφές αρκετών βιβλιοθηκών LP και μπορεί να είναι πιο άνετη για όσους είναι εξοικειωμένοι με αντικειμενοστρεφή προγραμματισμό.

Στο (Johnsson και Mathur, 2005) η φορητότητα είναι επίσης ζωτικής σημασίας για την ταχεία προσαρμογή της νέας τεχνολογίας, επιτρέποντας έτσι την έγκαιρη αποκόμιση οφελών από την αύξηση των μεγεθών της μνήμης, αύξηση της απόδοσης, ή μείωση του κόστους / απόδοση που προσφέρουν οι νέες τεχνολογίες. Ο πρωταρχικός στόχος του σχεδιασμού για τη Μηχανή Σύνδεσης με την Επιστημονική Βιβλιοθήκη λογισμικού, CMSSL είναι η παροχή υψηλού επιπέδου υποστήριξης για τις περισσότερες αριθμητικές μεθόδους, τόσο των παραδοσιακών και των πρόσφατα ανεπτυγμένων μεθόδων, όπως η ιεραρχική και πολλαπλών διαστάσεων μέθοδοι και άλλους ταχέως ονομαζόμενους N-αλγόριθμους που χρησιμοποιούνται για μεγάλης κλίμακας επιστημονικών

υπολογισμών μηχανικής. Υψηλού επιπέδου υποστήριξη στο πλαίσιο αυτό σημαίνει λειτουργικότητα η οποία βρίσκεται σε αρκετά υψηλό επίπεδο ώστε αρχιτεκτονικά χαρακτηριστικά είναι ουσιαστικά διαφανή για τον χρήστη και η υψηλή απόδοση μπορεί να επιτευχθεί. Το CMSSL είναι μια "παγκόσμια" βιβλιοθήκη. Για την παγκόσμια βιβλιοθήκη, οι λειτουργίες καλούνται άμεσα και είναι διαφανής για τον χρήστη και για την κατανεμημένη φύση των δομών δεδομένων που είναι διαφανής στο χρήστη. Το CMSSL παρέχει ανεξάρτητη λειτουργικότητα διανομής στοιχείων και έχει τη λογική για την αυτόματη επιλογή αλγόριθμου. Η επεκτασιμότητα είναι εξαιρετική. Η ευρωστία σε σχέση με την απόδοση επιτυγχάνεται με την αυτόματη επιλογή του αλγορίθμου ως συνάρτηση της κατανομής των δεδομένων τόσο για χαμηλό επίπεδο και το υψηλό επίπεδο λειτουργιών.

Τέσσερα άρθρα αυτής της κατηγορίας (Ferreira et al., 2007, Unalir et al., 2000, Inoue et al., 2010, Sabou, 2004) ασχολούνται με τα πλεονεκτήματα που αφορούν το κατανεμημένο περιβάλλον, τους πόρους, το HTTP πρωτόκολλο, θέματα προσομοίωσης, την αρχιτεκτονική client – server καθώς και με διαδικτυακές υπηρεσίες οι οποίες περιγράφονται από μια οντολογία.

Στο άρθρο (Ferreira et al., 2007) η αυξημένη πραγματικότητα επεκτείνει την πραγματικότητα με εικονικά στοιχεία, αλλά προσπαθεί να τοποθετήσει τον υπολογιστή σε ένα σχετικά διακριτικό και βοηθητικό ρόλο. Η τεχνολογία που απαιτείται για τους ορατούς υπολογιστές θα έρθει σε τρία μέρη: φθηνοί, χαμηλής ισχύος υπολογιστές, συμπεριλαμβανομένων εξίσου βολικές οθόνες, το λογισμικό για τις εφαρμογές και τα δίκτυα που τα δένουν όλα μαζί. Τα οφέλη από την αξιοποίηση του ενδιάμεσου λογισμικού είναι το βελτιωμένο μοντέλο προγραμματισμού και το κρύψιμο πολλών λεπτομερειών της εφαρμογής, τα οποία καθιστούν το ενδιάμεσο λογισμικό με βάση την ανάπτυξη των εφαρμογών πολύ πιο γρήγορα. Η κύρια συνεισφορά του άρθρου είναι ο ορισμός της αρχιτεκτονικής διαχείρισης ενός QoS (Quality of Service) τόσο για τον πελάτη (UE) όσο και στον κατανεμημένο διακομιστή παιχνιδιών. Το API είναι επαρκές για την πλατφόρμα για τα 3GPP δίκτυα σύμφωνα με τις τελευταίες προδιαγραφές του 3GPP End-to-end QoS.

Στο (Unalir et al., 2000) το FedeRaL έχει δύο κύριες εισφορές για τις μελέτες επαναχρησιμοποίησης της βιβλιοθήκης. Το αντικείμενο αυτού του εργαλείου είναι να δημιουργηθεί μια κλιμακούμενη και συνολικά επαναχρησιμοποιούμενες βιβλιοθήκες μέσω του διαδικτύου. Με την "κλιμακούμενη", εννοούμε μια βιβλιοθήκη επαναχρησιμοποίησης που κατασκευάζεται από το μηδέν και στη συνέχεια επεκτείνεται με το πλαίσιο επέκτασης της βιβλιοθήκης. Η XML χρησιμοποιείται στο FedeRaL από αυτές τις απόψεις: (α) Να περιγράφει τα μεταδεδομένα σχετικά με τα επαναχρησιμοποιήσιμα κατασκευαστικά στοιχεία. (β) Για να δημοσιεύει και να ανταλλάσσει περιεχόμενα του χώρου αποθήκευσης της βάσης δεδομένων. (δ) Ως μηνύματα μορφής για επικοινωνία μεταξύ των συλλογικών βιβλιοθηκών επαναχρησιμοποίησης.

Η XML παρέχει διαλειτουργικότητα με ευέλικτες, ανοικτές, που βασίζονται σε πρότυπα μορφές, διαχωρίζει τα δεδομένα από την παρουσίαση και τη διαδικασία. Το FedeRaL ενσωματώνει διαφορετικούς τύπους παροχής πληροφοριών σε διαφορετικά σημεία του συστήματος. Το FedeRaL προσφέρει μια διαλειτουργική αρχιτεκτονική επαναχρησιμοποίησης της βιβλιοθήκης με χρήση ανοικτού προτύπου πληροφοριών που επιτρέπει στους διαχειριστές την επαναχρησιμοποίηση για την κατασκευή των μοντέλων πληροφοριών στην κορυφή της επαναχρησιμοποίησης. Είναι διαλειτουργικό επειδή οι διαχειριστικές υπηρεσίες χρηστών επιτρέπουν στους χρήστες να περιηγηθούν και να αναζητήσουν διαφορετικές βιβλιοθήκες επαναχρησιμοποίησης οι οποίες είναι χτισμένες στην κορυφή του BIDM.

Στο άρθρο (Inoue et al., 2010) έχει αναπτυχθεί μια απόδειξη της έννοιας εφαρμογής με WAPDB και διαπιστώθηκε ότι προσφέρει μεγάλη αποτελεσματικότητα κόστους χωρίς σημαντικές επιπτώσεις στις επιδόσεις. Το κόστος ανάπτυξης μειώνεται σε λιγότερο από το ήμισυ με την επιβάρυνση (σε χρήση), μόλις λίγα msec σε χρόνους απόκρισης. Το WAPDB έχει τα απαραίτητα χαρακτηριστικά γνωρίσματα για την ανάπτυξη της εφαρμογής Ιστού, συμπεριλαμβανομένων ενός αποτελεσματικού προτύπου για έλεγχο πρόσβασης, έναν εύκολο μηχανισμό επέκτασης και αναζήτησης και τις δυνατότητες στατιστικών στοιχείων. Η επαναχρησιμοποίηση, η βελτίωση αυτής της ιδιότητας μειώνει το κόστος ανάπτυξης. Το WAPDB παρέχει έναν εύκολο μηχανισμό

επέκτασης λειτουργίας. Η επέκταση είναι δυνατή χωρίς τον πηγαίο κώδικα. Επιπλέον, δεν υπάρχουν περιορισμοί που πρέπει να τοποθετηθούν σχετικά με τις γλώσσες προγραμματισμού που χρησιμοποιούνται για την ανάπτυξη των επεκτάσεων. Η αναζήτηση και τα στατιστικά στοιχεία, είναι ουσιώδεις λειτουργίες στις περισσότερες εφαρμογές στο Web. Είναι ενσωματωμένο δεδομένου ότι απαιτεί επιπλέον ευρετήρια και πίνακες, ακόμη και αν το WAPDB θα πρέπει να έχει ένα μηχανισμό επέκτασης, είναι προτιμότερο ότι είναι απλό και κατάλληλο σε πόρους του Παγκόσμιου Ιστού.

Στο (Sabou, 2004) η οντολογία τομέα προσφέρει όλες τις απαραίτητες έννοιες για να περιγράψει τη λειτουργικότητα της υπηρεσίας web και αναφέρει επίσης πως οι έννοιες που έχουν χρησιμοποιηθεί ταιριάζουν σε ένα ευρύτερο φάσμα των πιθανών λειτουργιών. Η βασική διαδικασία εξαγωγής αποδείχθηκε πολύ χρήσιμη, εξήχθη το ήμισυ των εννοιών στη μη αυτόματη δημιουργία οντολογίας και πρότειναν επίσης μερικές νέες προσθήκες που αγνοήθηκαν κατά τη διάρκεια της δημιουργίας της οντολογίας. Ελέγχθηκε η εφαρμογή της μεθόδου εξόρυξης σε διαφορετικό κώδικα από εκείνον τον οποίο χρησίμευσε για το σχεδιασμό και έλαβε έννοιες από όλες τις μεγάλες κατηγορίες των λειτουργιών. Έχει αποδειχθεί ότι είναι δυνατό και χρήσιμο για την κατασκευή οντολογιών από την τεκμηρίωση του λογισμικού, μη αυτόματη δημιουργία μιας τέτοιας οντολογίας από την τεκμηρίωση API και χρησιμοποιώντας τη για να περιγράψει διάφορες διαδικτυακές υπηρεσίες. Διερευνήθηκε η δυνατότητα για την αυτοματοποίηση της διαδικασίας εξαγωγής της οντολογίας και παρουσιάστηκε μια μέθοδος για την εξαγωγή των εννοιών που δηλώνουν τη σημασιολογία της λειτουργικότητας. Ακόμα κι αν οι έννοιες αυτές αποτελούν μόνο μέρος των διαθέσιμων πληροφοριών στο API, η εξαγωγή τους είναι μια σημαντική βοήθεια για τον μηχανικό της οντολογίας.

Αυτό το άρθρο (Pham και Brown, 2005) παρουσιάζει τα πλεονεκτήματα των ασαφών συστημάτων απεικόνισης. Τα ασαφή συστήματα συχνά αποδίδουν καλύτερα από ό,τι τα παραδοσιακά συστήματα, λόγω της ικανότητάς τους να αντιμετωπίζονται με μη-γραμμικότητα και αβεβαιότητα. Ο κύριος λόγος είναι ότι, ενώ τα παραδοσιακά συστήματα κάνουν ακριβείς αποφάσεις σε κάθε στάδιο, τα ασαφή συστήματα διατηρούν τις πληροφορίες σχετικά με την αβεβαιότητα όσο το

δυνατόν περισσότερο και βασίζονται μόνο σε μια καθαρή απόφαση στο τελευταίο στάδιο. Ένα άλλο πλεονέκτημα είναι ότι οι γλωσσικοί κανόνες, όταν χρησιμοποιούνται σε ασαφή συστήματα, όχι μόνο να καταστήσουν τα εργαλεία πιο διαισθητικά, αλλά και να παρέχουν καλύτερη κατανόηση και εκτίμηση των αποτελεσμάτων. Έτσι, η οπτικοποίηση είναι συνυφασμένη με τις εργασίες που διεξάγονται προκειμένου να παρέχουν περισσότερες γνώσεις για τους χρήστες και να βελτιώσουν τη διαδικασία λήψης αποφάσεων. Η πολυδιάστατη κλιμάκωση και οι παράλληλες συντεταγμένες παρέχουν τρόπους για να εμφανιστούν πολυδιάστατα ασαφή δεδομένα σε 2D χωρίς να χαθεί καμία πληροφορία. Χρησιμοποιήθηκαν ασαφή γνωστικά δίκτυα με τη μορφή ενός γραφήματος για να κωδικοποιήσουν τις σχέσεις σε ένα σύνθετο σύστημα αλληλεπίδρασης. Αυτή η τεχνική είναι χρήσιμη για κωδικοποίηση ειδικών πληροφοριών οι οποίες συχνά παρουσιάζονται σε ασαφή συστήματα ελέγχου.

Αυτό το άρθρο (Xie και Pei, 2006) αναφέρει τους στόχους που επιτυγχάνει το εργαλείο MAPO. Το εργαλείο είναι σε θέση να εξάγει πληροφορίες χρήσης του API από ένα αρχείο προέλευσης που μπορεί να μην είναι σε θέση να μεταγλωττιστεί από έναν μεταγλωττιστή, επειδή η μηχανή αναζήτησης πηγαίου κώδικα δεν μπορεί να επιστρέψει όλα τα άλλα αρχεία προέλευσης, που εξαρτάται το αρχείο προέλευσης. Το εργαλείο είναι σε θέση να συμπεράνει συχνές χρήσεις του API που περιλαμβάνουν πληροφορίες συχνότητας μεταξύ των κλήσεων της μεθόδου. Η διαδοχή των πληροφοριών είναι ένα σημαντικό μέρος χρήσεων του API. Το εργαλείο είναι σε θέση να εξάγει συχνές χρήσεις API που περιλαμβάνουν κλήσεις μεθόδου από περισσότερες από μία κλάσεις, γιατί οι ρεαλιστικές χρήσεις του API συχνά περιλαμβάνουν μεθόδους από πολλαπλές κλάσεις. Το εργαλείο είναι σε θέση να παράγει μια μικρή λίστα των σχετικά συχνών προτύπων χρήσης του API για επιθεώρηση. Το εργαλείο MAPO εξάγει πιο πολύπλοκα πρότυπα χρήσης του API από τα τμήματα κώδικα που επιστρέφονται από μια μηχανή αναζήτησης κώδικα.

