Πριν ξεκινήσουμε:
τι κάνουμε και γιατί

Γιάννης Σμαραγδάκης

Τα κυριότερα παραδοτέα/προϊόντα μιας διεργασίας ανάπτυξης λογισμικού

- Άρθρωση του προβλήματος, κατανόηση (προδιαγραφές απαιτήσεων)
  - Ποιο πρόβλημα λύνουμε;
- Μορφοποίηση της λύσης (αρχιτεκτονική)
  - Πώς ίσως λύνεται το πρόβλημα
- Αναγωγή της λύσης στην πράξη (σχεδίαση)
  - Πώς θα λύσουμε πράγματι το πρόβλημα;
- Υλοποίηση της λύσης (συγγραφή κώδικα)
- Σχέσεις μεταξύ των παραπάνω
- Αποδείξεις συνέπειας των παραπάνω (ανάλυση/δοκιμασία - testing)

Βασική έμφαση της τεχνολογίας λογισμικού

- Πώς να περιγράψουμε προϊόντα;
  - τα συστατικά τους
  - τη σχέση και αλληλεπίδρασή τους
- Ποιες διαδικασίες πρέπει να ακολουθηθούν για να αναπτύξουμε τέτοια προϊόντα;
  - και να εξασφαλίσουμε την «ποιότητά» τους εν τέλει;
  - (παραδείγματα διαδικασίας;)
- Πώς να αναπτύξουμε και να εξελίξουμε προϊόντα με:
  - αποδεκτό κόστος
  - βελτιωμένη ποιότητα

Τεχνολογία λογισμικού = Προϊόντα + Διαδικασίες

Τι πιστεύουν εργαζόμενοι προγραμματιστές (για διάφορες διαδικασίες, πρακτικές, κλπ.)

- Ρώτησα ανθρώπους που αναπτύσουν λογισμικό επαγγελματικά εδώ και χρόνια
  - –σε περιβάλλον ομάδας, μεγάλη εταιρία λογισμικού
- Πολλές και ποικίλες γνώμες!
  - –τις οποίες θα ξανα-αναφέρουμε κατά τη διάρκεια του εξαμήνου
  - –μεταφρασμένα απ’ευθείας αποσπάσματα

Δομή ενός πραγματικού έργου ανάπτυξης λογισμικού

- Πόσοι προγραματιστές πλήρους απασχόλησης νομίζετε ότι δουλεύουν στο λογισμικό που χρησιμοποιείτε;
- Ποιος είναι ο λόγος προγραμματιστών/δοκιμαστών/διαχειριστών;
- Τι ποσοστό του χρόνου αφιερώνεται στη συγγραφή κώδικα;
- Σε τι ποσοστό του χρόνου τους γράφουν κώδικα οι προγραμματιστές;

Χοντρικά

- *ομάδα 7 ατόμων, μακροπρόθεσμο project:*
  - *3 προγραμματιστές*
  - *3 δοκιμαστές*
  - *1 διαχειριστής/διευθυντής (manager)*
- Πολλαπλασιάστε αναλογικά για μεγαλύτερες ομάδες, ή οργανώστε ιεραρχικά
- *Οι προγραμματιστές αφιερώνουν γύρω στο 40% του χρόνου τους γράφοντας κώδικα, άρα 15-20% της συνολικής προσπάθειας της ομάδα πηγαίνει στη συγγραφή κώδικα*

Υπάρχει πάντα ένταση ανάμεσα στη συγγραφή κώδικα και σε άλλες εργασίες

- "Ακόμα και αν η συγγραφή κώδικα είναι μόλις 20% του συνολικού χρόνου, είναι η μόνη φάση στην ανάπτυξη λογισμικού που δεν μπορείς να αποφύγεις!"
  - κατά πολλούς (π.χ. "Code Complete")
  - άλλοι θα πουν ότι δεν μπορείς να αποφύγεις καμμία φάση
- Σκληροπυρηνική άποψη:
  - *Όλα τα άλλα είναι απλά για να αντισταθμίσουν τις ανθρώπινες αδυναμίες των προγραμματιστών*
  - *3 προγραμματιστές + 3 δοκιμαστές + 1 μάνατζερ στο δικό μου project τους περασμένους 18 μήνες = 2 προγραμματιστές (3dev + 3test + 1PM = 2dev)*
  - παίζει ρόλο τι ήταν το project;
- Θα δούμε αυτή την ένταση επανειλημμένα

