Κατασκευή λογισμικού (software development)

Η Κατασκευή λογισμικού (Software construction) είναι μέρος της μηχανικής λογισμικού. Σκοπός της είναι να κατασκευάσει λογισμικό συνδυάζοντας κώδικα, έλεγχο ή επαλήθευση, τεστ, αποσφαλμάτωση και γενικά «καλό» λογισμικό με κριτήρια που αναφέρονται παρακάτω.

Οι όροι κατασκευή λογισμικού και προγραμματισμός θα έχουν την ίδια έννοια και θα χρησιμοποιούνται εναλλακτικά στη συνέχεια.

Μηχανική λογισμικού (Software engineering)

Ασχολείται με το σχεδιασμό, ανάπτυξη και συντήρηση λογισμικού και καλύπτει το μέρος της κατασκευής λογισμικού.

Στην ανάπτυξη λογισμικού έχουμε τις παρακάτω φάσεις:

Ανάλυση απαιτήσεων (Requirements analysis)

Καθορίζει τις απαιτήσεις της εφαρμογής.

Σχεδιασμός λογισμικού (Software design)

Γίνεται ο σχεδιασμός που περιλαμβάνει προδιαγραφές, περιγραφή, γραφικά (UML). Γενικά είναι ο «επί χάρτου» σχεδιασμός.

Κατασκευή λογισμικού (Software construction)

Στη φάση αυτή υλοποιείται η εφαρμογή σε μορφή κώδικα. Η υλοποίηση ακολουθεί μία σειρά από στάδια ή φάσεις, ελέγχους, αποσφαλμάτωση, τεστ, αξιολόγηση, εγκατάσταση κ.λπ. Όλα τα παραπάνω, και άλλα σχετικά θέματα, θα παρουσιαστούν αναλυτικά στο μάθημα αυτό.

Τεστ λογισμικού (Software testing)

Συστηματικός έλεγχος του λογισμικού για να αποκαλυφθούν λάθη (bugs).

Αποσφαλμάτωση (Debugging)

Ο εντοπισμός και διόρθωση σφαλμάτων.

Εγκατάσταση λογισμικού (Software deployment)

Η εγκατάσταση λογισμικού σε πραγματικό περιβάλλον έτοιμο για χρήση.

Συντήρηση λογισμικού (Software maintenance)

Η συντήρηση λογισμικού είναι απαραίτητη και σκοπός της είναι να διορθώνει λάθη που θα παρουσιαστούν, βελτιώσεις που θα χρειαστούν, προσαρμογές που θα απαιτηθούν.

Κριτήρια αξιολόγησης λογισμικού

Ανεξάρτητα της προσέγγισης, μεθοδολογίας, εργαλείων κλπ. που θα χρησιμοποιηθούν στη κατασκευή λογισμικού τα κριτήρια αξιολόγησης είναι:

Αξιοπιστία (Reliability)

Πόσο συχνά σωστά είναι τα αποτελέσματα του προγράμματος.

Σταθερότητα (Robustness)

Πόσο ικανοποιητικά αντιμετωπίζει ή διαχειρίζεται τα errors (όχι bugs).

Χρηστικότητα (Usability)

Πόσο χρηστικό είναι στον χρήστη. Αυτό κρίνεται από το πόσο φιλικό στο χρήστη είναι το περιβάλλον εργασίας.

Φορητότητα (Portability)

Πόσο φορητό είναι σε διαφορετικά συστήματα με διαφορετικό υλικό ή / και λογισμικό.

Συντηρησιμότητα (Maintainability)

Η ευκολία με την οποία μπορεί το πρόγραμμα να βελτιωθεί, να επεκταθεί, να διορθωθεί, να προσαρμοστεί σε νέα δεδομένα.

Απόδοση/Αποτελεσματικότητα (Efficiency/performance)

Εξαρτάται από τη σωστή διαχείριση πόρων και ταχύτητα του λογισμικού.

Αναγνωσιμότητα (Readability of source code)

Σημαντικό κριτήριο. Εκφράζει την ευκολία με την οποία ο προγραμματιστής μπορεί να διαβάσει και να κατανοήσει τον κώδικα.

Προτεραιότητα κριτηρίων

Από όλα τα παραπάνω υπολογίσιμα κριτήρια τα παρακάτω ίσως είναι τα πιο σημαντικά που πρέπει να προσέχει ένας προγραμματιστής.

- Αξιοπιστία (Reliability)
- Αναγνωσιμότητα (Readability of source code)
- Συντηρησιμότητα (Maintainability)

Αρχιτεκτονική λογισμικού

Παράγοντες που λαμβάνονται υπόψη από τον αναλυτή (αρχιτέκτονα λογισμικού - συστήματος)

Οργανισμός προγράμματος

Γενική περιγραφή του συστήματος (ίσως UML).

