

Τεχνολογία λογισμικού – Rewind

***softeng lecture* pays zero attention**

***before softeng exam* knows nothing about softeng**

SOLUTION



Disclaimer: Για ΟΠΟΙΟΔΗΠΟΤΕ λάθος ή παράλειψη δεν φέρω ΚΑΜΙΑ ευθύνη.

Συστατικά λογισμικού:

- πηγαίος κώδικας
- προδιαγραφές, εκθέσεις, αναφορές, κείμενα γενικά
- σχέδια
- διαγράμματα
- λοιπά υποστηρικτικά/τεκμηριωτικά/συμπληρωματικά έγγραφα

Αφορά:

- λογισμικό μεγάλης κλίμακας
- εφαρμογές για “κρίσιμες αποστολές”

Λογισμικό = ιδεατή οντότητα

- γίνεται αντιληπτό εκ των αποτελεσμάτων χρήσης
- δεν περιγράφεται εύκολα/μοναδικά

Πολλοί συμμετέχοντες, άλλα (πιθανώς αντικρουόμενα) συμφέροντα ο καθένας

Κύκλος ζωής λογισμικού:

- σύλληψη
- κατασκευή, χρήση και συντήρηση
- απόσυρση

Μοντέλα κύκλου ζωής:

- waterfall:
διακριτές φάσεις ανάπτυξης
- prototyping:
διαδοχικά πρωτότυπα, ολοένα και πιο πλούσια σε χαρακτηριστικά
- λειτουργική επαύξηση:
τμηματοποιημένος καταρράκτης
- σπειροειδής:
κύκλοι εργασιών με σταδιακή επέκταση χαρακτηριστικών (εκτίμηση ρίσκου σε κάθε κύκλο)
- πίδακας:
αντικειμενοστραφής προσέγγιση με επαναχρησιμοποίηση χαρακτηριστικών

Μοντέλο	Μέγεθος εφαρμογών	Μεταβολές στις απαιτήσεις	Προσαρμοστικότητα στον κατασκευαστή	Διάδοση
Καταρράκτη	Μικρό έως μεσαίο	Ανεπιθύμητες	Καμία	Μεγάλη με τάση μείωσης
Πρωτοτυποποίησης	Μικρό ως μεσαίο	Δεκτές	Μικρή	Μικρή με τάση αύξησης
Λειτουργικής επαύξησης	Μεσαίο ως μεγάλο	Ανεπιθύμητες	Καμία	Μικρή με τάση μείωσης
Σπειροειδής	Μεσαίο ως μεγάλο	Δεκτές	Αρκετή	Μικρή με τάση μείωσης
Πίδακα	Οποιοδήποτε	Δεκτές	Αρκετή	Μικρή
Γενικό	Οποιοδήποτε	Δεκτές	Μεγάλη	Μικρή με ισχυρές τάσεις αύξησης

Μοντέλα περιγραφής συστατικών στοιχείων λογισμικού:

- Αφαιρετικά (δεν περιέχουν όλες τις λεπτομέρειες)
- Συμπληρωματικά (πολλά μαζί περιγράφουν πλήρως το λογισμικό)
- Υλοποιήσιμα (καθοδηγούν την κατασκευή του λογισμικού)

Αρνητικά απλών συμβολισμών – ορισμών:

- όχι καθολικά αποδεκτοί
- αντιληπτοί με διφορούμενα
- με πολλές διαφορετικές διατυπώσεις

Τα παραπάνω αποφεύγονται με πρότυπα:

- δομή περιγραφής λογισμικού
- εξασφαλίζουν μία ελάχιστη πειθαρχία
- κοινό σημείο αναφοράς μεταξύ κατασκευαστών

Ορισμός: Έργο (project) λογισμικού

Ένα σύνολο ξεχωριστών, σύνθετων και συνδεδεμένων ενεργειών που έχουν έναν στόχο και πρέπει να ολοκληρωθούν εντός συγκεκριμένου χρονικού διαστήματος και προϋπολογισμού με βάση συγκεκριμένες προδιαγραφές.