Περισσότερα αποσπάσματα: προκαταρκτικά, «ελάχιστη στάθμη»

- *Ό,τι μεγέθους project κι αν έχεις, πρέπει να έχεις ένα σύστημα ελέγχου πηγαίου κώδικα (source control system), σύστημα καταχώρησης λαθών (bug tracking), μοναδιαίες δοκιμασίες (unit tests) που να τρέχουν εύκολα, κι ένα σύστημα κατασκευής (build) που να φτιάχνει αυτόματα το τελικό προϊόν και επιβεβαιώνει ότι το σύστημα μπορεί να κατασκευαστεί σε κάθε προσθήκη κώδικα (no breaking the build). Αν κάτι απ'αυτά λείπει, μάζεψε τη σκηνή και άντε σπίτι.*
- *Οργάνωσε εξ αρχής τα εργαλεία ανάπτυξης, ελέγχου πηγαίου κώδικα, διαδικασία κατασκευής, debugger και profiler, αυτόματες δοκιμασίες, δοκιμασίες επίδοσης, κτλ.*

Σκαλωσιές

- *Κανείς δεν επενδύει αρκετά σε καλά συστήματα κατασκευής (build systems) και εργαλεία. Με εκπλήσει που στην [εταιρία] ξοδεύουμε τόσο κόπο για το ίδιο το προϊόν ενώ το σύστημα κατασκευής που το παράγει είναι πάντα μπακαλοδουλειά. Σκέτος εφιάλτης. Υπάρχει πάντα ένα μίγμα από makefiles και perl scripts και [ειδικό εργαλείο της εταιρίας] και ένα σωρό αναξιόπιστες ταρζανιές που τρελαίνουν κάποιον σαν εμένα που νοιάζεται γι'αυτό το κομμάτι της διαδικασίας και για το πώς υποφέρει η παραγωγικότητα όλων μας απ΄αυτό.*

Συμβάσεις

- *Φτιάξτε έναν «αρχιτεκτονικό κατάλογο ελέγχου» για κομβικά ζητήματα όπως συμβάσεις πολυνηματικού προγραμματισμού, επανείσοδο (re-entrancy), αντοχή σε σφάλματα (μάλλον αυτό είναι πολύ προχωρημένο), ασύγχρονη είσοδο/έξοδο που μπορεί να μπλοκάρει για αόριστο χρόνο (πιο ρεαλιστικό), χειρισμό λανθασμένων εισόδων (που να περιλαμβάνει περίεργες περιπτώσεις όπως αντικείμενα που δημιουργούνται από λάθος factory), συνθήκες χαμηλής μνήμης, κλπ.*

  Αποφασίστε επίσης για βασικές συμβάσεις όπως αν ο η καλούσα ή η κλειθείσα συνάρτηση αρχικοποιεί τις παραμέτρους με τις οποίες εξάγει δεδομένα, συμβάσεις για τη σειρά παραμέτρων (οι έξοδοι τελευταίες ή πρώτες;), κλπ. Στον κώδικα για το [πρόσφατο project] υπάρχει εκπληκτική ποσότητα κώδικα τυφλοσούρτη για τέτοια πράγματα.
- Πειθαρχημένη παρέκλιση, “REVIEW”