Το τελευταίο άρθρο αυτής της κατηγορίας ερευνά τα οφέλη ενός υπολογιστή που έχει API μηχανισμό σύνδεσης σε σύγκριση με έναν υπολογιστή χωρίς μηχανισμό προστασίας, προκείμένου να ανιχνευθεί κακόβουλο λογισμικό. Το άρθρο (Marhusin, et al., 2008) αναφέρει τη δυναμικά συνδεδεμένη βιβλιοθήκη

(DLL) ως μέσο για τα συστήματα παρακολούθησης κλήσεων. Μία από τις τεχνικές που εφαρμόζονται η οποία αντικατέστησε μια πραγματική DLL του Internet Explorer με μια ψεύτικη. Η ψεύτικη DLL αναμεταδίδει μηνύματα που αποστέλλονται από και προς την αρχική DLL. Κάνοντάς το με αυτόν τον τρόπο, μπορεί να αναλύσει και να καταγράψει το πρότυπο μηνύματος των ασφαλών κλήσεων του συστήματος. Χρησιμοποιείται ένα πλαίσιο που διευκόλυνε τις προσπάθειες σύνδεσης. Το πλεονέκτημα της χρήσης του πλαισίου ήταν ότι θα μπορούσε να αναπτύξει γρήγορα τη σύζευξη του προγράμματος. Η όλη διαδικασία επαναλαμβάνεται δύο ακόμη φορές, προκειμένου να εκτιμήσει τη συνέπεια των αποτελεσμάτων. Αποδείχθηκε ότι τα αποτελέσματα από τρεις επαναλήψεις ήταν σε γενικές γραμμές συνεπή. Αυτό είναι σημαντικό, διότι αν έπρεπε να γίνουν περισσότερες επαναλήψεις θα ήταν πολύ χρονοβόρο. Τα αποτελέσματα που προέκυψαν από τα πειράματα δείχνουν ότι τα δεδομένα της API σύνδεσης είναι η κατάλληλη επιλογή για να ψάξουν για την ανίχνευση κακόβουλου λογισμικού, χωρίς να ανησυχούν σχετικά με την επίδοση.

3.2 ΜΕΙΟΝΕΚΤΗΜΑΤΑ ΤΩΝ ΔΙΑΣΥΝΔΕΣΕΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΕΦΑΡΜΟΓΩΝ (APIs)

Στην ενότητα αυτή μελετάμε τα μειονεκτήματα των διασυνδέσεων προγραμματισμού εφαρμογών (APIs), σύμφωνα με τους τέσσερις ερευνητικούς τομείς όπως προέκυψαν στην ενότητα 2.3.1.

3.2.1 Μελέτη των μειονεκτημάτων του τομέα εφαρμογής των APIs

Στην πρώτη κατηγορία αναλύουμε τα μειονεκτήματα των 7 μελετών που αφορούν τον τομέα εφαρμογής των APIs. Πιο συγκεκριμένα, τα άρθρα (Pai et al., 2003, Singh et al., 2005, Sudarsan και Ribbens, 2010, Carvalho και Cachopo, 2011, Gotlieb και Bernard, 2006) περιγράφουν τα μειονεκτήματα σχετικά με τον σχεδιασμό, την εφαρμογή και την επίδοση των διασυνδέσεων προγραμματισμού εφαρμογών.

Στο άρθρο (Pai et al., 2003) ένας σχεδιαστής μιας κρυφής μνήμης μεσολάβησης δεν μπορεί να προβλέψει όλες τις πιθανές χρήσεις για τη κρυφή μνήμη μεσολάβησης και ως εκ τούτου δεν μπορεί να περιλαμβάνει όλα τα χαρακτηριστικά που απαιτούνται από την εκτέλεση της εφαρμογής. Η ευελιξία του μοντέλου διαδικασίας συνεπάγεται μεγαλύτερη επιβάρυνση στο λειτουργικό

σύστημα, συμπεριλαμβανομένου της επιπλέον μνήμης για την αποθήκευση της κατάστασης της διαδικασίας και μεγαλύτερη επιβάρυνση στη CPU, όταν το λειτουργικό σύστημα παρέχει ενδοεπικοινωνιακούς μηχανισμούς για την ανταλλαγή πληροφοριών μεταξύ του πυρήνα μεσολάβησης και της μονάδας. Το ICAP επιτρέπει στους διακομιστές να ενημερώνουν στατικά τους διακομιστές μεσολάβησης ότι τα HTTP δεδομένα για ορισμένες επεκτάσεις αρχείων δεν πρέπει να περάσουν σε αυτούς. Οι αλλαγές σε ένα API με δυνατότητα διακομιστή μεσολάβησης επηρεάζουν τις μονάδες μόνο αν η προδιαγραφή API αλλάξει.

Στο (Singh et al., 2005) ένα άλλο πρόβλημα με τα αρχεία καταγραφής είναι ότι απαιτούν κάποια μορφή σειριοποίησης δεδομένων σε ένα αρχείο κειμένου για την ανάλυση των αποφάσεων σε πολλαπλές δύσκολες εφαρμογές. Επίσης, οι σημερινοί μηχανισμοί συλλογής δεδομένων δημιουργούν υπερβολική επιβάρυνση, ιδιαίτερα για τη διατήρηση των αποτελεσμάτων του πειράματος για μελλοντική χρήση. Εάν η εφαρμογή εκτελείται σε έναν αριθμό κόμβων, μετά το πείραμα καταλήγει στο συμπέρασμα, οι χρήστες πρέπει να συνδεθούν σε όλα τα μηχανήματα και μη-αυτόματα να αντιγράψουν τα αρχεία μετρήσεων και τα αρχεία καταγραφής του συστήματος σε έναν απομακρυσμένο υπολογιστή για περαιτέρω ανάλυση. Αυτή είναι μια χρονοβόρα και επαναλαμβανόμενη διαδικασία, η οποία καθυστερεί την εκτέλεση της επόμενης σειράς πειραμάτων περιμένοντας οι πόροι για να γίνουν διαθέσιμοι. Μπορεί επίσης να οδηγήσει σε χαμένα αρχεία. Εάν ο ερευνητής επιθυμεί να αλλάξει τη συμπεριφορά της συλλογής, πρέπει να επαναμεταγλωττίσει και να αναδιατάξει την εφαρμογή, η οποία είναι μια επιρρεπής σε λάθη και χρονοβόρα διαδικασία κάθε φορά που κάποιος θέλει να αλλάξει τη συμπεριφορά της συλλογής. Το πολύ επιθετικό φιλτράρισμα μπορεί να "πετάξει" λεπτομέρειες που αποδεικνύονται ζωτικής σημασίας για την κατανόηση ορισμένων φαινομένων, με αποτέλεσμα την επανεκτέλεση του πειράματος με διαφορετικές ρυθμίσεις του φίλτρου.

Τα μειονεκτήματα του άρθρου (Sudarsan και Ribbens, 2010) είναι ότι η υψηλή χωρητικότητα στις υπολογιστικές πλατφόρμες είναι εξ ορισμού δαπανηρή, οπότε το κόστος της υποχρησιμοποίησης είναι υψηλό. Μια εργασία κυριαρχείται από ένα σχετικά μικρό αριθμό αυτών των εργασιών που είναι πιο δύσκολο να προγραμματιστούν αποτελεσματικά σε μια μεγάλη ομάδα. Ένα βασικό πρόβλημα

είναι ότι οι συμβατικοί παράλληλοι προγραμματιστές είναι στατικοί. Μια εργασία μια φορά εκχωρείται σε ένα σύνολο πόρων, που συνεχίζει να χρησιμοποιεί τις ίδιες πηγές μέχρι να τελειώσει την εκτέλεση. Όταν προγραμματίζονται εργασίες σύντομης εκτέλεσης οι εργασίες φθάνουν πάρα πολύ γρήγορα, θα πρέπει να είναι σε ουρά και να προγραμματιστούν στατικά. Αυτό συμβαίνει επειδή αυτές οι εργασίες έχουν γενικά μικρότερες απαιτήσεις σε επεξεργαστή και έτσι θα πρέπει να προγραμματιστούν άμεσα για την εκτέλεση. Ως αποτέλεσμα, αυτές οι εργασίες θα χρησιμοποιήσουν όλους τους επεξεργαστές στο σύστημα αφήνοντας πολύ λίγους ή καθόλου επεξεργαστές για την αλλαγή μεγέθους. Ως εκ τούτου, όλες οι εφαρμογές θα αναγκαστούν να εκτελεστούν στην εκκίνηση του μέγεθος του επεξεργαστή τους κατά τη διάρκεια της εκτέλεσής τους. Αυτό θα βελτιώσει τη συνολική χρησιμοποίηση του συστήματος, αλλά θα αυξήσει επίσης την επιμέρους εφαρμογή γύρω από το χρόνο. Αν οι εργασίες έρχονται με σχετικά μεγαλύτερα διαστήματα καθυστερήσεων, τότε θα είναι σε θέση να αναπτυχθούν και να χρησιμοποιούν ολόκληρο το σύστημα για την εκτέλεσή τους.

Στο (Carvalho και Cachopo, 2011) η έλλειψη ρεαλιστικών σημείων αναφοράς είναι ένας από τους παράγοντες που εμποδίζουν την ανάπτυξη, τη δοκιμή και την αποδοχή των συστημάτων λογισμικού μνήμης συναλλαγών (STM). Εφαρμόζοντας τα STMs σε μεγαλύτερα, πιο ρεαλιστικά σημεία αναφοράς, όπως το STMBench7 και το Lee-TM σημεία αναφοράς, δείχνουν συνήθως πολύ μεγάλη επιβάρυνση σε σύγκριση με την ενιαία νημάτωση διαδοχικής έκδοσης του σημείου αναφοράς. Επιπλέον δεν παρέχει ένα τεστ ορθότητας που να είναι σε θέση να ελέγξει εάν μια εκτέλεση έχει αποφέρει σωστά αποτελέσματα. Αυτό το σημείο αναφοράς έχει επίσης διάφορα μειονεκτήματα. Δεν κάνουν όλες οι εφαρμογές σημασιολογικές αξιολογήσεις των εξεταζόμενων STMs. Δεν είναι εύκολο να ενσωματώσει τις εφαρμογές Stamp σε ορισμένους αλγορίθμους STM, γιατί απαιτείται να τροποποιηθεί ο πηγαίος κώδικάς του και να αλλάξει ο τύπος του συνόλου των θέσεων μνήμης που προσπελάζονται από μια συναλλαγή. Ένας από τους περιορισμούς του σημείου αναφοράς LeeTM, ωστόσο είναι ότι δεν επιτρέπει την επέκτασή του σε νέες μορφές λειτουργιών και ερευνά παραλλαγές των σεναρίων του ανταγωνισμού. Επιπλέον, δεν υπάρχει δυνατότητα μεταβολής της ανάγνωσης / εγγραφής της αναφοράς, επειδή όλες οι συναλλαγές γράφουν κάτι, σε αντίθεση με τις περισσότερες εφαρμογές.

Στο άρθρο (Gotlieb και Bernard, 2006) η προσέγγιση της συμμετρικής δοκιμής παραμένει περιορισμένη στην εφαρμογή σε ένα περιορισμένο μέρος του Cosmo Java Card των API. Όταν ο ελεγκτής θέλει να εκμεταλλευτεί μια γεννήτρια τυχαίων δεδομένων δοκιμών, αντιμετωπίζει δύο κύρια προβλήματα. Το πρώτο είναι το κλασικό πρόβλημα πρόβλεψης, που απαιτείται ένας αυτόματος τρόπος ελέγχου της ορθότητας του αποτελέσματος. Το δεύτερο πρόβλημα είναι να προσδιοριστεί το επίπεδο ποιότητας ελέγχου που επιτεύχθηκε από μια τέτοια προσέγγιση δοκιμών. Σε γενικές γραμμές, είναι δύσκολο να ποσοτικοποιηθεί το πόσο αξιόπιστο είναι ένα πρόγραμμα που έχει δοκιμαστεί μόνο από τυχαία δεδομένα δοκιμών. Ο κύριος περιορισμός του συμμετρικού ελέγχου προκύπτει όταν κάποιος προσπαθεί να τον εφαρμόσει σε μη-συμμετρικές μεθόδους. Για να αντιμετωπιστεί αυτό το πρόβλημα, διερευνώνται άλλες ιδιότητες για να ελέγξουν την ορθότητα του αποτελέσματος της Java Card των APIs.

Τα μειονεκτήματα της τεχνολογίας RFID αναλύονται στο άρθρο (Bellotti et al., 2008). Εξακολουθεί να είναι προβληματική για τα υπάρχοντα συστήματα ενημέρωσης περιεχομένου για την κάλυψη απαιτήσεων των τελικών χρηστών επειδή οι ταξινομητές δεν είναι σε θέση να αναγνωρίσουν επαρκώς όλες τις πιθανές περιπτώσεις των διάχυτων εφαρμογών ή τους κανόνες των έμπειρων συστημάτων, δεν είναι σε θέση να συλλάβουν σχετικά με τις καταστάσεις, ιδίως σχετικά με τις δυναμικές υποκειμενικές (σχετίζονται με το χρήστη) πτυχές. Το oDect των APIs παρέχει ειδική στήριξη στον προγραμματιστή για να αντιμετωπίσει εύκολα δύο χαρακτηριστικά προβλήματα των εφαρμογών RFID, το πρόβλημα των συγκρούσεων και την ετικέτα εντοπισμού τρεμοπαίγματος. Το πρόβλημα των συγκρούσεων έχει ως εξής. Τα αποτελέσματα από μια απλή λειτουργία σάρωσης είναι συνήθως ατελή, διότι όλες οι ετικέτες δεν μπορεί να είναι ανιχνεύσιμες σε κάθε σάρωση. Το άλλο τυπικό πρόβλημα που επηρεάζει την τεχνολογία RFID είναι η ανίχνευση της ετικέτας που τρεμοπαίζει. Λόγω του προβλήματος σύγκρουσης ακόμη και αν ελαχιστοποιηθούν με την παραπάνω έννοια που εισήχθη από τη περίοδο σάρωσης μερικές ετικέτες μπορεί συνεχώς να εμφανίζονται και να εξαφανίζονται σε διαδοχικές σαρώσεις, η οποία μπορεί να οδηγήσει σε ασταθή εντοπισμό των ετικετών. Η διεπαφή oDect δεν υποστηρίζει τοποθέτηση.

Στο τελευταίο άρθρο (Shan και Hua, 2006) αυτής της κατηγορίας, στη μελέτη μας δεν αναφέρονται μειονεκτήματα όσον αφορά τα πλαίσια διαδικτυακής εφαρμογής.

3.2.2 Μελέτη των μειονεκτημάτων του τομέα ποιότητας των APIs

Σε αυτή τη ενότητα μελετάμε τα μειονεκτήματα των APIs που σχετίζονται με τα χαρακτηριστικά ποιότητας. Στα άρθρα (Stylos et al., 2008, Kim et al., 2011, Stylos και Myers, 2008, Hoffman και Strooper, 2000) μελετάμε τα προβλήματα που αφορούν τη δυνατότητα επαναχρησιμοποίησης των APIs, δυσκολίες όσον αφορά την χρηστικότητα των APIs, καθώς επίσης και τις επανασχεδιάσεις στην ποιότητα του λογισμικού.