Περιορισμοί

- κόστος
- χρόνος
- scope
- ποιότητα

Από τα παραπάνω ελέγχουμε 2 άξονες, αυτούς όπου ο πελάτης είναι ανελαστικός.

Σχέδιο έργου

- ενδιάμεσα παραδοτέα
- εκτίμηση προσπάθειας/κόστους/χρόνου
- milestones

Ιεραρχική κατανομή εργασιών (ούτε πολύ περιληπτική, ούτε υπερβολικά λεπτομερής)

- πακέτα εργασίας (υποέργα)
- επιμέρους δράσεις ανά πακέτο
- αλληλοεξαρτήσεις

Timeline:

- absolute dates ή
- relative time intervals

Critical path: αλυσίδα ενεργειών που αν σπάσει, διαλύεται το έργο.

Slack/float: χρονικό περιθώριο καθυστέρησης κάποιας υπενέργειας χωρίς εξωτερικές επιπτώσεις.

Σχέσεις σε διάγραμμα Gantt:

- FS: *if A=Finished \Rightarrow B can Start*
- FF: *if A=Finished \Rightarrow B can Finish*
- SS: *if A has Started \Rightarrow B can Start*
- SF: *if A has Started \Rightarrow B can Finish*

Ανθρωπο-προσπάθεια:

- 1 AΩ = 1 Ανθρωπο-Ωρα
- 1 AH = 1 Ανθρωπο-Ημέρα
- 1 AM = 1 Ανθρωπο-Μήνας
- 1 AE = 1 Ανθρωπο-Έτος

1 AH = 8 AΩ, 1 AM = 20 ή 21 AH, 1 AE = 11 AM

Ομάδες:

- Μικρές – 1 άτομο, πολλοί ρόλοι
- Μεγάλες – 1 άτομο, 1 ρόλος

Ρόλοι:

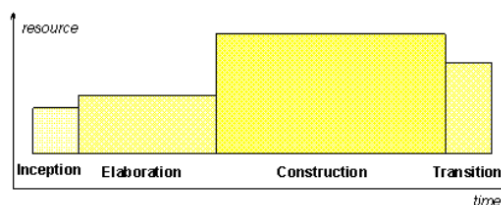
- Project/Product manager:
 - Οριοθέτηση στόχων ομάδας + χαρακτηριστικών λογισμικού
 - Διαχείριση χρόνου και προσπάθειας των μελών σε μακροσκοπικό επίπεδο
- Architect:
 - Major design decisions
 - Θέσπιση: interfaces, components, dependencies
 - Επιλογή εργαλείων
- Team Lead
 - Ανάθεση επιμέρους εργασιών σε λοιπά μέλη
 - Καθημερινή διαχείριση χρόνου + προσπάθειας μελών

Rational Unified Process (RUP):

- Inception
- Elaboration
- Construction
- Transition

Όλα τα παραπάνω είναι ξεχωριστά iterative το καθένα.

	<u>Inception</u>	<u>Elaboration</u>	<u>Construction</u>	<u>Transition</u>
Effort	~5 %	20 %	65 %	10%
Schedule	10 %	30 %	50 %	10%



Inception:

- καθορισμός στόχων
- ανάλυση αναγκών stakeholders
- εμβέλεια, οριακές συνθήκες, κριτήρια αποδοχής
- καθορισμός μέρους των απαιτήσεων

Δραστηριότητες: ο καθορισμός της εμβέλειας του έργου + του business case + εύρεση μίας υποψήφιας αρχιτεκτονικής + προετοιμασία περιβάλλοντος για την ανάπτυξη του έργου.

Ολοκλήρωση: full specification του business case + κριτήρια επιτυχίας + αρχική εκτίμηση ρίσκου + εκτίμηση πόρων για την επόμενη φάση (elaboration).