Κυρίες κλάσεις (οντότητες)

Ποιες είναι οι κυρίες κλάσεις – οντότητες.

Σχεδιασμός δεδομένων

Οι πηγές δεδομένων όπως αρχεία, βάσεις δεδομένων και πως αυτά θα σχεδιαστούν.

Επιχειρηματικοί κανόνες

Όπως η απαίτηση το περιεχόμενο να ανανεώνεται κάθε ήμερα.

Σχεδιασμός διεπαφής χρήστη

Είναι ένα μεγάλο κεφάλαιο και ασχολούνται πλέον ειδικοί. Σε κάθε περίπτωση ο σχεδιαστής πρέπει να συνεργάζεται με τον προγραμματιστή.

Διαχείριση πόρων

Είναι οι πόροι αρκετοί να καλύψουν τις ανάγκες του έργου;

Ασφάλεια

Πρέπει να ακολουθηθούν συγκεκριμένες οδηγίες που θα παρέχουν ασφάλεια όπως από εισαγωγή δεδομένων σε φόρμες, cookies, κρυπτογραφία κ.λπ.

Απόδοση (ταχύτητα)

Γίνεται εκτίμηση προτεραιοτήτων ανάμεσα σε ταχύτητα, πόρους και κόστος.

Επεκτασιμότητα

Κατά πόσο το σύστημα μπορεί να επεκταθεί σε μελλοντικές ανάγκες και κατά πόσο έχει σχεδιαστεί σωστά για αυτή τη δυνατότητα.

Διαλειτουργικότητα (Interoperability)

Μπορεί να συνεργάζεται με άλλα συστήματα, λογισμικό κ.λπ.

Internationalization (“I18n”)/Localization (“L10n”)

Όταν το πρόγραμμα παρέχει δυνατότητα για πολλαπλό Localization. Localization η διαδικασία “μετάφρασης” του προγράμματος σε άλλη γλώσσα.

Επεξεργασία σφαλμάτων (errors)

Ένα από τα σημαντικά θέματα στο χώρο του προγραμματισμού.

Γλώσσες προγραμματισμού

Κάθε γλώσσα έχει το πλεονεκτήματά της και τα μειονεκτήματά τους. Και ο προγραμματιστής πρέπει να διαλέξει την κατάλληλη για κάθε έργο.

C

Γενικής χρήσης γλώσσα σχετισμένη περισσότερο με το λειτουργικό UNIX. Θεωρείται για πολλούς ως μητέρα γλώσσα και για άλλες γλώσσες.

C++

Αντικειμενοστρεφής γλώσσα βασισμένη επάνω στη C. Παρέχει όλα τα καλά της αντικειμενοστέφειας όπως classes, polymorphism, exception handling, templates κ.λπ.

C#

Γενικής χρήσης αντικειμενοστρεφής γλώσσα αναπτυγμένη από τη Microsoft για τη δική της πλατφόρμα. Είναι παρόμοια της C, C++, και Java.

Java

Γενικής χρήσης αντικειμενοστρεφής γλώσσα αναπτυγμένη αρχικά από τη Sun Microsystems και τώρα από την Oracle.

Για να τρέξουν οι εφαρμογές της java χρειάζονται τη JVM (java virtual machine). Αρκετά δημοφιλής ειδικότερα σε εφαρμογές web.

Javascript

Η javascript είναι μια σκριπτ γλώσσα και χρησιμοποιείται για δυναμικές ιστοσελίδες στη μεριά του πελάτη (client side).

Perl

Βασίζεται στη C και UNIX utilities. Χρησιμοποιείται περισσότερο από διαχειριστές συστήματος. “Perl” ακρωνύμιο του Practical Extraction and Report Language.

php

Γλώσσα σκριπτ και open-source. Χρησιμοποιείται για διαδικτυακές εφαρμογές στη μεριά του server (server side).

Python

Μια αντικειμενοστρεφής γλώσσα τύπου σκριπτ και χρησιμοποιείται σε πολλά περιβάλλοντα. Βρίσκει εφαρμογή περισσότερο σε διαδικτυακές εφαρμογές.

Visual Basic

Υποστηρίζεται από τη Microsoft για τη δική της πλατφόρμα.

SQL

Γλώσσα ερωτημάτων (query language). Χρησιμοποιείται για τη διαχείριση δεδομένων σε βάσεις δεδομένων.