Διαδικασίες

- *Αν ήμουν υπεύθυνος ομάδας θα έδινα έμφαση στη γρήγορη ανάπτυξη πρότυπης υλοποίησης (rapid prototyping). Όσο πιο λίγο περιπετειώδες είναι ένα έργο τόσο λιγότερο χρειάζεσαι πρότυπα υλοποίησης, αλλά αν δεν υπάρχουν **πολύ πλήρεις προδιαγραφές**, θα έλεγα ότι μια πρότυπη υλοποίηση αξίζει τον κόπο και με το παραπάνω. Μπορεί κανείς να χρησιμοποιήσει το πρότυπο για να φτιάξει ένα αρχικό σύνολο δοκιμών, αν και είναι μάλλον καλύτερο να κρατηθεί το πρότυπο μακριά από τους προγραμματιστές όταν αρχίσουν την πραγματική υλοποίηση, αλλιώς απλά θα εξομοιώσουν το πρότυπο.*
- *Το "Scrum" είναι δημοφιλής μέθοδος τελευταία. Έχω πάρει μόνο μια ιδέα αλλά έχει σύντομους στόχους (milestones), καλές προτεραιότητες, και πολύ άμεσο πάρε-δώσε μεταξύ των μελών της ομάδας. Όλα αυτά ακούγονται πολύ καλά για μικρές ομάδες.*

Περί μάκρο-διαδικασιών

- *Έχω παρατηρήσει ότι όταν κάποιος έχει να ακολουθήσει μια διαδικασία (π.χ. συμπλήρωσε μια φόρμα για το «μοντέλο απειλών ασφαλείας») το κάνει πολύ ευχαρίστως γιατί σημαίνει ότι σημειώνει συγκεκριμένη πρόοδο για εκείνη την ώρα της ημέρας. Είναι πολύ εύκολο να γεμίσεις τη μέρα ενός εργαζομένου με τέτοια πράγματα και θα έχουν υψηλή προτεραιότητα γιατί είναι πολύ πιο εύκολο για το manager να πει «δεν συμπλήρωσες τη φόρμα Χ» παρά να πει «δεν γράφεις καλό κώδικα» ή «η πρόοδος σου είναι αργή».*

Διαδικασίες και Μέγεθος Έργου

- *Μεγάλες ομάδες που δουλεύουν σε μεγάλες βάσεις κώδικα με περισσότερη πολυπλοκότητα πρέπει να έχουν πολλή «διαδικασία». Οι καλές μεγάλες ομάδες θα διαλέξουν το έλαχιστο διαδικασίας που θα δώσει το μέγιστο πλεονέκτημα. Όμως εν τέλει, αν κάποιος δεν πολυθέλει διαδικασίες θα είναι πιο ευτυχισμένος σε μικρότερες ομάδες/προϊόντα.*

Διαδικασία και Σχεδιασμός

- *Χρειαζόμασταν οπωσδήποτε περισσότερο χρόνο για να σχεδιάσουμε πριν όλα τα άλλα. Είχαμε πολύ χρόνο (μήνες!) και τον σπαταλήσαμε σε «ομάδες εργασίας» που συναντιόντουσαν συνεχώς και έχαναν το χρόνο τους σε στοιχειώδη πράγματα. Μετά αποσπαστήκαμε από ένα άλλο project που μας φόρτωσε η διοίκηση και στο τέλος είχαμε μια βδομάδα για να συμπληρώσουμε το 95% του σχεδιασμού.*

Διαδικασία και Σχεδιασμός

- *Ένας φίλος μου είναι σε μια άλλη ομάδα στην [εταιρία] και έχουν «κρίση ποιότητας». Στα τελευταία τους milestones είχαν βαθμό οπισθοδρόμησης (regression rate) 30% ανά checkin (δεν έχω ιδέα τι στατιστικά θεωρούνται φυσιολογικά γενικά αλλά 30% δεν μου φαίνεται και υπερβολικά μεγάλο – ίσως 15% είναι πιο λογικό;) κι έτσι η «λύση» είναι ότι ένας από τους αρχιτέκτονες (που ο φίλος μου τον θεωρεί πανάχρηστο και ηλίθιο) προσπαθεί να επιβάλει κάποια μέτρα ποιότητας (π.χ. κυκλωματική πολυπλοκότητα, ποσοστό σχολίων ανά γραμμή κώδικα, κτλ.) που θα επιβάλονται αυτόματα στο checkin για να βελτιώσουν τον κώδικα. Ο φίλος μου μου έστειλε το κείμενο αυτού του τύπου και βασικά έλεγε «για να διορθώσουμε τα προβλήματά μας θα κάνουμε αυτό» με κάποια μέτρα ποιότητας που τα έβγαλε απ' το κεφάλι του χωρίς καμμία λογική σύνδεση με το πρόβλημα και χωρίς κανένα πλάνο για να εκτιμήσει αν η νέα διαδικασία δουλεύει.*