Elaboration:

- καθορισμός κινδύνων
- διατύπωση “ευσταθούς” οράματος
- καθορισμός αρχιτεκτονικής
- προγραμματισμός πόρων-κόστους

Δραστηριότητες: καθορισμός αρχιτεκτονικής + επικύρωσή της + ορισμός μιας baseline αρχιτεκτονικής.

Ολοκλήρωση: λεπτομερές πλάνο ανάπτυξης + περιβάλλον και εργαλεία + κριτήρια αξιολόγησης τελικού προϊόντος + μοντέλο ανάλυσης θεματικού πεδίου + εφαρμόσιμη baseline αρχιτεκτονική.

Construction:

- διαχείριση πόρων εντός περιορισμών
- διαίρεση σε iterations

Δραστηριότητες: ανάπτυξη software components + διαχείριση πόρων/έλεγχος διαδικασίας + αξιολόγηση iteration.

Ολοκλήρωση: συστατικά στοιχεία λογισμικού με τα χαρακτηριστικά που προστέθηκαν στο iteration + περιγραφή αποτελεσμάτων του iteration + περιπτώσεις και έλεγχοι + ποσοτικά κριτήρια αξιολόγησης επόμενου/ων iteration.

Transition:

- διάθεση προϊόντος στους χρήστες
- διανομή, εγκατάσταση, ρυθμίσεις, bugfixes...

Δραστηριότητες: έλεγχος στο περιβάλλον του πελάτη + προσαρμογές με βάση τα σχόλια + διάθεση για παραγωγική χρήση + υλικά υποστήριξης χρηστών.

Ολοκλήρωση: επικαιροποιημένη τεκμηρίωση + νέες δυνατότητες μετά το τέλος του αρχικού κύκλου.

RUP – Θετικά vs Αρνητικά	
έγκαιρη αποτίμηση κινδύνων	δεν είναι ιδιαίτερα ευέλικτη
δυναμική εξειδίκευση απαιτήσεων	δεν έχει ξεκάθαρες οδηγίες για υλοποίηση
δεν βασίζεται σε docs	πολλή δουλειά για προσαρμογή στο εκάστοτε περιβάλλον
εστιάζει στο ίδιο το λογισμικό	

Agile:

- Σύνολο μεθόδων και πρακτικών που περιγράφονται στο Agile Manifesto.
- Λύσεις από συνεργασία αυτοδιοικούμενων και πολυλειτουργικών ομάδων.

Αρχές:

- self-organizing & cross-functional teams
- προσαρμοστικός σχεδιασμός, εξελικτική ανάπτυξη, ταχεία παράδοση και συνεχής βελτίωση
- ταχεία και ευέλικτη προσαρμογή στις αλλαγές

Agile manifesto

1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	7	Working software is the primary measure of progress.
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	9	Continuous attention to technical excellence and good design enhances agility.
4	Business people and developers must work together daily throughout the project.	10	Simplicity--the art of maximizing the amount of work not done--is essential.
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	11	The best architectures, requirements, and designs emerge from self-organizing teams.
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

SCRUM (μία από πολλές agile μεθοδολογίες):

- Sprint: χρονική περίοδος στην οποία κατασκευάζεται ένα λειτουργικό κομμάτι software.
- Sprint planning: όλη η ομάδα καθορίζει τι θα γίνει στο επόμενο sprint session.
- Daily scrum: 15' για συγχρονισμό ομάδων και προγραμματισμό του επόμενου 24ώρου.
- Sprint review: στο τέλος του sprint session, αξιολόγηση του τι φτιάχτηκε και ενημέρωση του Backlog.
- Sprint retrospective: μετά το sprint review, πριν το sprint planning – αυτοέλεγχος ομάδας και πλάνο για βελτιώσεις στο επόμενο sprint.