Το άρθρο (Stylos et al., 2008) αναφέρει ότι τα APIs είναι συχνά δύσκολα και χρονοβόρα για τους προγραμματιστές να τα χρησιμοποιήσουν. Η χρήση των APIs που δεν πληρούν τις ειδικές απαιτήσεις των σχεδιαστών μπορεί συχνά να είναι κουραστική, δύσκολη, ή ακόμη και αδύνατη. Ως αποτέλεσμα συχνά οι προγραμματιστές περνούν πολύ χρόνο προσπαθώντας να κάνουν τις υπάρχουσες εργασίες τους στα APIs και μπορεί να καταλήξουν να γράψουν κώδικα από την αρχή αντί να χρησιμοποιήσουν ένα δύσκολο στη χρήση API. Το API θεωρήθηκε ως παροχή χαμηλού επιπέδου λειτουργικότητας, ενώ η συγκεκριμένη κατηγορία χρηστών - προγραμματιστές εφαρμογών είχαν υψηλότερου επιπέδου στόχους, δίνοντας προτεραιότητα στην απλότητα της χρήσης του έλεγχου και τη διαφάνεια, οι περιπτώσεις χρήσης τους δεν ήταν τόσο περίπλοκες να απαιτούν σημαντικά ποσά διακριτότητας ή ευελιξίας και ο χρόνος που δαπανάται για την κατανόηση και τον εντοπισμό σφαλμάτων της χρήσης του API ήταν πολύ περιορισμένος σε σύγκριση με τους προγραμματιστές της πλατφόρμας.

Στο (Kim et al., 2011) οι μηχανικοί λογισμικού συχνά αποφεύγουν τις επανασχεδιάσεις όταν περιορίζονται από την έλλειψη πόρων. Διαπιστώθηκε ότι ένα υψηλό ποσοστό των επανασχεδιάσεων μερικές φορές ακολουθείται από μια αυξανόμενη αναλογία των αναφορών σε σφάλματα. Βρήκαν ότι σφάλματα εισάγονται από ελλιπείς ή λανθασμένες επανασχεδιάσεις, παρόλο που η αρχική πρόθεση της επανασχεδίασης ήταν να βελτιωθεί η συντηρησιμότητα του λογισμικού. Αν και η πρόθεση της επανασχεδίασης είναι η βελτίωση της συντήρησης του λογισμικού, η επανασχεδίαση θα μπορούσε να είναι ενδεχομένως

επιρρεπής σε λάθη. Πολλά εργαλεία επανασχεδίασης κάνουν μια φτωχή εργασία των λαθών της επικοινωνίας και οι προγραμματιστές δεν τα αξιοποιούν τόσο αποτελεσματικά όσο θα μπορούσαν. Οι προγραμματιστές συχνά κάνουν επανασχεδιάσεις με άλλο πρόγραμμα αλλαγής-νήμα επανασχεδίασης και αυτά δεν είναι καλά υποστηριζόμενα από τα υπάρχοντα εργαλεία της επανασχεδίασης. Η μέθοδος του εντοπισμού της προέλευσης ενός σφάλματος είναι πολύ περιορισμένη, ιδίως όταν η διόρθωση σφαλμάτων περιλαμβάνει επιπλέον κώδικα. Αυτό το ζήτημα στην εγκυρότητα της μελέτης μπορεί να έχει επηρεάσει τα αποτελέσματα στο χρόνο για την αντιμετώπιση του συγκεκριμένου σφάλματος.

Στο άρθρο (Stylos και Myers, 2008) τα APIs στα οποία οι μέθοδοι ήταν βοηθητικά αντικείμενα είναι δύσχρηστα, διότι: (α) Δεν αναμένεται εύρεση μιας μεθόδου, οι συμμετέχοντες θέτουν μερικές φορές ένα ερώτημα επιλογής τους(σωστό) από την αρχική κλάση. (β) Οι προγραμματιστές έπρεπε να αναγνωρίσουν ότι απαιτείται η χρήση μιας επιπρόσθετης κλάσης. (γ) Οι προγραμματιστές έπρεπε να εντοπίσουν την επιπλέον κλάση. Αρκετές από τις αναπάντεχες δυσκολίες που παρατηρήθηκαν είναι ότι οι προγραμματιστές που έχουν ως αποτέλεσμα τα APIs δεν ακολουθούν αυτό το απλό μοντέλο, με το να μην παρέχουν έναν προεπιλεγμένο δομητή (ή οποιονδήποτε δημόσιο δομητή). Οι προγραμματιστές ξόδεψαν περισσότερο χρόνο προσπαθώντας να καταλάβουν τις κλάσεις, οι οποίες κατέληξαν λαμβάνοντας περισσότερο χρόνο. Σε ένα μεγαλύτερο και λιγότερο οροθετημένο API, η στρατηγική αυτή θα ήταν πιθανόν λιγότερο αποτελεσματική. Ένα επιπλέον ζήτημα που αναδεικνύεται από αυτή τη μελέτη ήταν η δυσκολία των προγραμματιστών να βρουν κλάσεις που είναι χρήσιμες για την έναρξη της εξέτασης ενός API. Οι προγραμματιστές στις μελέτες συχνά αλλοιώνουν ή παρακάμπτουν πλήρως τα κείμενα τεκμηρίωσης της κλάσης, επιλέγοντας να αναφερθούν στην λίστα των μεθόδων και των πεδίων αντί αυτών.

Στο άρθρο (Hoffman και Strooper, 2000) παρουσιάζεται ένα πλαίσιο δοκιμής για την δημιουργία περιπτώσεων δοκιμών από Z προδιαγραφές. Ενώ αυτές οι προσεγγίσεις προσφέρουν μεγάλες δυνατότητες για την αυτοματοποίηση, είναι δύσκολο να εφαρμοστούν στην πράξη διότι η επίσημη προδιαγραφή για βιομηχανικό λογισμικό είναι συνήθως μη-διαθέσιμη. Η εφαρμογή για μια συγκεκριμένη περιοχή είναι απλή αλλά κουραστική έχει δύο μειονεκτήματα: είναι

δύσχρηστη και δεν είναι διαθέσιμη στη μορφή ProcessTuple (μόνο με τη μορφή Vector). Δοκιμές αυτών των εφαρμογών είναι σημαντικές, διότι οι CWR βλάβες προκαλούν συχνά διακοπή της εφαρμογής. Δυστυχώς, ο έλεγχος είναι επίσης ακριβός, διότι οι πολιτικές CWR κυριαρχούνται από ειδικές περιπτώσεις. Γράφοντας ένα προσαρμοσμένο πρόγραμμα οδήγησης για κάθε πολιτική είναι σίγουρα πάρα πολύ ακριβό. Είναι εξαιρετικά δαπανηρή σε χρόνο, τόσο πολύ έτσι ώστε οι εξαρτώμενες περιοχές πρέπει να διατηρούνται αρκετά μικρές.

Τα μειονεκτήματα αυτού του άρθρου (Liu et al., 2009) είναι ότι το λογισμικό με χαμηλή ποιότητα θα οδηγήσει στην αποτυχία του συστήματος πληροφοριών. Η πλατφόρμα δοκιμής με υψηλή ποιότητα μπορεί να βρει περισσότερες δυσλειτουργίες, οι οποίες δεν συνάδουν με τις απαιτήσεις. Οι διαφορετικές λέξεις-κλειδιά για την ίδια περιγραφή μπορεί να καταστεί δύσκολο για τους μηχανικούς του λογισμικού να κατανοήσουν και να σχεδιάσουν τις πλατφόρμες δοκιμής.

Το τελευταίο άρθρο (Zibran et al., 2011) αυτής της κατηγορίας ασχολείται με τα θέματα ευχρηστίας του API συχνά αυξάνουν το κόστος της ανάπτυξης (π.χ., χρόνος, προσπάθεια) και μειώνουν την ποιότητα του κώδικα. Οι προγραμματιστές έγραφαν τον κώδικα του πελάτη συχνά σκεπτόμενοι από τα προβλήματα δυνατότητας χρησιμοποίησης των APIs που χρησιμοποιούν. Τέτοια προβλήματα χρηστικότητας μειώνουν την παραγωγικότητα του προγραμματιστή και προκαλούν περιττή πολυπλοκότητα στον κώδικα του πελάτη. Μόλις ένα API έχει αναπτυχθεί, είναι δύσκολο να αλλάξει, γιατί οποιαδήποτε αλλαγή μπορεί να σπάσει τον κώδικα του πελάτη που απαιτεί αντίστοιχες αλλαγές (μετάβαση από τα παλιά APIs σε νεότερα) σε όλες τις εφαρμογές που καλεί αυτό το API. Μερικές αναφορές σφάλματος επισημαίνουν προβλήματα στους χρήστες για να πάρουν προειδοποίηση ή μηνύματα λάθους, λόγω της χρήσης απαρχαιωμένων μεθόδων.

3.2.3 Μελέτη των μειονεκτημάτων του τομέα ανάλυσης των APIs

Στην τρίτη κατηγορία μελετάμε τα μειονεκτήματα που σχετίζονται με την ανάλυση των APIs. Πιο συγκεκριμένα στα (Sarkar et al., 2007, Tselikas et al., 2007, Fujiwara et al., 2003 και Ratiu και Jurjens, 2008) παρουσιάζονται ζητήματα που σχετίζονται με τα μειονεκτήματα των μετρικών στις διασυνδέσεις προγραμματισμού εφαρμογών. Στο (Sarkar et al.,2007) οι κυκλικές εξαρτήσεις αναιρούν άμεσα πολλά από τα οφέλη της διαμόρφωσης. Είναι προφανώς πιο

δύσκολο να προβλέψουν τις συνέπειες της αλλαγής μιας μονάδας, αν και εξαρτάται και εξαρτιόταν από άλλες μονάδες. Οι κυκλικές εξαρτήσεις γίνονται ακόμη πιο προβληματικές όταν οι μονάδες οργανώνονται με τη μορφή οριζόντιων στρωμάτων σε ένα μεγάλο σύστημα λογισμικού. Οι μετρικές, τα κυκλωματικά μέτρα, ο δείκτης συντηρησιμότητας, το μέτρο διαμόρφωσης, η μετρική σύζευξης-συνοχής, η μετρική με βάση μη-δομημένες πληροφορίες, σίγουρα δεν μετρούν την ομοιότητα του σκοπού ή της κοινότητας των στόχων. Οι μετρικές αυτές επίσης δεν μετρούν το βαθμό στον οποίο μια μονάδα ενθυλακώνει εσωτερικά (δηλαδή μη-API) λειτουργίες και τις κρατά από το να εκτεθούν στον εξωτερικό κόσμο. Η ανίχνευση υπονοούμενων εξαρτήσεων συχνά απαιτεί μια δυναμική ανάλυση του χρόνου εκτέλεσης του λογισμικού. Η ανάλυση αυτή είναι χρονοβόρα και δύσκολη για την εκτέλεση πολύπλοκων εφαρμογών, ιδίως τις εφαρμογές που τρέχουν σε εκατομμύρια γραμμές κώδικα και τα οποία περιλαμβάνουν επιχειρηματικά σενάρια που μπορεί να εκτελεστούν σε χιλιάδες, δημιουργεί μια διαφορετική υπονοούμενη εξάρτηση μεταξύ των μονάδων. Η μέτρηση των χαρακτηριστικών αυτών με ποσοτικό τρόπο είναι δύσκολη, δεδομένου ότι είναι δύσκολο να προβλεφθεί εκ των προτέρων αν μια συγκεκριμένη μονάδα είναι πιθανό να αλλάξει. Οι βασικές τιμές των μετρικών λαμβάνουν μια στροφή προς το χειρότερο, όταν η αρχική αρθρωτή δομή του λογισμικού έχει καταστραφεί.

Στο (Tselikas et al., 2007) οι φορείς εκμετάλλευσης δικτύων μόνο ωστόσο δεν είναι σε θέση να αναπτύξουν υπηρεσίες στα δίκτυά τους τόσο γρήγορα όσο απαιτεί η ζήτηση. Οι αλληλεπιδράσεις με το δίκτυο πυρήνα τείνουν να είναι περίπλοκες, έτσι η ενσωμάτωση των υπηρεσιών δίνει το δικαίωμα στον πυρήνα του δικτύου και το σημαντικότερο απαιτώντας τους φορείς του δικτύου για τη διαχείριση και τη διατήρηση των υπηρεσιών αυτών μετά την ανάπτυξη, αναπόφευκτα γίνεται μια αργή και επίπονη διαδικασία. Προηγμένες υπηρεσίες μπορεί να περιλαμβάνουν πολλές αλληλεπιδράσεις μεταξύ των διαφορετικών δικτύων και οντοτήτων εφαρμογής, οι οποίες μερικές φορές δημιουργούνται σε πραγματικό χρόνο εισάγοντας μεγάλες καθυστερήσεις. Οι περισσότερες από τις απορρίψεις των αιτήσεων προκαλούνται από την έλλειψη πόρων, μηχανημάτων υποδοχής όπως φαίνεται από τις εξαιρέσεις που ρίχνονται στη πλευρά του πελάτη. Το πιο σημαντικό συμπέρασμα που αντλείται από την ανάλυση της απόδοσης είναι ότι η επίδραση στην απόδοση που υπέστη από τη χρήση ενός

επιπέδου ενδιάμεσου λογισμικού εφαρμόζοντας ανοικτές διεπαφές μεταξύ του δικτύου και των υπηρεσιών είναι μια αποδεκτή επιβάρυνση.

Το (Fujiwara et al., 2003) αναφέρει ότι ένα αντικειμενοστρεφές πλαίσιο είναι ένα επαναχρησιμοποιήσιμο σχέδιο ή ένα σύστημα ή υποσύστημα που υλοποιείται μέσω μιας συλλογής συγκεκριμένων και αφηρημένων κλάσεων και τη συνεργασία τους. Παρέχει μια γενική λύση σε μια σειρά από παρόμοια προβλήματα σε ένα πεδίο εφαρμογής. Ωστόσο, είναι δύσκολο να το εισάγει στην οργάνωση που έχει με την παραδοσιακή μέθοδο επαναχρησιμοποίησης, είναι δύσκολο να μεταφέρει το νέο πλαίσιο για την ανάπτυξη, καθώς οι προγραμματιστές στο τμήμα χρησιμοποιούν την αρχική τεχνική επαναχρησιμοποίησης που είναι μια συμβατική μονάδα με βάση την επαναχρησιμοποίηση.

Στα μειονεκτήματα αυτού του άρθρου (Ratiu και Jurjens, 2008) περιλαμβάνεται ότι η αναπαράσταση των εννοιών επηρεάζει την καταλληλότητα του τομέα και την ικανότητα των χρηστών να χειριστούν και να συνδυάσουν τις (υλοποιήσεις) έννοιες σε επίπεδο API. Η αναπαράσταση των εννοιών επηρεάζει τον τρόπο με τον οποίο οι πελάτες του API μπορούν να χειριστούν και να συνθέσουν τις έννοιες σε επίπεδο API. Οι έννοιες που δεν αναφέρονται ρητά στο API είναι διαθέσιμες στους χρήστες του API μόνο σε συνδυασμό με άλλες έννοιες και αυτό επηρεάζει αρνητικά την ορατότητα των εννοιών σε επίπεδο API. Στην εννοιολογική πολυπλοκότητα όσο μεγαλύτερη είναι η μετρική CC τόσο πιο περίπλοκοι συνδυασμοί των εννοιών χρησιμοποιούνται στο API. Περίπλοκοι συνδυασμοί των εννοιών είναι δύσκολο να κατανοηθούν και τα στοιχεία του προγράμματος για την εφαρμογή τους είναι χρήσιμα μόνο σε ειδικές περιπτώσεις. Στην αναπαράσταση ασάφειας μια έννοια είναι διφορούμενη που αναπαρίσταται στο API εάν παρουσιάζεται από διαφορετικούς τύπους. Κατά συνέπεια, οι χρήστες της βιβλιοθήκης δεν μπορούν να χρησιμοποιήσουν ομοιόμορφα αυτήν την έννοια σε επίπεδο API και πολλές φορές θα πρέπει να την μετατρέψουν μεταξύ διαφορετικών αναπαραστάσεών του.