Μεθοδολογίες και μοντέλα

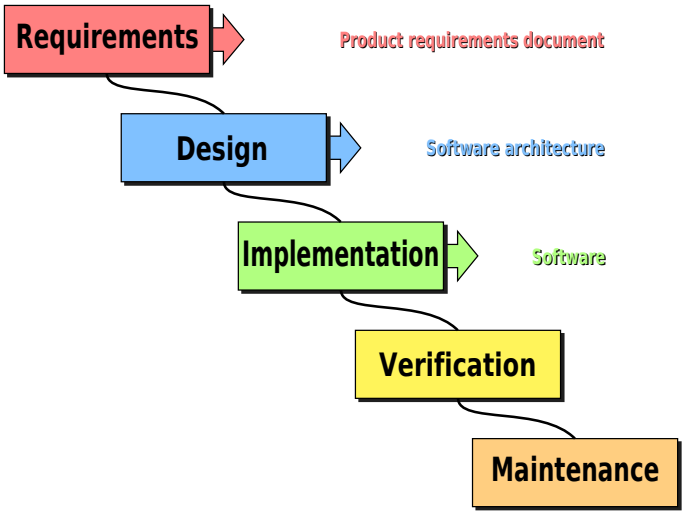
Οι μεθοδολογίες που συναντάμε είναι:

Waterfall model (το μοντέλο του καταρράκτη)

Ακολουθεί τις φάσεις (Ανάλυση, σχεδιασμός, υλοποίηση, τεστ, εγκατάσταση και συντήρηση).

Με το μοντέλο αυτό το τελικό προϊόν παραδίδεται αφού πρώτα ολοκληρωθεί η ανάλυση μετά ο σχεδιασμός κλπ.

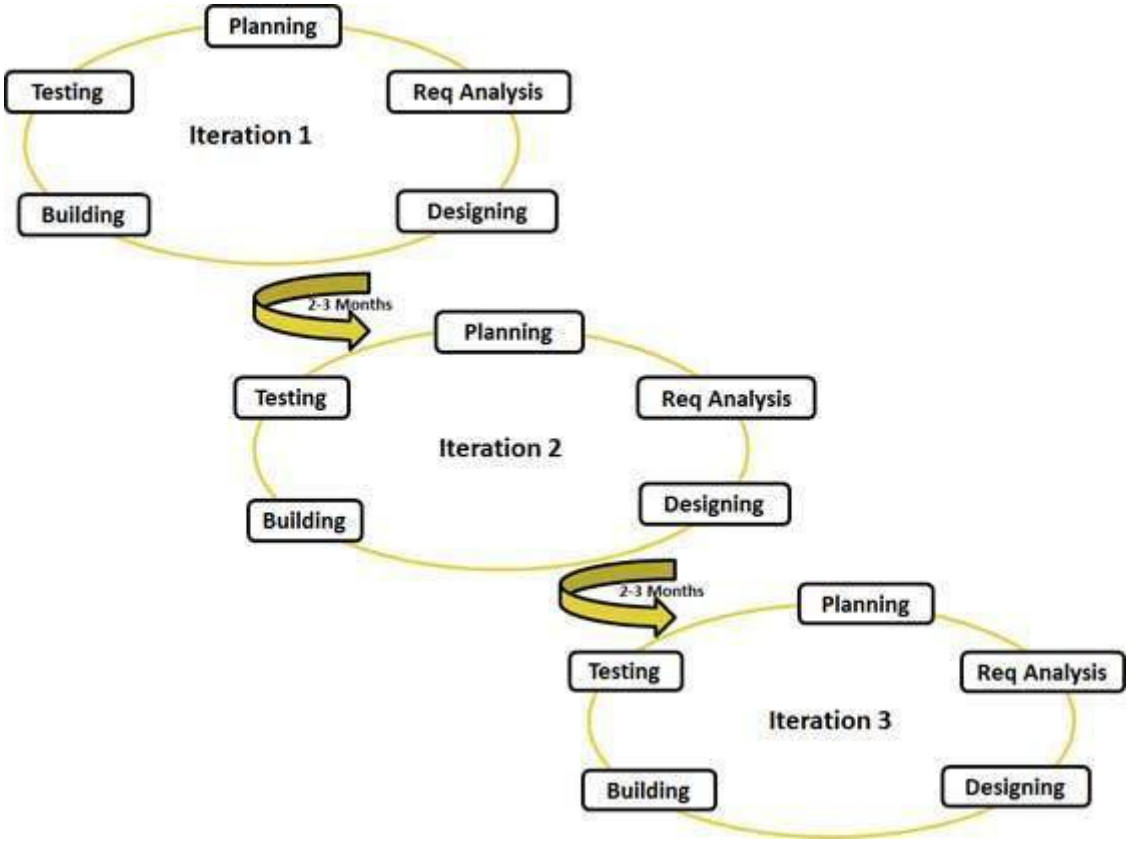
Σε κάθε φάση μπορούμε να γυρίσουμε σε κάποια προηγούμενη αν χρειαστεί και αυτό μπορεί να επαναλαμβάνεται μέχρι την τελική παράδοση.