(Backlog = features to be implemented)

Διαχείριση λογισμικού:

- version control
- build automation
- bug detection
- tests
- continuous integration
- διαχείριση συστατικών λογισμικού και εκδόσεών τους

Επιπλέον όροι:

- Continuous integration (CI): server που εκτελεί αυτόματα tests μετά από κάθε commit.
- Staging deployment: εγκατάσταση λογισμικού σε ενδιάμεσο περιβάλλον.
- Production deployment: εγκατάσταση λογισμικού στο παραγωγικό περιβάλλον.
- Δημοσίευση σε artifact repository.

Artifact releases:

- pre-release (internal release)
- early access (EA)
 - alpha
 - beta
 - release candidate
- release (GA – general availability)

Συμφωνία πριν την εκκίνηση ενός έργου:

- κοινή τεχνική κουλτούρα
- κοινές συμβάσεις
- κοινά tools & procedures
- κοινά αποδεκτός και ξεκάθαρος ο επιμερισμός ευθύνης (ownership)

Για ownership:

Bus factor = #ανθρώπων που αν τους φύγουν (aka “τους χτυπήσει λεωφορείο”) χάνεται η τεχνογνωσία για κομμάτι του project σε τέτοιο βαθμό που όλο το project σταματάει

Για συμβάσεις:

- δομή κώδικα
- δομή αρχείων

Build automation:

- dependencies
- compilation
- testing
- CI
- software artifact assembly
- publish artifacts

Software artifacts:

- αυτοτελή αρχεία έτοιμα προς εκτέλεση
- βιβλιοθήκες (μερικώς αυτοτελή αρχεία προς ενσωμάτωση σε εφαρμογές)
- δομή & περιεχόμενο εξαρτώνται από γλώσσα, ΛΣ κτλ.

Dependencies:

- compile-time (“static” linking) – συστατικά που πρέπει να είναι διαθέσιμα κατά τη μεταγλώττιση
- runtime (“dynamic” linking) – συστατικά που πρέπει να είναι διαθέσιμα κατά την εκτέλεση
- transitive dependencies - εξάρτηση της εξάρτησης

Versioning: [major].[minor].[revision].[build]

- major: σημαντική αλλαγή
- minor: προσθήκη ή βελτίωση
- revision: patch, bugfix, security update (maintenance release)
- build: αυτόματη αρίθμηση build tool (internal release)

Semantic versioning: [major].[minor].[patch]

- major: breaking change
- minor: add new backwards compatible functionality
- patch: apply backwards compatible bug fix

Μας νοιάζει η έκδοση από άποψη του χρήστη του κώδικά μας, δηλαδή τι public API εκθέτει. Αν κάνουμε αλλαγή που θα σπάσει τα app που έχουν χτιστεί με βάση την τρέχουσα έκδοση και εξαρτώνται άμεσα από αυτή, τότε `major++`, αλλιώς κοιτάμε αν είναι `minor/patch`.

Απαιτήσεις λογισμικού:

- Λειτουργικές: τι λειτουργίες θέλουμε να επιτελεί
- Μη-λειτουργικές: όλα τα άλλα (ασφάλεια, interface κτλ)

SRS:

1. Μέρος 1ο
 1. ταυτότητα εγγράφου
 2. σκοπός
 3. εμβέλεια
 4. ορισμοί, ακρωνύμια, συντομογραφίες
 5. πηγές αναφορών
 6. περίληψη
2. Μέρος 2ο
 1. στίγμα (ποιος είναι ο ανταγωνισμός & πού διαφέρει το τρέχον προϊόν)
 2. προοπτική (επιδιωκόμενη πορεία λογισμικού)
 3. γενικές λειτουργίες
 4. χαρακτηριστικά χρηστών
 5. περιορισμοί (γενικοί, δλδ περιβάλλον χρήσης, σχέσεις με πελάτη, νόμοι κτλ)
 6. παραδοχές & εξαρτήσεις
3. Μέρος 3ο
 1. Απαιτήσεις εξωτερικών διεπαφών
 1. χρήστη
 2. υλικού
 3. λογισμικού