Τα άρθρα (Farooq et al., 2010, Rao και Kak, 2011, Kawrykow και Robillard, 2009) παρουσιάζουν τα προβλήματα όσον αφορά τους αλγόριθμους εντοπισμού σφαλμάτων καθώς επίσης, και τις δυσκολίες ανίχνευσης και διόρθωσης σφαλμάτων.

Στο άρθρο (Farooq et al., 2010) ενώ οι έλεγχοι χρηστικότητας στο εργαστήριο παρέχουν βαθιά ανατροφοδότηση με βάση τους πραγματικούς χρήστες αλληλεπιδρώντας με τη διεπαφή, έχουν πολλά μειονεκτήματα. Για API δοκιμές ευχρηστίας συγκεκριμένα, οι προγραμματιστές λογισμικού με εξειδικευμένες γνώσεις τομέα μπορεί να είναι δύσκολο να προσληφθούν. Ένα άλλο μειονέκτημα είναι ότι οι δοκιμές ευχρηστίας χρειάζονται σημαντικό χρόνο για το σχεδιασμό της μελέτης. Αυτό μπορεί να αποτελέσει σημαντική πρόκληση για τις ομάδες προϊόντων που ανάλογα με το στάδιο του κύκλου ζωής ανάπτυξης λογισμικού, συχνά απαιτούν γρήγορη ανατροφοδότηση σχετικά με την ευχρηστία των API. Επιπλέον, οι έλεγχοι ευχρηστίας είναι ακριβοί (π.χ., το κόστος της πρόσληψης των χρηστών που χρησιμοποιούν τις εγκαταστάσεις και το εργαστήριο). Συγκρίθηκαν η ευρετική αξιολόγηση, οι κατευθυντήριες γραμμές του λογισμικού, τα γνωστικά περάσματα, καθώς και οι δοκιμές ευχρηστίας και βρέθηκε ότι η ευρετική αξιολόγηση εντόπισε τα πιο σοβαρά λάθη χρηστικότητας, που δεν είχαν εντοπιστεί μέσω των κατευθυντηρίων γραμμών με το λιγότερο ποσοστό προσπάθειας. Μια πρώτη συνάντηση με κακοσχεδιασμένα API μπορεί να αφήσει μια μόνιμη εντύπωση της πολυπλοκότητας και μπορεί να αποθαρρύνει ακόμη τους χρήστες από την υιοθέτηση μιας συγκεκριμένης τεχνολογίας. Οι API αξιολογήσεις ίδιας χρηστικότητας εκθέτουν το σκεπτικό του σχεδιασμού του API και εννοιολογικά αποκαλύπτει ελλείψεις στο σχεδιασμό του.

Τα προβλήματα που διερευνά το άρθρο (Rao και Kak, 2011) είναι οι δύο βασικές προκλήσεις είναι οι εξής: η αναντιστοιχία λεξιλογίου και η έλλειψη διαθεσιμότητας των καλών συνόλων δεδομένων αξιολόγησης. Η αναντιστοιχία λεξιλογίου προκύπτει από τη χρήση των συντομογραφιών και αλληλουχιών των ονομάτων των μεταβλητών και αναγνώρισης από τους προγραμματιστές κατά την ανάπτυξη του κώδικα. Το κυριότερο μειονέκτημά του είναι η εν γένει μεγάλη διάσταση και η σποραδικότητα των διανυσμάτων του εγγράφου. Οι δυναμικές τεχνικές εντοπισμού σφαλμάτων υποφέρουν από το μειονέκτημα ότι βασίζονται στην ύπαρξη των δύο ροών ελέγχου, το πέρασμα της ροής του ελέγχου και της προβληματικής ροής ελέγχου. Τα Unigram μοντέλα έχουν το εξής μειονέκτημα: αν μια λέξη δεν έχει εμφανιστεί ποτέ σε ένα έγγραφο, παίρνει την τιμή 0, που είναι η πιθανότητα εμφάνισης στο διάνυσμα πιθανότητας για το συγκεκριμένο έγγραφο. Αυτό μπορεί να δημιουργήσει προβλήματα σε πιθανολογικούς τύπους

διανύσματος. Τα εξελιγμένα μοντέλα όπως LDA, LSA, και CBDM δεν ξεπερνούν απλούστερα μοντέλα όπως Unigram ή VSM για IR με βάση τον εντοπισμό σφαλμάτων για μεγάλα συστήματα λογισμικού. Το μοντέλο LDA δίνει τη χειρότερη επίδοση χωρίς ερωτήματα / σφάλματα.

Στο άρθρο (Kawrykow και Robillard, 2009) τα APIs μπορούν να αναπτυχθούν πολύ μεγάλα και πολύπλοκα και οι προγραμματιστές που χρησιμοποιούν αυτά δεν μπορούν να ανακαλύψουν όλη τη λειτουργικότητα που προσφέρουν. Τα API εξελίσσονται ανεξάρτητα από τα έργα που βασίζονται σε αυτά, και οι προγραμματιστές μπορούν να εξακολουθούν να αγνοούν τις βελτιώσεις στο API που θα μπορούσαν να οδηγήσουν σε βελτιώσεις στον κώδικά τους. Τα API, μερικές φορές εξελίσσονται με ένα συμβατό τρόπο, χωρίς κανένα στοιχείο να αναγράφεται ως ξεπερασμένο. Η εκ νέου εφαρμογή των υπηρεσιών που προσφέρονται μέσω των APIs μπορεί να έχει πολλές αρνητικές επιπτώσεις στην ποιότητα του συστήματος, όπως: μείωση διαμόρφωσης, περίπλοκο κώδικα πελάτη, κρυμμένη πρόθεση του σκοπού μιας δήλωσης, την ελαττωμένη απόδοση και την ενδεχόμενη απαξίωση. Παρά το γεγονός ότι η άμεση επίπτωση της μη βέλτιστης χρήσης του API είναι δύσκολο να εκτιμηθεί, η μακροπρόθεσμη επίδραση στην διατηρησιμότητα του συστήματος είναι απίθανο να είναι καλή. Ένας περιορισμός της προσέγγισης είναι ότι ανιχνεύει μόνο μια ειδική κλάση για τις περιπτώσεις όπου η χρήση API μπορεί να βελτιωθεί, εκείνες που αφορούν τις μεθόδους πελάτη που μιμούνται τη λειτουργικότητα που παρέχεται από τις επιμέρους μεθόδους της βιβλιοθήκης. Δύο βασικοί παράγοντες που επηρεάζουν ενδεχομένως τα αποτελέσματα που αναφέρθηκαν είναι η επιλογή των στόχων των έργων και το εγχειρίδιο αξιολόγησης των αποτελεσμάτων της προσέγγισης. Οι API απομιμήσεις μπορούν να προκύψουν μέσα από μια σειρά παραγόντων και συνήθως προσθέτουν περιττή πολυπλοκότητα στον κώδικα πελάτη, ένα γνωστό εμπόδιο στην συντηρησιμότητα.

Τρία άρθρα αυτής της κατηγορίας τα (Perkins, 2005, Xing και Stroulia, 2007, Gharaibeh et al., 2007) ασχολούνται με τα μειονεκτήματα της επανασχεδίασης των βιβλιοθηκών διασύνδεσης προγραμματισμού εφαρμογών.

Η προσέγγιση της επανασχεδίασης έχει πολλά μειονεκτήματα αυτά αναφέρει το άρθρο (Perkins, 2005). Οι αλλαγές περιορίζονται σε αυτές που υποστηρίζονται

από το εργαλείο επανασχεδίασης, παρά από αυθαίρετες που μπορούν να εκφραστούν με τη σύνταξη του κώδικα. Η προσέγγιση αυτή εισάγει μια νέα γλώσσα ή μια μορφή αρχείου για να εκφράσει τις αλλαγές, και ένα ξεχωριστό αρχείο που θα πρέπει να αποστέλλεται στους πελάτες. Σε σύγκριση με τη συμπεριλαμβανόμενη προτεινόμενη αλλαγή στο αρχείο προέλευσης, οι επιλογές αυτές το καθιστούν πιο δύσκολο για τους προγραμματιστές της βιβλιοθήκης για την παραγωγή / αναθεώρηση των αλλαγών τους και για τους χρήστες να αξιολογήσουν τις αλλαγές. Η προσέγγιση αναγκάζει τους προγραμματιστές της βιβλιοθήκης να χρησιμοποιήσουν ένα ειδικό εργαλείο για την καταγραφή των επανασχεδιάσεων, αντί να χρησιμοποιούν το συνηθισμένο περιβάλλον ανάπτυξης τους για να επεξεργαστούν τον κώδικα, μια διαδικασία με την οποία είναι ήδη εξοικειωμένοι και είναι εύκολη. Η τεχνική αυτή απαιτεί οι προγραμματιστές να κάνουν τη δουλειά τους δύο φορές: μία φορά με τον συνήθη τρόπο, έπειτα μια δεύτερη φορά να καταγράφονται. Η επανάληψη των εργασιών είναι τόσο ενοχλητική και επιρρεπής σε λάθη, ειδικά όταν γίνεται σε ένα άγνωστο περιβάλλον. Το αποτέλεσμα της πραγματοποίησης των αλλαγών μπορεί να ελεγχθεί μόνο μετά την εκτέλεση της επανασχεδίασης, γεγονός που καθιστά δύσκολο να καθοριστεί αν η αλλαγή είναι κατάλληλη και χωρίς λόγο είναι δύσκολο να υποστηρίξει εάν είναι απαραίτητη. Ο κώδικας του πελάτη μπορεί να αποτύχει να εκτελεστεί και να μεταγλωττιστεί, αν τα παλαιά στοιχεία της διεπαφής αφαιρεθούν. Η μεταγλώττιση μπορεί να δώσει προειδοποίηση ότι η βιβλιοθήκη χρησιμοποιεί απαρχαιωμένες μεθόδους.

Στο άρθρο (Xing και Stroulia, 2007) η τεκμηρίωση γράφεται μερικές φορές σε πολύ συμπαγή-ακόμη- κρυπτογραφική γλώσσα όχι εύκολα κατανοητή από τους περισσότερους προγραμματιστές εφαρμογών. Η προσέγγιση αυτή μεταθέτει το βάρος της αναβάθμισης του πηγαίου κώδικα για τη βιβλιοθήκη αλλαγών διασύνδεσης από τους προγραμματιστές εφαρμογών για τους προγραμματιστές της βιβλιοθήκης. Το κύριο σφάλμα του είναι ότι θεωρεί, ότι η αλλαγή και ο μετασχηματισμός είναι μη αυτόματα καθορισμένα για κάθε αλλαγή διεπαφής, η οποία απαιτεί μια σημαντική προσπάθεια για να γράψει και να διατηρήσει. Επιπλέον, η διαδικασία της μετάβασης είναι ευαίσθητη στην πληρότητα και την ορθότητα των προδιαγραφών του προγραμματιστή της βιβλιοθήκης. Προβλήματα μετάβασης, μπορούν να προκληθούν από τις αλλαγές στις ιδιότητες του, όπως η

ορατότητα, οι τροποποιητές και οι αλλαγές στη σχέση της με άλλα στοιχεία, όπως ο σχετικός τύπος δεδομένων, η δήλωση εξαίρεσης και η ιεραρχία κληρονομικότητας. Επιπλέον, το API στοιχείο του πλαισίου μπορεί μερικές φορές να αλλάξει δραματικά, συμπεριλαμβανομένης της κατάργησης ορισμένων από τα στοιχεία του και τις αλλαγές σε όλα τα σχετικά στοιχεία τους. Μια άλλη πιθανή αιτία για την αποτυχία του Diff-CatchUp να συστήσει αντικαταστάσεις είναι το γεγονός ότι οι κλάσεις των χρηστών και οι μέθοδοι εφαρμόζουν πολύπλοκες λειτουργίες που μερικές φορές γίνονται ασύνδετες. Τα σφάλματα μεταγλώττισης και οι προειδοποιήσεις είναι ουσιαστικά συντακτικά προβλήματα που οι προγραμματιστές εφαρμογής έχουν να επιλύσουν πριν να μπορέσουν να δημιουργήσουν την εφαρμογή τους με επιτυχία και να το επαναλάβουν με το εξελιγμένο στοιχείο του πλαισίου.

Στο (Gharaibeh et al., 2007) είναι προφανές ότι οι επανασχεδιάσεις έχουν κάνει την αρχική πρόθεση του πολυμορφισμού άκυρη, κατά συνέπεια τη δημιουργία ασυμβατότητας και σημασιολογικών λαθών μεταξύ του κώδικα πελάτη και του API. Οι αλλαγές επανασχεδίασης γίνονται ανεξάρτητα στο API και του κώδικα πελάτη, η σημασιολογία του προγράμματος, συμπεριλαμβανομένων τόσο του API και του κώδικα του πελάτη μπορεί να αλλάξει κατά λάθος. Οι συγκρούσεις προκύπτουν λόγω της συμβατότητας-σπάζοντας αλλαγές επανασχεδίασης στο API. Η αλλαγή στη δομή της κληρονομικότητας μπορεί να προκαλέσει απροσδόκητη συμπεριφορά στον κώδικα του πελάτη. Αυτό οφείλεται στη μεταβολή της διεπαφής μεταξύ του πελάτη και του API. Οι επανασχεδιάσεις μπορούν να επηρεάσουν τη δομή της κληρονομικότητας και μεταφέρουν την εφαρμογή σε μια ασυνεπή κατάσταση, στην οποία η εφαρμογή συμπεριφέρεται με διαφορετικό τρόπο. Η απευθείας ενημέρωση λογισμικού προσθέτει επιπλέον πολυπλοκότητα στην παραδοσιακή συντήρηση του λογισμικού, διότι εκτός από την ενημέρωση του κώδικα, την κατάσταση του προγράμματος θα πρέπει επίσης να ενημερωθεί για τη νέα έκδοση.