Agile software development

Και το μοντέλο αυτό ακολουθεί τις φάσεις (Ανάλυση, σχεδιασμός, υλοποίηση, τεστ, εγκατάσταση και συντήρηση).

Με το μοντέλο αυτό το προϊόν παραδίδεται τμηματικά. Αρχικά παραδίδεται ένα πρώτο προϊόν και στη συνέχεια επαναλαμβάνεται πάλι η ίδια διαδικασία μέχρι να ολοκληρωθεί το έργο.



Τύποι δεδομένων

Ο τύπος δεδομένων ορίζεται ως ένα σύνολο τιμών και ένα σύνολο πράξεων και που το αποτέλεσμα ανήκει στο σύνολο τιμών.

Η Java έχει πρωταρχικούς τύπους (primitive types) τιμών όπως:

- char (16 bit. Για unicode χαρακτήρες όπως 'A' και 'Φ')
- int (32 bit. Ακέραιες τιμές όπως 5 και -200, μέχρι ± 2^31, ή περίπου ± 2 billion)

- long (64 bit. για ακέραιες τιμές μέχρι ± 2^63)
- short (16 bit)
- boolean (για true ή false)
- float (32 bit. Για δεκαδικές τιμές)
- double (64 bit. Για δεκαδικές τιμές)

Η Java έχει επίσης τύπους δεδομένων σε μορφή αντικειμένου όπως:

- String εκφράζει μια συμβολοσειρά.
- BigInteger εκφράζει έναν ακέραιο για μεγάλες τιμές.

Στη Java οι πρωταρχικοί τύποι γράφονται με πεζά γράμματα ενώ τα αντικείμενα-τύποι με κεφαλαίο το πρώτο (όπως οι κλάσεις).

Στατικός έλεγχος, δυναμικός έλεγχος και μη έλεγχος

Υπάρχουν γλώσσες στατικού ελέγχου και δυναμικού ελέγχου.

Σε αυτές του στατικού (java) γίνεται έλεγχος πριν τη μεταγλώττιση ενώ στις δυναμικού κατά την εκτέλεση (run time) όπως η python.

Στον στατικό έλεγχο πιάνονται (catch):

- Λάθος σύνταξη
- Λάθος όνομα (reserved words)
- Λάθος αριθμός ορισμάτων σε συνάρτηση
- Λάθος τύπος ορισμάτων
- Λάθος τύπος επιστροφής (return)

Στον δυναμικό έλεγχο πιάνονται (catch):

- Λάθος τιμές ορισμάτων (x/y όπου y=0)
- Λάθος τιμές επιστροφής
- Τιμές εκτός ορίων (int, πίνακες)
- Κλήση ή αναφορά σε αντικείμενα με τιμή null

Γενικά στον στατικό έλεγχο το λάθος έχει σχέση με τον τύπο του ορίσματος ενώ στον δυναμικό με την τιμή του ορίσματος.

Στον μη έλεγχο δεν πιάνεται κανένα λάθος.

Κακές πρακτικές

- Μακρύς κώδικας χωρίς έλεγχο
- Μη καταγραφή λεπτομερειών, σημειώσεων, παρατηρήσεων κ.λπ.
- Να υποθέτουμε ότι δεν υπάρχουν λάθη

Καλές πρακτικές

- Γράψτε λίγο κώδικα και ελέγξτε

- Τεκμηριώστε τις προυποθέσεις για σωστή λειτουργία
- Στατικός έλεγχος παντού

Ο στόχος

Ο κύριος στόχος είναι να παράγουμε "καλό" λογισμικό.

- Σωστό, χωρίς λάθη
- Κατανοητό
- Εύκολο σε αλλαγές και ενημέρωση

Εκτός των παραπάνω υπάρχουν και άλλα κριτήρια αλλά αυτά είναι τα πιο βασικά για έναν προγραμματιστή.

Ασκήσεις

Ελέξτε για δυναμικά, στατικά ή μη ελεγχόμενα λάθη στα παρακάτω παραδείγματα.

```
int n = 5;
if (n) {
        n = n + 1;
}
```

```
int big = 200000;
big = big * big;
System.out.println(big);
```

```
double probability = 1/5;
System.out.println(probability);
```

```
int sum = 0;
int n = 0;
int average = sum/n;
System.out.println(average);
```

```
double sum = 7;
double n = 0;
double average = sum/n;
System.out.println(average);
```

```
final int n = 5;
System.out.println(n++);
```