Διαδικασία και Δοκιμασίες

- *Μια διαδικασία που πραγματικά μ'αρέσει είναι η ανάπτυξη που καθοδηγείται από δοκιμασίες (test-driven development). Έγραψα στο blog μου γι'αυτό πριν λίγο καιρό [...] Λέω επίσης για την «κάλυψη κώδικα» σαν ένα χρήσιμο μέτρο ποιότητας. Είναι το μόνο μέτρο που ξέρω που το χρησιμοποιούν σχεδόν όλοι. Η ιδέα είναι ότι αν έχεις λιγότερο από ~80% κάλυψη τεμαχίων (block coverage) από τα test σου τότε κάτι πάει στραβά.*

Διαδικασία και Δοκιμασίες

- *Η καλύτερη προσέγγιση στην ανάπτυξη λογισμικού είναι να βρεις ένα τρόπο να κάνεις «τα πάντα δυο φορές», κάτι σαν το διπλογραφικό σύστημα στη λογιστική. Σε κάποιες περιπτώσεις βασίζεσαι στο σύστημα τύπων. Σε άλλες γράφεις asserts. Σε άλλες μοναδιαίες δοκιμασίες. Θα έλεγα ότι αυτό είναι το πιο σημαντικό: υπάρχει κάποιος τρόπος που το μηχάνημα να ελέγξει το κάθε στοιχείο ενός προγράμματος έναντι κάποιου άλλου στοιχείου; Έλεγξε όσο γίνεται, όσο πιο νωρίς γίνεται, χρησιμοποιώντας compile-time asserts και πολύπλοκους τύπους που θα μεταγλωττιστούν σε σχεδόν τίποτα. (Βρήκα ένα σωρό bugs πριν κανά-δυο μήνες όταν άλλαξα ένα typedef που χρησιμοποιούταν με ελαφρά διαφορετικούς τρόπους σε ένα class template με 4 ασύμβατα instantiations που δεν μπορούσες να χρησιμοποιήσεις το ένα αντί για το άλλο χωρίς μετατροπή.) Μια καλή ιδέα είναι ένα ξεχωριστό σύστημα κατασκευής που τρέχει αυτόματες δοκιμασίες πολύ αργά αλλά κάνει πολλούς ελέγχους συνέπειας σε όλες τις δομές δεδομένων.*

Διαδικασία και Δοκιμασίες

- [Από τον ίδιο που είπε 3dev + 3test + 1PM = 2dev]
  Ένα πρόβλημα που έχουμε με δοκιμαστές που δεν κάνουν τίποτε άλλο είναι ότι δεν υπάρχει τρόπος ελέγχου τους. Λένε απλά «αυτό δοκιμάστηκε» και όλοι (ή μάλλον ο manager) τους πιστεύουν. Πρότεινα δύο πιθανά επίπεδα ελέγχου. Το πρώτο είναι να επιτρέψουμε στους προγραμματιστές να βάλουν σε συγκεκριμένα σημεία του κώδικα κάποιο macro ή κάτι τέτοιο που να σημαίνει «αυτή η περίπτωση πρέπει να καλυφθεί». Οι δοκιμασίες θα πρέπει να καλύψουν όλες αυτές τις περιπτώσεις. Οι δοκιμαστές μας στηρίζονται σε μέτρα κάλυψης και εξ'αιτίας διάφορων περιπτώσεων για χειρισμό λαθών ένα 70% θεωρείται αποδεκτό. Κανείς δεν ξέρει αν σημαντικές περιπτώσεις μένουν χωρίς έλεγχο. Το ακόμα υψηλότερο επίπεδο ελέγχου θα ήταν να επιτρέψουμε στους προγραμματιστές να προσθέσουν macros που θα εισάγουν bugs επίτηδες σε κάποιο ειδικό build.
- [άλλος προγραμματιστής, χωρίς να απαντάει στον προηγούμενο]
  Κοίτα, μιλάω από εμπειρία. Οι δοκιμαστές μας βρίσκουν ένα σωρό bugs.