Στα άρθρα (Ellis et al., 2007, Matsuzawa και Ikeda, 1998, Ren et al., 2011) μελετάμε μερικά από τα μειονεκτήματα των αντικειμενοστρεφών μοντέλων. Στο άρθρο (Ellis et al., 2007) μια μελέτη χρήστη συγκρίνοντας τη χρηστικότητα του προτύπου εργοστάσιο και των κατασκευαστών στα σχέδια API βρέθηκαν πολύ

σημαντικά αποτελέσματα τα οποία δείχνουν ότι το εργοστάσιο είναι καταστρεπτικό για τη χρηστικότητα API σε πολλές διαφορετικές καταστάσεις. Τα αποτελέσματα έδειξαν ότι οι χρήστες απαιτούν σημαντικά περισσότερο χρόνο για να δημιουργήσουν ένα αντικείμενο με ένα αφηρημένο εργοστάσιο σε σχέση με ένα δομητή κατά την εκτέλεση του πλαισίου. Ένας προγραμματιστής μπορεί να χρησιμοποιήσει μόνο ένα μικρό μέρος της συνολικής λειτουργικότητας ενός API, αλλά μαθαίνοντας ακόμη ότι το υποσύνολο είναι ένα δύσκολο έργο για τους νέους προγραμματιστές. Η δημιουργία αντικείμενων από το πρότυπο αφηρημένου εργοστασίου που χρησιμοποιούνται στο API είναι πολύ πιο χρονοβόρα από τους δομητές, ανεξάρτητα από το πλαίσιο ή το επίπεδο της εμπειρίας του προγραμματιστή χρησιμοποιώντας το API. Το πρότυπο δεν μπορεί να χρησιμοποιηθεί όταν ο σχεδιαστής θέλει να κρύψει την ύπαρξη των υποκατηγοριών ή την εξάλειψη των συγκεκριμένων εξαρτήσεων. Το πρότυπο του αφηρημένου εργοστασίου είναι αποδεδειγμένα πιο δύσκολο από ό,τι οι κατασκευαστές για τους προγραμματιστές για να το χρησιμοποιούν, ανεξάρτητα από το πλαίσιο. Το πρότυπο αφηρημένου εργοστασίου διαβρώνει τη χρηστικότητα των APIs στα οποία αυτό χρησιμοποιήθηκε.

Το άρθρο (Matsuzawa και Ikeda, 1998) αναφέρει ότι οι απρογραμμάτιστοι συνδυασμοί των διαφόρων πλαισίων συχνά προκαλούν το πρόβλημα, το οποίο σε μεγάλο βαθμό μειώνει την αναμενόμενη αποτελεσματικότητάς τους. Υπάρχουν δύο σοβαρά προβλήματα, 1) η μεγάλη πολυπλοκότητα του κώδικα και 2) η μη ευελιξία, της ισχυρής σχέσης μεταξύ των ενδιάμεσων λογισμικών. Η κατηγορία του Μοντέλου και η κατηγορία της Προβολής συνδέονται αντίστοιχα με διαφορετικά ενδιάμεσα λογισμικά. Αυτό έχει το εξής μειονέκτημα. Οι περισσότερες κλάσεις σε κάθε κατηγορία εξαρτώνται από τις άλλες κατηγορίες. Όταν μια κλάση σε ένα ενδιάμεσο λογισμικό είναι τροποποιημένο, έχουν τροποποιηθεί πολλές άλλες κλάσεις που ανήκουν σε άλλα ενδιάμεσα λογισμικά. Επομένως είναι απαραίτητο να είναι γνωστά και τα δυο ενδιάμεσα λογισμικά. Η κλάση του διαχειριστή διασύνδεσης εξακολουθεί να εξαρτάται τόσο από το περιβάλλον GUI όσο και από την κλάση των βιβλιοθηκών OODB. Δεδομένου ότι η τροποποίηση της κλάσης του διαχειριστή διασύνδεσης επηρεάζει την κλάση Προβολής και την κλάση Μοντέλου σε μεγάλο βαθμό, είναι δύσκολο να αλλάξει ο συνδυασμός

ανάμεσα στην κλάση του διαχειριστή διασύνδεσης και της κλάσης του Μοντέλου / Προβολής.

Το άρθρο (Ren et al., 2011) μελετά τα προβλήματα που σχετίζονται με την ανάπτυξη του λογισμικού. (1) Διαφορετικές εταιρείες χρειάζονται διαφορετικό λογισμικό, χωρίς ένα καθολικό λογισμικό που να μπορεί να εφαρμοστεί σε κάθε επιχείρηση διαχείρισης του συστήματος πληροφοριών. Ο κύκλος ανάπτυξης του λογισμικού είναι μακρύς, τα σφάλματα στον κώδικα, η αξιοπιστία του λογισμικού είναι κακή, δεν είναι ευνοϊκή για τη διατήρηση και την αναβάθμιση. Η ανάγκη της επιχείρησης για λογισμικό πιο σύνθετο, δεν είναι ικανοποιημένη από το λογισμικό που παρέχεται από εταιρείες ανάπτυξης λογισμικού των επιχειρήσεων.
2) Η ποιότητα του λογισμικού είναι αναξιόπιστη, η συντήρηση του λογισμικού είναι κακή, οι χρήστες συνήθως δεν είναι ικανοποιημένοι με την ολοκλήρωση του λογισμικού. Το σφάλμα στο λογισμικό είναι αναπόφευκτο και για τις δοκιμές του λογισμικού πρέπει να δαπανήσουν πολύ χρόνο. Το προσωπικό συντήρησης του λογισμικού δεν χρειάζεται μόνο να κατανοήσει τις επιχειρηματικές διαδικασίες, αλλά να είναι εξοικειωμένοι με την τεχνολογία ανάπτυξης λογισμικού, για να λύσουν ένα σφάλμα στο λογισμικό που μπορεί να εισάγει νέα σφάλματα και πολλοί άνθρωποι δεν είναι πρόθυμοι να συμμετάσχουν στην συντήρηση του λογισμικού. Αν το λογισμικό συχνά προκαλεί λάθη, θα προκαλέσει τη δυσαρέσκεια των χρηστών και τελικά θα εγκαταλειφθεί.
3) Το προσωπικό ανάπτυξης λογισμικού δεν μπορεί να ανταποκριθεί στις ανάγκες της αγοράς, που πρέπει να κυριαρχήσουν με πάρα πολύ γνώση. Η ανάπτυξη του λογισμικού απαιτεί κυρίως: τεχνολογία βάσης δεδομένων, τεχνολογία πρόσβασης στη βάση δεδομένων και μια ποικιλία των συστημάτων διαχείρισης βάσεων δεδομένων, των δικτύων και της τεχνολογίας του Διαδικτύου, τα εργαλεία ανάπτυξης, τις σχετικές δομές, τις βιβλιοθήκες των κλάσεων, τις δοκιμές του λογισμικού, τα εργαλεία ανάλυσης και πολλές άλλες τεχνολογίες. Δεδομένης της επιτάχυνσης της τεχνολογικής αναβάθμισης, οι προγραμματιστές μαθαίνουν συνεχώς νέες τεχνικές, ενώ είναι απασχολημένοι δουλεύοντας.
4) Η συχνή ροή των προγραμματιστών λογισμικού έχει μια μεγάλη απώλεια για την εταιρεία. Το λογισμικό είναι τα πνευματικά προϊόντα των προγραμματιστών, ο καθένας έχει τις δικές του προγραμματιστικές συνήθειες, τις ιδέες και τις μεθόδους. Είναι πολύ δύσκολο να διαβάσουν το πρόγραμμα κάποιου άλλου. Αν οι

προγραμματιστές αποχωρήσουν από το σχέδιο στα μισά της ανάπτυξης, οι άλλοι θα είναι δύσκολο να το αναλάβουν.
5) Το λογισμικό που δεν έχει την κατάλληλη τεκμηρίωση, προκαλεί μεγάλες δυσκολίες στο τελευταίο μέρος της ανάπτυξης, τη συντήρηση και την ανακατασκευή. Στην ανάπτυξη του λογισμικού θα πρέπει πρώτα να συνταχθεί μια τεκμηρίωση, στη συνέχεια να συνταχθεί το πρόγραμμα και να τροποποιηθεί το πρόγραμμα μετά την τροποποίηση της τεκμηρίωσης. Μερικοί προγραμματιστές απασχολούνται για να συντάξουν ή να τροποποιήσουν το πρόγραμμα και ξεχνάνε να τροποποιήσουν την τεκμηρίωση, με αποτέλεσμα το πρόγραμμα να έρχεται σε αντίθεση με την τεκμηρίωση και τις διαδικασίες.

Τα άρθρα (Watson, 2009, Berglund, 2003, Rupakheti και Hou 2011, Wu et al., 2010, Hou και Yao, 2011, Dekel και Herbsleb, 2009) ασχολούνται με τα προβλήματα που αφορούν θέματα τεκμηρίωσης των APIs. Στο άρθρο (Watson, 2009) παρουσιάζονται οι συνέπειες των κακοσχεδιασμένων APIs όσον αφορά τη μειωμένη αξιοπιστία του τελικού προϊόντος και τη μειωμένη παραγωγικότητα του προγραμματιστή χρησιμοποιώντας το API. Οι προγραμματιστές περιμένουν το API για να τους πει τι πρέπει να ξέρουν έτσι ώστε να μην χρειάζεται να το αναζητήσουν. Ενώ το ίδιο το API είναι η πρώτη πηγή της τεκμηρίωσης που συναντά ένας προγραμματιστής, ενισχύει την ανάγκη για αποτελεσματική τεκμηρίωση αναφοράς, όταν λέει ότι από τη στιγμή που ο προγραμματιστής ζητά τη βοήθεια του, έχουν ήδη αποτύχει με πολλές άλλες πηγές τεκμηρίωσης όταν οι προγραμματιστές έχουν την ατυχία να φτάσουν σε εκείνο το σημείο. Χρησιμοποιώντας διαφορετικούς όρους για την ίδια έννοια αυξάνει επίσης το κόστος εντοπισμού και η προκύπτουσα τεκμηρίωση μπορεί να μην είναι τόσο σαφής.

Στο άρθρο (Berglund, 2003) οι βιβλιοθήκες αυξάνουν το ποσό της συλλογής πληροφοριών και εκμάθησης - ανάγνωσης που απαιτούνται από τους προγραμματιστές. Αυτό προέρχεται από το πρόβλημα της γνωστικής απόστασης, δηλαδή η πνευματική προσπάθεια που απαιτείται από τους προγραμματιστές να χρησιμοποιούν βιβλιοθήκες για την ανάπτυξη. Ο χρόνος που χρειάζεται για να συλλέξει τη γνώση, πώς να χρησιμοποιήσει ξανά μια κλάση και τη σύνταξη, όπως η υπογραφή της μεθόδου, από την τεκμηρίωση, είναι ένα κόστος στην ανάπτυξη

λογισμικού. Οι προσωρινές διαστάσεις των πληροφοριών στο διαδίκτυο μερικές φορές θεωρούνται ως ένα μειονέκτημα. Η ταχύτητα με την οποία οι βιβλιοθήκες αλλάζουν, αυξάνονται και πολλαπλασιάζονται κάνοντας την τεκμηρίωση ασταθή. Γράφοντας την τεκμηρίωση πριν από την κυκλοφορία των βιβλιοθηκών μπορεί, στην πραγματικότητα να μην είναι αναγκαία σε ένα παγκόσμιο δικτυωμένο περιβάλλον ανάπτυξης.

Το άρθρο (Rupakheti και Hou 2011) αναφέρει ότι είναι αρκετά χρονοβόρο για να βρεθεί σχετική βοήθεια από την πληθώρα των πληροφοριών στον παγκόσμιο ιστό. Οι βιβλιοθήκες λογισμικού, τα πλαίσια, και τα APIs τείνουν να γίνουν πιο περίπλοκα καθώς ωριμάζουν με την πάροδο του χρόνου. Οι χρήστες του πλαισίου πρέπει να γνωρίζουν και να ακολουθήσουν αυτούς τους πολυάριθμους κανόνες, προκειμένου να ολοκληρώσουν μια εργασία προγραμματισμού. Σε γενικές γραμμές, η επαναχρησιμοποίηση δεν μπορεί να χρησιμοποιηθεί με τον καλύτερο δυνατό τρόπο όπως σχεδιάστηκε αρχικά. Ένας προγραμματιστής μπορεί να κάνει λάθη κατά την εκτέλεση μιας πολύπλοκης διαδικασίας λύσης. Ενώ τα συστήματα που βασίζονται στην αναζήτηση, μπορεί να αποφέρουν πολύτιμα παραδείγματα και τεκμηρίωση για τον προγραμματιστή, μπορεί να έχουν ακόμη τις δυσκολίες στη διαμόρφωση των σωστών ερωτήσεων σε πρώτο πλάνο ή στην κατανόηση και την προσαρμογή των παραδειγμάτων στο έργο. Τέτοια προβλήματα είναι συχνά συνέπεια της έλλειψης κατανόησης των εσωτερικών εργασιών του πλαισίου και των APIs στην πλευρά των προγραμματιστών, καθώς και άλλες δυσκολίες που σχετίζονται με τις ανάγκες πληροφόρησης των προγραμματιστών.

Στο άρθρο (Wu et al., 2010) η τρέχουσα τεκμηρίωση του API οργανώνεται με γενικές κατευθυντήριες γραμμές που δεν είναι για κάθε συγκεκριμένη χρήση. Οι προγραμματιστές πρέπει να διασχίζουν μη αυτόματα όλα τα έγγραφα API για να επιβεβαιώσουν την χρηστικότητα. Αυτή είναι χρονοβόρα και επιρρεπή σε λάθη διαδικασία για τον σωστό προγραμματισμό με API. Ωστόσο, η χρήση API με παραδείγματα κώδικα εξακολουθεί να είναι επίπονη για τους προγραμματιστές. Πρώτον, κατά τη διάρκεια της διαδικασίας επιλογής παραδειγμάτων κώδικα, δεν υπάρχει υποστήριξη για την παροχή ενός περιγράμματος παραδείγματος κώδικα για τους προγραμματιστές για να εκτιμήσουν αποτελεσματικά την καταλληλότητα του επιλεγμένου έργου για τον προγραμματισμό του. Δεύτερον, ένα παράδειγμα

κώδικα δείχνει μόνο τη χρήση του API σε ένα συγκεκριμένο πλαίσιο. Για την προσαρμογή του API σε πλαίσιο προγραμματισμού, οι προγραμματιστές πρέπει να διασχίσουν ολόκληρα τα API έγγραφα για τις απαραίτητες πληροφορίες. Δύο ανάγκες οι οποίες εξακολουθούν να υπάρχουν είναι η έλλειψη του εργαλείου υποστήριξης για τους προγραμματιστές. Οι προγραμματιστές πρέπει να βοηθήσουν στην επιλογή των σχετικών με αυτά παραδείγματα κώδικα και οι προγραμματιστές χρειάζονται βοήθεια για την εκπλήρωση της αποστολής της εργασίας προγραμματισμού με βάση τα επιλεγμένα παραδείγματα κώδικα.

Τα μειονεκτήματα που μελετά αυτό το άρθρο (Hou και Yao, 2011) είναι ότι η μείωση της ζεύξης, η ακατάλληλη σύζευξη μπορεί να κάνει τη κατανόηση και τη συντήρηση δύσκολη χωρίς λόγο. Η ενθυλάκωση: Μια μέθοδος API μπορεί να καταργηθεί, διότι θεωρείται ότι εκθέτει πάρα πολύ εσωτερική πληροφορία που θα ήταν καλύτερο να ενθυλακωθεί. Η συμμόρφωση με τις συμβάσεις υπερτύπου: Η πρόθεση για μια μέθοδο σε μια υποκατηγορία για να υπερβεί μια μέθοδο υπερτύπου, αυτή εκφράζεται από την κατοχή των δύο μεθόδων που μοιράζονται την ίδια υπογραφή. Ο επανασχεδιασμός των υπαρχόντων χαρακτηριστικών: Μερικές μέθοδοι API χαρακτηρίστηκαν ως παρωχημένες, ως αποτέλεσμα μιας ευρύτερης προσπάθειας αναδιάρθρωσης και επανασχεδιασμού. Η λειτουργική αναβάθμιση των υπαρχόντων χαρακτηριστικών: Όταν ένα υπάρχον χαρακτηριστικό δεν είναι αρκετά ισχυρό, θα είναι αδύνατο ή δύσκολο να επιτευχθεί κάποια λειτουργικότητα. Όταν οι API μέθοδοι μεταβλήθηκαν κατά τέτοιο τρόπο ώστε ένα πρόγραμμα-πελάτη είναι σπασμένο, μπορεί να είναι κουραστικό να φέρει το πρόγραμμα-πελάτη πίσω για να λειτουργήσει με το επικαιροποιημένο API.

Στο (Dekel και Herbsleb, 2009) τα σύγχρονα συστήματα λογισμικού συνδυάζουν κώδικα που γράφτηκε από πολλά άτομα και κάνουν χρήση των εξωτερικών βιβλιοθηκών και των APIs. Οι ενδιαφερόμενοι σε αυτές τις ρυθμίσεις δεν είναι πιθανό να είναι πλήρως εξοικειωμένοι με όλες τις τρέχουσες γνώσεις σχετικά με τα αντικείμενα και τις υπηρεσίες στο έργο και τον κώδικα τρίτων. Η έλλειψη ενημέρωσης από τις οδηγίες χρήσης και οι προειδοποιήσεις μπορούν να οδηγήσουν σε αποτυχίες εκτέλεσης και τις δυσκολίες συντήρησης. Οι ενδεχόμενοι καταναλωτές της εν λόγω τεκμηρίωσης περνούν το μεγαλύτερο μέρος του χρόνου

τους στην περιήγηση κώδικα το οποίο περιλαμβάνει πολλές κλήσεις μεθόδου. Συχνά οι προγραμματιστές μαθαίνουν νέα APIs από παραδείγματα κώδικα, τα ευρήματα έχουν επίσης επιπτώσεις για τις τρέχουσες πρακτικές εκμάθησης. Ωστόσο, δεδομένου ότι ο δυναμικός τύπος μιας μεταβλητής δεν μπορεί να προβλεφθεί και μπορεί να αλλάξει κατά τη διάρκεια της εκτέλεσης, ο πραγματικός στόχος των εν λόγω κλήσεων δεν μπορεί να προσδιοριστεί. Για το λόγο αυτό, οι περισσότεροι IDEs παρουσιάζουν απλώς τα JavaDocs για το στατικό στόχο, ενδεχομένως αφήνοντας τους χρήστες αγνοώντας νέες ή αντικρουόμενες οδηγίες σε μια έκδοση υπέρβασης. Αυτό μπορεί να έχει σοβαρές συνέπειες αν η συμμόρφωση παραβιάζεται και η τεκμηριωμένη συμπεριφορά των εκδόσεων υπέρβασης συγκρούεται με αυτή που υπερβαίνεται. Ορισμένα KIs μπορεί να είναι παρωχημένα, λανθασμένα, ή να είναι δύσκολο να ερμηνευθούν. Ένας σχετικός περιορισμός ήταν ότι τα άτομα δεν ήταν προηγουμένως εξοικειωμένα με τα APIs και τη βάση κώδικα.

Τα δυο άρθρα (Robillard, 2009, Hou και Li, 2011) διερευνούν τις δυσκολίες εκμάθησης των APIs. Στο (Robillard, 2009) τα παραδείγματα του κώδικα μπορούν να γίνουν μάλλον εμπόδιο, όταν υπάρχει αναντιστοιχία μεταξύ του στόχου του παραδείγματος και του στόχου του παραδείγματος του χρήστη. Κάποιες αποφάσεις για το σχεδιασμό μπορούν να επηρεάσουν τη συμπεριφορά του API με τους λεπτούς τρόπους που συγχέουν οι προγραμματιστές. Οι προσπάθειες για τη βελτίωση της χρηστικότητας της δομής ενός API χρειάζεται να συμπληρωθεί από τις προσπάθειες για να βελτιωθούν οι διαθέσιμοι πόροι για να τους μάθουν. Οι διαθέσιμοι πόροι για την εκμάθηση ενός API είναι σημαντικοί και οι ελλείψεις στον τομέα αυτό εμποδίζουν την πρόοδο εκμάθησης του API. Ένα εμπόδιο για τον προγραμματιστή ήταν να καταλάβει ότι υπάρχουν πολλαπλοί τρόποι για να κάνει κάτι. Ως εμπόδιο για την εκμάθηση του API πολλοί ερωτηθέντες προσδιόρισαν την απουσία του API σε παραδείγματα χρήσης προσαρμοσμένα στις ανάγκες τους. Ένα άλλο σημαντικό εμπόδιο είναι η αναξιοπιστία της πηγής των παραδειγμάτων. Οι παρατηρήσεις αυτές έχουν συνέπειες για τους χρήστες του API, τους σχεδιαστές, τους συγγραφείς της τεκμηρίωσης και των κατασκευαστών του εργαλείου ανάπτυξης. Για παράδειγμα, εάν η συμπεριφορά του API φαίνεται ανεξήγητη, η απάντηση μπορεί να έχει να κάνει τόσο με το σχεδιασμό του όσο και με το χαμηλό επίπεδο της δομής του.

Μια σειρά εμποδίων μελετούνται στο άρθρο (Hou και Li, 2011). Αυτά περιλαμβάνουν θέματα όπως η ανεπαρκής ή ελλιπής τεκμηρίωση για τις λεπτές πτυχές του σχεδιασμού, ακατάλληλές δομές σχεδιασμού και δυσκολία στην εκμάθηση. Α. Ρωτώντας για μια λύση. Το δύσκολο να βρει τη λύση συμβαίνει όταν η αντιστοίχιση από την τεκμηρίωση API σε μια επιθυμητή λύση είναι η λιγότερο προφανής. Οι προγραμματιστές μπορούν να ζητήσουν ακόμη τέτοιες ερωτήσεις, ακόμη και μετά την ανάγνωση της τεκμηρίωσης API. Υπάρχουν επίσης APIs των οποίων οι λειτουργίες είναι μόνο ανεπαρκείς ή καθόλου, τεκμηριωμένες.

Β. Χρησιμοποιώντας μια ανακριβή λύση. Κάποια APIs μπορούν να μοιραστούν τόσο την ομοιότητα της επιφάνειας που οι προγραμματιστές μπορεί να μην είναι σε θέση να διακρίνουν εύκολα τους ρόλους αυτών των APIs. Ως εκ τούτου, δεν ξέρουν πότε να τους χρησιμοποιήσουν. Μερικές φορές μια φαινομενικά εύκολη, αλλά ακατάλληλη λύση θα μπορούσε να επιλεγεί γιατί η σωστή δεν ήταν προφανής ή πολύ δύσκολο να χρησιμοποιηθεί.

Γ. Χρησιμοποιώντας μια λύση λανθασμένα. Πρώτον, ο προγραμματιστής μπορεί να αγνοεί την λεπτή εξάρτηση μεταξύ δύο APIs, ή το αποτέλεσμα της προϋπόθεσης στο API. Ο δεύτερος λόγος αυτής της κατηγορίας εκτελεί ένα σχέδιο πολλαπλών λανθασμένων βημάτων ή ελλιπή (ελλιπής ακολουθίες API). Τρίτον, η σημασιολογία ορισμένων API δεν είναι σαφώς τεκμηριωμένη και ως εκ τούτου, μπορεί να προκαλέσει σύγχυση στους προγραμματιστές. Υπάρχουν δύο κύριοι λόγοι για τους οποίους η πλατφόρμα-συγκεκριμένων περιπτώσεων, δεν ήταν τεκμηριωμένη. Ο ένας είναι ότι σε ορισμένες περιπτώσεις απαιτούν την προσαρμογή των πτυχών του βασικού πλαισίου, που δεν έχει ακόμα δημοσιευθεί ως API. Ο άλλος οφείλεται στην εξέλιξη του πλαισίου. Όταν ένα πλαίσιο εξελίσσεται, οι προηγούμενοι τρόποι χρήσης μπορεί να καταστούν ξεπερασμένοι και να πρέπει να αντικατασταθούν από νέους τρόπους. Υπάρχουν τρία προβλήματα με την τεκμηρίωση. Ορισμένες λύσεις ήταν ατεκμηρίωτες. Για πολλές άλλες περιπτώσεις, διαπιστώθηκε ότι η τεκμηρίωση περιείχε τις αναγκαίες πληροφορίες, αλλά φάνηκε ότι οι προγραμματιστές είχαν δυσκολία στη πρόσβαση των λύσεων. Αυτό είναι πολύ πιθανό, επειδή οι αναγκαίες πληροφορίες, ήταν είτε βαθιά ριζωμένες στο μακροσκελές κείμενο ή διασκορπισμένες σε διάφορες τοποθεσίες.

Στο (Kernebeck, 1997) ο κώδικας επαναχρησιμοποίησης μόνος του είναι άχρηστος, εάν η ποιότητα του κώδικα δεν μπορεί να αποδειχθεί με τη βοήθεια των απαιτήσεων με βάση τα έγγραφα ελέγχου, τα αρχεία αξιολόγησης και τις διαδικασίες δοκιμών. Πολλά επιθυμητά χαρακτηριστικά, όπως η φιλικότητα προς τον χρήστη, η φορητότητα ή επαναχρησιμοποίηση δεν μπορούν να μετρηθούν άμεσα. Η επαναχρησιμοποίηση ενός ενδεχόμενου στοιχείου του λογισμικού μπορεί να προσδιοριστεί χρησιμοποιώντας στοιχεία σχετικά με τη γλώσσα προγραμματισμού και τη διαθεσιμότητα των μεταφραστών της γλώσσας του προγράμματος. Η επιθυμητή διεπαφή, που περιγράφεται στη γλώσσα προγραμματισμού, πρέπει να συγκριθεί με τη διεπαφή του στοιχείου λογισμικού. Εάν η σύγκριση αυτή δεν οδηγήσει σε αποκλίσεις, δεν υπάρχει προσπάθεια επαναχρησιμοποίησης. Ένα άλλο μειονέκτημα είναι ότι οι οριακές τιμές εξαρτώνται από το προσωπικό του λογισμικού και τον τομέα εφαρμογής.

Στο τελευταίο άρθρο αυτής της κατηγορίας παρουσιάζονται τα μειονεκτήματα της μεθόδου του εννοιολογικού χάρτη. Στο (Gerken et al., 2011) γράφοντας ένα κομμάτι του κώδικα είναι συχνά μια επίπονη διαδικασία. Επιπλέον, χρησιμοποιώντας ένα API είναι μια συνεχής διαδικασία μάθησης, όμως οι προγραμματιστές διαβάζουν σπάνια την τεκμηρίωση εκ των προτέρων, αλλά μάλλον αναζητούν για παραδείγματα ή τεκμηρίωση στον αέρα. Με αυτόν τον τρόπο, μια μέθοδος έρευνας για τη χρηστικότητα του API πρέπει να είναι σε θέση να κατανοήσει αυτή την μαθησιακή διαδικασία στη πάροδο του χρόνου και να επιτρέψει στον ερευνητή τον εντοπισμό των εμποδίων εκμάθησης. Εμπόδια εκμάθησης ενός API, όπως εμπόδια επιλογής ή εμπόδια πληροφοριών, σε ένα μεγάλο πεδίο μελέτης μπορούν να χρησιμοποιηθούν ξανά σε μια ομάδα ποιοτικών δεδομένων. Ο εντοπισμός αυτών των εμποδίων εκμάθησης μπορεί να είναι ένα βήμα για να εκτιμηθεί το όριο ενός API, το οποίο βασικά σημαίνει πόσο δύσκολο είναι να επιτύχει ορισμένα αποτελέσματα με αυτό. Οι αλλαγές είναι κουραστικές και οι συμμετέχοντες είναι απρόθυμοι να τις κάνουν.

3.2.4 Μελέτη των μειονεκτημάτων του τομέα γενικά ζητήματα των APIs

Στη τέταρτη κατηγορία μελετάμε τα μειονεκτήματα των άρθρων γενικού περιεχομένου όσον αφορά τα APIs. Πιο συγκεκριμένα, τα άρθρα (WENDYKIER και NAGY, 2010, Koehler et al., 2008, Dezso et al., 2011, Johnsson και Mathur,

2005) αναφέρονται στα προβλήματα των πολυδιάστατων πινάκων και των επιστημονικών υπολογισμών, στα παράλληλα εργαλεία ανάλυσης των αποδόσεων, καθώς επίσης στους παράλληλους επεξεργαστές, στη χρήση γραφημάτων και αλγόριθμων δικτύου.

Στο (WENDYKIER και NAGY, 2010) τα μειονεκτήματα της χρήσης της Java σε επιστημονικό προγραμματισμό είναι ότι δεν περιλαμβάνουν κανένα στοιχειώδη τύπο για μιγαδικούς αριθμούς, μια ανικανότητα να κάνουν υπερφόρτωση τελεστή και δεν υπάρχει υποστήριξη για IEEE εκτεταμένης ακρίβειας. Επιπλέον, στη Java οι πίνακες δεν έχουν σχεδιαστεί για υψηλής απόδοσης υπολογισμό. Επιπλέον στην Java οι πίνακες δεν έχουν δυνατότητα αλλαγής μεγέθους και μόνο σε 32-bit πίνακες είναι δυνατόν. Στο Colt ένας ενιαίος συνεχόμενος μονοδιάστατος πίνακας Java χρησιμοποιείται για να αποθηκεύσει τα στοιχεία όλων των πυκνών 2D και 3D πινάκων, υπάρχουν δύο προβλήματα με αυτή την προσέγγιση. Πρώτον, οι αλγόριθμοι αποσύνθεσης του πίνακα τυπικά αναμένουν πινάκες εισόδου για να χρησιμοποιήσουν τη σειρά κύριας στήλης. Δεύτερον, επειδή στη Java οι δείκτες του πίνακα πρέπει να είναι 32-bit ακέραιες τιμές, ο 1D πίνακας δεν μπορεί να περιέχει περισσότερα από 2^{31} στοιχεία, που αποτελεί σημαντικό περιορισμό για μεγάλης κλίμακας προβλήματα. Η Java δεν συμμορφώνεται πλήρως με το πρότυπο IEEE 754, δεδομένου ότι δεν υποστηρίζει τις σημαίες για IEEE 754 εξαιρέσεις, άκυρες λειτουργίες, υπερχείλιση, διαίρεση με το μηδέν, ανεπάρκεια, ανακριβή αποτελέσματα δεν υπάρχει περίπτωση να παρουσιαστεί όταν η τιμή ενός αριθμού κινητής υποδιαστολής γίνεται είτε στο άπειρο ή NaN.

Στο άρθρο (Koehler et al., 2008) η συμπεριφορά μιας RC εφαρμογής μπορεί να είναι ιδιαίτερα δύσκολη να παρατηρηθεί και να κατανοηθεί λόγω πρόσθετων επιπέδων παραλληλισμού και πολύπλοκων αλληλεπιδράσεων μεταξύ των ετερογενών πόρων σε τέτοιου είδους συστήματα. Οι τεχνικές εντοπισμού σφαλμάτων πρέπει να σταματήσουν την εφαρμογή FPGA για να ανακτήσουν τα δεδομένα. Δυστυχώς, η τεχνική αυτή απομονώνει αποτελεσματικά το FPGA από το υπόλοιπο σύστημα, το οποίο συνήθως δεν μπορεί να διακοπεί. Ενώ η απομόνωση ενθαρρύνεται στην αποσφαλμάτωση, είναι εξαιρετικά προβληματική η ανάλυση των επιδόσεων από το στοιχείο αλληλεπίδρασης του συστήματος. Ωστόσο, λόγω του μεγάλου ύψους του παραλληλισμού είναι δυνατόν σε ένα

FPGA, την παρακολούθηση όλων των on-chip επικοινωνία, μπορεί να θεωρηθεί σημαντική επιβάρυνση. Ο έλεγχος μπορεί να γίνει εμπόδιο, όταν πάρα πολλοί κύκλοι χρησιμοποιούνται για την εγκατάσταση, την ολοκλήρωση. Η δυαδική οργάνωση για FPGAs είναι πολύ δύσκολή. Επιπλέον, η δυαδική οργάνωση χάνει κάποια ευελιξία από τη σύνθεση και η εφαρμογή μπορεί να έχει αλλάξει σημαντικά ή να εξαλειφθούν ορισμένα δεδομένα κατά τη διάρκεια της βελτιστοποίησης ή να καθίστανται μερικά δεδομένα απροσπέλαστα μέσω της δρομολόγησης FPGA. Μια άλλη σημαντική διαφορά αφορά την περιορισμένη διαθεσιμότητα μνήμης σε ένα FPGA. Τα δεδομένα ανίχνευσης μπορούν εύκολα να εξαντλούν τους πόρους του μπλοκ της μνήμης RAM, προκαλώντας έλλειψη πόρων και για τη κατανομή και την ανίχνευση.

Το (Dezso et al., 2011) στο LEMON, αρκετές βιβλιοθήκες αποθηκεύουν συναφή δεδομένα απευθείας στα αντικείμενα στον κόμβο και αντικείμενα τόξου των γραφημάτων. Ο σχεδιασμός αυτός επιτρέπει τη διευκόλυνση της υλοποίησης, αλλά έντονα περιορίζει την ευελιξία της βιβλιοθήκης. Το LEMON παρέχει επίσης λύσεις που βασίζονται στον επισκέπτη, αλλά μόνο για τους βασικούς αλγόριθμους αναζήτησης γραφήματος BFS, και DFS. Το LEDA παρέχει λιγότερη ευελιξία στη χρήση αλγορίθμων από ότι οι άλλες δύο βιβλιοθήκες. Τρία βασικά προβλήματα στο πλαίσιο των δοκιμών είναι α) η εύρεση των συντομότερων μονοπατιών από ένα καθορισμένο κόμβο πηγή σε ένα γράφημα με μη αρνητικά μήκη τόξου β) η εξεύρεση μιας μέγιστης ροής μεταξύ δύο κόμβων σε ένα δίκτυο με χωρητικότητες τόξου γ) την εξεύρεση μιας ελάχιστης ροής κόστους από ένα σύνολο των κόμβων της προσφοράς σε ένα σύνολο των κόμβων της ζήτησης σε ένα δίκτυο με περιορισμούς χωρητικότητας και του κόστους του τόξου.

Στο (Johnsson και Mathur, 2005) μερικές παράλληλες μέθοδοι δεν έχουν καλή αριθμητική συμπεριφορά ως ακολουθία των μεθόδων και αυτό το μειονέκτημα συχνά αυξάνεται με το βαθμό του παραλληλισμού. Η ταχύτητα των chip μνήμης παρουσιάζουν το πιο αυστηρό περιορισμό όσον αφορά την απόδοση. Το δεύτερο πιο αδύναμο τεχνολογικό στοιχείο είναι το σύστημα επικοινωνίας. Ο μονοδιάστατος χώρος διευθύνσεων που χρησιμοποιείται για τις συμβατικές γλώσσες δεν είναι κατάλληλος για τις περισσότερες εφαρμογές για τα MPPs. Η

γραμμικότητα του χώρου διευθύνσεων μπορεί επίσης να οδηγήσει σε κακή απόδοση για πολυδιάστατους πίνακες ή παρεμβολή στα συστήματα μνήμης.

Τέσσερα άρθρα αυτής της κατηγορίας (Ferreira et al., 2007, Unalir et al., 2000, Inoue et al., 2010, Sabou, 2004) ασχολούνται με τα μειονεκτήματα που αφορούν το κατανεμημένο περιβάλλον, τους πόρους, το HTTP πρωτόκολλο, θέματα προσομοίωσης, την αρχιτεκτονική client – server καθώς και με διαδικτυακές υπηρεσίες οι οποίες περιγράφονται από μια οντολογία.

Στο άρθρο (Ferreira et al., 2007) οι εφαρμογές παιχνιδιών κινητής τηλεφωνίας αντιμετωπίζουν προβλήματα που είναι διαφορετικά από τις σταθερές εφαρμογές του δικτύου. Τα θέματα αυτά περιλαμβάνουν την κυμαινόμενη συνδεσιμότητα, την ποιότητα των υπηρεσιών και την κινητικότητα της υποδοχής όπου δεν υπάρχει εξειδικευμένη λύση ενδιάμεσου δικτύου για μεγάλης κλίμακας κινητά και διάχυτα παιχνίδια αυξημένης πραγματικότητας.

Οι λόγοι για τις αποτυχίες στο (Unalir et al., 2000), περιλαμβάνουν περιορισμένη προσβασιμότητα και την επεκτασιμότητα του χώρου αποθήκευσης, αποκλειστικό έλεγχο καταχωρημένων στοιχείων και φτωχή οικονομία κλίμακας. Ο χρόνος επεκτασιμότητας της βιβλιοθήκης επαναχρησιμοποίησης μπορεί να ολοκληρωθεί, αλλά η διεπαφή χρήσης του FedeRaL δεν έχει ακόμη υλοποιηθεί. Επίσης, υπάρχουν κάποια προβλήματα, ενώ προσαρμόζονται στην τεχνολογία πολύπλευρης ταξινόμησης OLAP. Υπάρχει μια τεράστια ποσότητα των επαναχρησιμοποιούμενων στοιχείων, αλλά μια μεγάλη έλλειψη βιώσιμων υποδομών και των συνολικών μηχανισμών αναζήτησης. Η υποδομή των βιβλιοθηκών επαναχρησιμοποίησης διαφέρουν μεταξύ τους σε αυτό το δυναμικό και ανοιχτό σύνολο πληροφοριών. Οι βιβλιοθήκες επαναχρησιμοποίησης είναι η έλλειψη της υποστήριξης της διαχείρισης διανεμημένων επεκτάσιμων χώρων αποθήκευσης και η δυναμική διαλειτουργικότητα μεταξύ τους. Ένα κοινό πρόβλημα που αντιμετωπίζουν πολλές βιβλιοθήκες επαναχρησιμοποίησης σήμερα είναι η ενιαία και κλιμακούμενη πρόσβαση των πολλαπλών χώρων αποθήκευσης.

Στο (Inoue et al., 2010) οι προγραμματιστές συνεχίζουν να αναγκάζονται να αναπτύσσουν τα δικά τους δεδομένα αποθήκευσης. Ορισμένα δεδομένα

αποθήκευσης είναι εφοδιασμένα με Atom που έχουν αναπτυχθεί αλλά δεν παρέχουν ούτε αποτελεσματικό έλεγχο της πρόσβασης, ούτε εύκολες μεθόδους επέκτασης. Η έλλειψη του έλεγχου πρόσβασης παράγει αναποτελεσματικό φιλτράρισμα των εξωτερικών στοιχείων αποθήκευσης έναντι των αποτελεσμάτων αναζήτησης και η έλλειψη μιας μεθόδου επέκτασης αποτρέπει τους προγραμματιστές από την προσθήκη επιπλέον λειτουργικοτήτων. Ως αποτέλεσμα, οι αποθηκεύσεις Atom σπάνια χρησιμοποιούνται, αυξάνοντας έτσι το κόστος ανάπτυξης των εφαρμογών με τα APIs στον Ιστό. Λόγω του αναποτελεσματικού ελέγχου πρόσβασης, το WAPDB δεν μπορεί να χρησιμοποιηθεί για τη διαχείριση λογαριασμού, λόγω των περιορισμών που τίθενται στις συναλλαγές. Αυτό σημαίνει ότι οι λογαριασμοί χρηστών διαχειρίζονται από ένα παραδοσιακό RDBMS, ενώ οι πόροι Web είναι αποθηκευμένοι σε WAPDB.

Στο (Sabou, 2004) η δημιουργία των οντολογιών είναι δύσκολη και δαπανηρή, εμποδίζοντας έτσι την εξάπλωση των υπηρεσιών Ιστού. Η απόκτηση των σημασιολογικών περιγραφών υπηρεσιών Ιστού είναι μια χρονοβόρα και πολύπλοκη εργασία των οποίων η αυτοματοποίηση είναι επιθυμητή. Η προσέγγιση στο πρόβλημα της δημιουργίας ποιοτικών οντολογιών παρακινείται από την παρατήρηση ότι, δεδομένου ότι οι υπηρεσίες Ιστού είναι απλά ανοίγματα του υπάρχοντος λογισμικού στην προσβασιμότητα Ιστού, υπάρχει μεγάλη επικάλυψη μεταξύ της λειτουργικότητας που προσφέρεται από μια διαδικτυακή υπηρεσία και από την βασική της υλοποίηση. Η ακρίβεια σημασίας είναι σχεδόν διπλάσια από το κείμενο σε σχέση με το τμήμα παραμέτρου. Οι κώδικες είναι πάρα πολύ μικροί για την εφαρμογή των στατιστικών μεθόδων, αγνοούνται όλες οι υπάρχουσες πληροφορίες για τη δομή και ως εκ τούτου χάνουν πολλά από τη σημασιολογία.

Τα μειονεκτήματα των ασαφών συστημάτων απεικόνισης παρουσιάζει το άρθρο (Pham και Brown, 2005). Η πολυπλοκότητα που προκύπτει από τις πληροφορίες αβεβαιότητας καθιστά πιο δύσκολο για έναν άνθρωπο να καταλάβει τον τρόπο λειτουργίας αυτών των συστημάτων, ιδιαίτερα πώς να ερμηνεύσει την επίπτωση της ασάφειας της κάθε μεταβλητής για την αλληλεπίδρασή της με άλλες μεταβλητές και πώς η διάδοση της εν λόγω ανακρίβειας επηρεάζει το επίπεδο της εμπιστοσύνης των αποτελεσμάτων σε κάθε στάδιο. Επιπλέον, οι μέθοδοι

απεικόνισης συχνά εστιάζουν σε σύνολα δεδομένων και μόνο χαλαρά συνδεδεμένα με την αναλυτική διαδικασία. Για έναν άπειρο χρήστη, αυτό μπορεί να σημαίνει πολλές δοκιμές και σφάλματα προσπαθειών για να προσδιοριστεί ο καλύτερος τρόπος για να αποκτήσει γνώσεις σε συγκεκριμένες εργασίες. Μια άλλη τεχνική είναι να προβάλει τα στοιχεία για τη μείωση των διαστάσεων και θα εμφανίσει τα αποτελέσματα σε μια γραφική παράσταση. Ωστόσο, παρότι η τεχνική αυτή παρέχει ένα υψηλό επίπεδο ανάλυσης από τα πιο σημαντικά στοιχεία των δεδομένων, έχει ένα μειονέκτημα λόγω της απώλειας των πληροφοριών κατά τη διάρκεια της διαδικασίας.

Στο (Xie και Pei, 2006) οι προγραμματιστές μπορούν επίσης να χρησιμοποιήσουν μια μηχανή αναζήτησης πηγαίου κώδικα για την αναζήτηση ανοιχτών πηγών αποθήκευσης για τα πηγαία αρχεία που χρησιμοποιούν τα ίδια APIs. Παρ 'όλα αυτά, ο αριθμός των επιστρεφόμενων πηγαίων αρχείων είναι συχνά μεγάλος. Είναι δύσκολο για τους προγραμματιστές να μάθουν τις χρήσεις του API από ένα μεγάλο αριθμό των επιστρεφόμενων αποτελεσμάτων. Αυτά τα APIs, που είναι εξοπλισμένα μόνο με απλά έγγραφα API, ωστόσο είναι συχνά περίπλοκα και όχι επαρκώς τεκμηριωμένα. Το έγγραφο API δεν παρέχει επαρκείς πληροφορίες για να μάθουμε πώς να χρησιμοποιούμε το API. Μπορούν να χρησιμοποιηθούν κάποιες μηχανές αναζήτησης κώδικα, ωστόσο τα πολυάριθμα και εσφαλμένα ταξινομημένα αποτελέσματα που επιστρέφονται από τις μηχανές αναζήτησης πηγαίου κώδικα δεν μπορούν γρήγορα και περιεκτικά να βοηθήσουν να κατανοήσουμε τα κοινά μεταξύ αυτών των αρχείων προέλευσης.

Το τελευταίο άρθρο αυτής της κατηγορίας ερευνά τα μειονεκτήματα ενός υπολογιστή που έχει API μηχανισμό σύνδεσης σε σύγκριση με έναν υπολογιστή χωρίς μηχανισμό προστασίας, προκείμένου να ανιχνευθεί κακόβουλο λογισμικό. Στο άρθρο (Marhusin, et al., 2008) η μείωση της απόδοσης που προκαλείται από τα συστήματα παρακολούθησης κλήσεων θα μπορούσε να επηρεάσει δυσμενώς το μηχάνημα συνδέοντας τα API του λειτουργικού συστήματος (OS), θα μπορούσε να παρεμποδίσει την απόδοση του λειτουργικού συστήματος. Ανεξάρτητα από το αν ο υπολογιστής είχε απλή, προστασία από ιούς ή σύνδεση API, όλα τα προγράμματα πήραν περισσότερο χρόνο μέχρι την ενεργοποίηση του GUI. Ωστόσο, ένας υπολογιστής με πρόγραμμα προστασίας από ιούς και σύζευξη

προκάλεσε καθυστέρηση κατά τον πρώτο γύρο για να γίνει πολύ περισσότερη. Επίσης, σε όλες τις περιπτώσεις, το πρόγραμμα προστασίας από ιούς προκαλείται περισσότερο από ό,τι η καθυστέρηση συνδέοντας κατά τον πρώτο γύρο του πειράματος. Καταλήγουμε στο συμπέρασμα ότι ο υπολογιστής δεν εκτελείται πραγματικά καλά, όταν εκτελείται ένα πρόγραμμα κατά τον πρώτο γύρο μετά από επανεκκίνηση. Προφανώς ο υπολογιστής απαιτεί περισσότερο χρόνο για να ξεκινήσει το πρόγραμμα.

3.3 ΣΥΜΠΕΡΑΣΜΑΤΑ

Το κεφάλαιο αυτό περιέγραψε ένα πεδίο μελέτης, που ασχολήθηκε με την παρουσίαση των πλεονεκτημάτων και μειονεκτημάτων των APIs. Έγινε μια λεπτομερή ανάλυση και καταγραφή των πλεονεκτημάτων και μειονεκτημάτων των τεσσάρων ερευνητικών κατηγοριών, κάθε μιας ξεχωριστά, όπως προέκυψαν από την μελέτη των 47 άρθρων, όπου διερευνήθηκαν οι διασυνδέσεις προγραμματισμού εφαρμογών (APIs).

Βιβλιογραφία - Αναφορές

1. Αλατζιά Όλγα (2012), Πτυχιακή εργασία με θέμα: “Μελέτη Αξιολόγησης Χρήσης Προτύπων Σχεδίασης στις βιβλιοθήκες”,
2. Χαραλαμπίδου Σοφία (2010), Πτυχιακή εργασία με θέμα: “Εμπειρική μελέτη για τη χρήση προτύπων σχεδίασης σε παιχνίδια ανοιχτού λογισμικού”,
3. Apostolos Ampatzoglou, Ioannis Stamelos, Sofia Charalampidou, “Research State of the Art on GoF Design Patterns: A Systematic Mapping Study”, Elsevier Editorial System(tm) for Journal of Systems and Software Manuscript Draft
4. Brereton P., Kitchenham B., Budgen D., Turner M., Khalil M. (2007), “Lessons from applying the systematic literature review process within the software engineering domain”, *Journal of Systems and Software*, Elsevier, Vol. 80, No. 4, pp 571-583
5. Cleidson R. B. de Souza, David Redmiles, Li-Te Cheng, David Millen, John Patterson, “Sometimes You Need to See Through Walls - A Field Study of Application Programming Interfaces”, *Proceedings of the 2004 ACM conference on Computer supported cooperative work (CSCW ’04),* ACM, pp. 63 – 71, Chicago, Illinois, USA, 6 -10 November 2004
6. Dyba T., Dingsoyr T. (2008), “Empirical studies of agile software development: A systematic review”, *Information and Software Technology*, Elsevier, Vol 50, No 9-10, pp. 833-859
7. Kitchenham B. (2007), “Procedures for undertaking systematic literature reviews”, *Joint Technical Report*, Computer Science Department, Keele University.
8. Kitchenham B., Brereton O. P., Budgen D., Turner M., Bailey J., Linkman S. (2009), “Systematic literature reviews in software engineering – A systematic literature review”, *Information and Software Technology*, Elsevier, Vol 51, No 1, pp 7-15.

Διαδικτυακοί Τόποι

http://en.wikipedia.org/wiki/Api

http://wiki.netbeans.org/API_Design

ΠΑΡΑΡΤΗΜΑ Μελέτες που συμπεριλαμβάνονται στην ανασκόπηση της βιβλιογραφίας

[S01] Jeffrey Stylos, Benjamin Graf, Daniela K. Busse, Carsten Ziegler, Ralf Ehret and Jan Karstens, "A Case Study of API Redesign for Improved Usability", IEEE *Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, pp.189-192, Herrsching am Ammersee, 15-19 September 2008

[S02] Vivek S. Pai, Alan L. Cox, Vijay S. Pai and Willy Zwaenepoel, "A Flexible and Efficient Application Programming Interface (API) for a Customizable Proxy Cache", *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems (*USITS'03*)*, ACM, Vol. 4, pp. 13-13, Berkeley, CA, USA, 2003

[S03] Miryung Kim, Dongxiang Cai and Sunghun Kim, "An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution", *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*, ACM, Waikiki, Honolulu, Hi, USA, 21-28 May 2011

[S04] Yukari Matsuzawa and Nobuyuki Ikeda, "An Object-Oriented Reference Model with Frameworks and Libraries", *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS '98)*, ACM, pp. 346, Washington, DC, USA, 1998

[S05] Umer Farooq, Leon Welicki, and Dieter Zirkler, "API Usability Peer Reviews: A Method for Evaluating the Usability of Application Programming Interfaces", *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*, ACM, pp. 2327-2336, Atlanta, Georgia, USA, 10–15 April 2010

[S06] Santonu Sarkar, Girish Maskeri Rama and Avinash C. Kak, "API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization", *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, IEEE, VOL. 33, NO. 1, pp. 14-32, January 2007

[S07] Jeff H. Perkins, "Automatically Generating Refactorings to Support API Evolution", *Proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE '05)*, ACM, pp. 111 – 114, Lisbon, Portugal, 2005

[S08] Nikolaos D. Tselikas, Nikolaos L. Dellas, Eleftherios A. Koutsoloukas, Sofia H. Kapellaki, George N. Prezerakos and Iakovos S. Venieris, "Distributed service provision using open APIs-based middleware: ''OSA/Parlay vs. JAIN'' performance evaluation study", *Journal of Systems and Software*, Elsevier, Vol. 80, Issue 5, pp. 765–777, NY, USA, May 2007

[S09] Tao Xie and Jian Pei, "MAPO: Mining API Usages from Open Source Repositories", *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR '06)*, ACM, pp. 54–57, Shanghai, China, 22–23 May 2006

[S10] Daqing Hou and Lin Li, "Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions", *19th International Conference on Program Comprehension*, IEEE, pp. 91-100, Kingston, Canada, 22 - 24 June 2011

[S11] Francesco Bellotti, Riccardo Berta, Massimiliano Margarone and Alessandro De Gloria, "oDect: an RFID-based object detection API to support applications development on mobile devices", *Journal Software—Practice & Experience*, Vol. 38, Issue 12, pp. 1241-1259, NY, USA, October 2008

[S12] Manpreet Singh, Maximilian Ott, Ivan Seskar and Pandurang Kamat, "ORBIT Measurements Framework and Library (OML): Motivations, Design, Implementation, and Features", *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM'05*), IEEE, pp. 146-152, Washington, DC, USA, 2005

[S13] PIOTR WENDYKIER and JAMES G. NAGY, "Parallel Colt: A High-Performance Java Library for Scientific Computing and Image Processing", *Journal ACM Transactions on Mathematical Software (TOMS)*, ACM, Vol. 37, Issue 3, NY, USA, September 2010

[S14] Seth Koehler, John Curreri and Alan D. George, "Performance analysis challenges and framework for high-performance reconfigurable computing", *Journal Parallel Computing*, Elsevier, Vol. 34, Issue 4-5, pp. 217–230, Amsterdam, The Netherlands, May 2008

[S15] Pedro Ferreira, João Orvalho and Fernando Boavida, "Quality of Service APIs for a 3GPP Mobile and Pervasive Large Scale Augmented Reality Gaming Middleware", *International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST 2007)*, IEEE, pp.179-184, Cardiff, Wales, UK, 12- 14 September 2007

[S16] Shivani Rao and Avinash Kak, "Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models", *Mining Software Repositories (MSR '11)*, ACM, Waikiki, Honolulu, Hi, USA, 21-22 May 2011

[S17] Chandan R. Rupakheti and Daqing Hou, "Satisfying Programmers' Information Needs in API-Based Programming", *19th IEEE International Conference on Program Comprehension (ICPC '11)*, IEEE, pp. 250-253, Washington, DC, USA, 2011

[S18] Fernando Miguel Carvalho and Joao Cachopo, "STM with Transparent API Considered Harmful", *Proceedings of the 11th International Conference on Algorithms and architectures for parallel processing (ICA3PP'11)*, Springer, pp. 326–337, Verlag Berlin Heidelberg, 2011

[S19] Tony C. Shan and Winnie W. Hua, "Taxonomy of Java Web Application Frameworks", *Proceedings of the IEEE International Conference on e-Business (ICEBE '06)*, IEEE, pp. 378-385, Washington, DC, USA, 2006

[S20] Jens Gerken, Hans-Christian Jetter, Michael Zöllner, Martin Mader and Harald Reiterer, "The Concept Maps Method as a Tool to Evaluate the Usability of APIs*"*, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*, ACM, pp. 3373-3382, Vancouver, BC, Canada, 7–12 May 2011

[S21] Brian Ellis, Jeffrey Stylos, and Brad Myers, "The Factory Pattern in API Design: A Usability Evaluation", *29th International Conference on Software Engineering (ICSE'07)*, IEEE, pp.302-312, Minneapolis, Minnesota, 20 - 26 May 2007

[S22] Jeffrey Stylos and Brad A. Myers, "The Implications of Method Placement on API Learnability", *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (SIGSOFT '08)*, ACM, pp. 105-112, Atlanta, Georgia, USA, 9–15 November 2008

[S23] Daniel Hoffman and Paul Strooper, "Tools and Techniques for Java API Testing", *Australian Software Engineering Conference*, IEEE, pp.235, Gold Coast, Queensland, Australia, 28 -30 April 2000

[S24] Martin P. Robillard, "What Makes APIs Hard to Learn? Answers from Developers", *IEEE Computer Society Press Los Alamitos*, IEEE, Vol. 26, Issue 6, pp. 27-34, CA, USA, November 2009

[S25] Hikaru Fujiwara, Shinji Kusumoto, Katsuro Inouea, Ayane Suzuki, Toshifusa Ootsubo and Katsuhiko Yuura, "Case studies to evaluate a domain specific application framework based on complexity and functionality metrics", *Information and Software Technology,* Elsevier, Vol. 45, Issue 1, pp. 43–49, January 2003

[S26] Uwe Kernebeck, "Component libraries for software re- use", *Proceedings of the 1996 Avionics Conference and Exhibition*, Elsevier, Vol. 21, Issue 1, pp. 49-54, July 1997

[S27] Rajesh Sudarsan and Calvin J. Ribbens, "Design and performance of a scheduling framework for resizable parallel applications", *Parallel Computing*, Elsevier, Vol. 36, Issue 1, pp. 48–64, January 2010

[S28] Erik Berglund, "Designing electronic reference documentation for software component libraries", *The Journal of Systems and Software*, Elsevier, Vol. 68, Issue 1, pp. 65–75, 15 October 2003

[S29] Balazs Dezso, Alpar Juttner and Peter Kovacs, "LEMON – an Open Source C++ Graph Template Library", *Proceedings of the Second Workshop on Generative Technologies (WGT 2010)*, Elsevier, Vol. 264, Issue 5, pp. 23–45, 7 July 2011

[S30] Yongchang Ren, Deyi Jiang, Tao Xing and Ping Zhu, "Research on software development platform based on SSH framework structure", *Procedia Engineering*, Elsevier, Vol. 15, pp. 3078–3082, 2011

[S31] Binh Pham and Ross Brown, "Visualisation of fuzzy systems: requirements, techniques and framework", *Future Generation Computer Systems*, Vol. 21, Issue 7, pp. 1199–1212, July 2005

[S32] Murat Osman Unalir, Oguz Dikenelli and Erden Basar, "FedeRaL: A Tool for Federating Reuse Libraries over the Internet", *Research and Advanced Technology for Digital Libraries*, Springer, pp. 374 - 383, Verlag Berlin Heidelberg 2000

[S33] Marta Sabou, "From Software APIs to Web Service Ontologies: A Semi-automatic Extraction Method", *Methods and Tools of Parallel Programming Multicomputers*, Springer, pp. 410–424, Verlag Berlin Heidelberg 2004

[S34] Takeru Inoue, Hiroshi Asakura, Yukio Uematsu, Hiroshi Sato and Noriyuki Takahashi, "Rapid Development of Web Applications by Introducing Database Systems with Web APIs", *Advances in Software Engineering*, Springer, pp. 327–336, Verlag Berlin Heidelberg 2010

[S35] S. Lennart Johnson and Kapil K. Mathur, "Scientific Software Libraries for Scalable Architectures", *Research and Advanced Technology for Digital Libraries Lecture Notes in Computer Science*, Springer, 2005

[S36] Zhenyu Liu, Genxing Yang and Lizhi Cai, "Test Suite Cooperative Framework on Software Quality", *The Semantic Web-ISWC Lecture Notes in Computer Science*, Springer, pp. 289–292, Verlag Berlin Heidelberg 2009

[S37] Arnaud Gotlieb and Patrick Bernard, "A Semi-empirical Model of Test Quality in Symmetric Testing: Application to Testing Java Card APIs", *Sixth International Conference on Quality Software (QSIC'06)*, IEEE, pp.329-336, Beijing, China, 27 - 28 October 2006

[S38] Mohd Fadzli Marhusin, Henry Larkin, Chris Lokan and David Cornforth, "An Evaluation of API Calls Hooking Performance", *International Conference on Computational Intelligence and Security*, IEEE Vol. 1, pp. 315-319, 13 – 17 December 2008

[S39] Zhenchang Xing and Eleni Stroulia, "API-Evolution Support with Diff-CatchUp", *IEEE Transactions on Software Engineering*, IEEE, Vol. 33, No. 12, pp. 818-836, December 2007

[S40] Ye-Chi Wu, Lee Wei Mar and Hewijin Christine Jiau, "CoDocent: Support API Usage with Code Example and API Documentation", *Fifth International Conference on Software Engineering Advances*, IEEE, pp.135-140, Nice, France, 22 - 27 August 2010

[S41] Bashar Gharaibeh, Tien N. Nguyen and J. Morris Chang, "Coping with API Evolution for Running, Mission-Critical Applications Using Virtual Execution Environment", *Seventh International Conference on Quality Software (QSIC 2007)*, IEEE, pp.171-180, Portland, Oregon, USA, 11- 12 October 2007

[S42] Daniel Ratiu and Jan Juerjens, "Evaluating the Reference and Representation of Domain Concepts in APIs", *The 16th IEEE International Conference on Program Comprehension*, IEEE, pp.242-247, 10 - 13 June 2008

[S43] Daqing Hou and Xiaojia Yao, "Exploring the Intent behind API Evolution: A Case Study", *8th Working Conference on Reverse Engineering*, IEEE, pp.131-140, Limerick, Ireland, 17 - 20 October 2011

[S44] Uri Dekel and James D. Herbsleb, "Improving API documentation usability with knowledge pushing", *IEEE 31st International Conference on Software Engineering*, pp.320-330, Vancouver, BC, Canada, 16- 24 May 2009

[S45] David Kawrykow and Martin P. Robillard, "Improving API Usage through Automatic Detection of Redundant Code", *IEEE/ACM International Conference on Automated Software Engineering*, IEEE, pp. 111-122, Auckland, New Zealand, 16 - 20 November 2009

[S46] Robert B. Watson, "Improving software API usability through text analysis: A case study", *International Professional Communication Conference*, IEEE, pp.1-7, Waikiki, HI, USA, 19 - 22 July 2009

[S47] Minhaz F. Zibran, Farjana Z. Eishita and Chanchal K. Roy, "Useful, But Usable? Factors Affecting the Usability of APIs", *18th Working Conference on Reverse Engineering*, IEEE, pp.151-155, Limerick, Ireland, 17 - 20 October 2011