4. επικοινωνιών
2. Λειτουργικές απαιτήσεις
 1. Τρόπος λειτουργίας 1, 2, 3...
 1. Λειτουργική απαίτηση 1, 2, 3...
3. Απαιτήσεις επιδόσεων
4. Περιορισμοί σχεδίασης
 1. υλικό (λόγω hardware που θα το τρέξει)
 2. συμμόρφωση σε πρότυπα
5. Χαρακτηριστικά
 1. αξιοπιστία
 2. διαθεσιμότητα
 3. ασφάλεια
 4. μεταφερσιμότητα
6. Άλλες απαιτήσεις

Προβλήματα:

- επικοινωνίας με τον πελάτη
- προτύπων (λύση τα UML)
- γλώσσας
- οικονομικά

Ισχύει:

requirements ≠ solution

solution ≫ ∑ requirements

Πρέπει:

$\sum valueOfProblemsSolvedAndDelivered \gg$

$\sum costOfProblemsCaused + \sum costOfSolutionComponents$

Αρχιτεκτονικές οπτικές:

- logical view: λειτουργικότητα σε high level
- physical view: component topology μετά το deployment
- development view: programming implementation
- process view: runtime behavior
- scenarios/use-case view: έλεγχοι χρηστικής απόδοσης

Αρχιτεκτονικά πρότυπα: Γενικές και επαναχρησιμοποιήσιμες λύσεις σε κοινά προβλήματα αρχιτεκτονικής.

Αρχιτεκτονικά στυλ: συγκεκριμένη μέθοδος με αξιοπρόσεκτα χαρακτηριστικά.

- εφαρμόζεται σε δεδομένο περιβάλλον
- οριοθετεί πιθανά συστήματα που μπορεί να προκύψουν
- παρέχει τα ποιοτικά χαρακτηριστικά του κάθε συστήματος

Χαρακτηριστικά καταναμεμημένων συστημάτων:

- Consistency
- Availability
- Network partition
- Latency
- Throughput
- Scalability

Consistency (C):

Συνέπεια δεδομένων, δηλ κάθε read βλέπει το πιο πρόσφατο write.

Οι ACID συναλλαγές είναι ακόμα πιο αυστηρές από αυτό.

Συγκεκριμένα:

- Atomicity
- Consistency
- Isolation
- Durability

Availability (A):

Κάθε request παίρνει κάποιο response (όχι απαραίτητα το πιο πρόσφατο write). Γενικά απαιτούνται replicas των δεδομένων.

Network Partition (P):

Αν σπάσει το δίκτυο σε επιμέρους συνεκτικές συνιστώσες, που δεν επικοινωνούν μεταξύ τους. Γίνεται αν έχουμε κάποιο network failure.

Latency (L):

Καθυστέρηση απόκρισης συστήματος.

Throughput:

Πλήθος αιτημάτων που ικανοποιεί ταυτόχρονα το σύστημα ανά πάσα χρονική στιγμή.

Θεώρημα CAP: $P \Rightarrow \neg(A \wedge C)$

Αν πέσει partition στο δίκτυο, ή σώνω το availability ή το consistency, και τα δύο όχι.

Θεώρημα PACELC: $[P \Rightarrow \neg(A \wedge C)] \wedge [(\neg P) \Rightarrow \neg(L \wedge C)]$

Ό,τι είχα πριν + αν δεν έχω partition και το δίκτυο πάει καλά, τότε ή μειώνω latency ή έχω καλό consistency, όχι και τα δύο.

Κατηγορίες καταναμεμημένων:

- $[P \Rightarrow A] \wedge [(\neg P) \Rightarrow L]$ PA/EL
- $[P \Rightarrow C] \wedge [(\neg P) \Rightarrow C]$ PC/EC (ACID databases)
- $[P \Rightarrow C] \wedge [(\neg P) \Rightarrow L]$ PC/EL

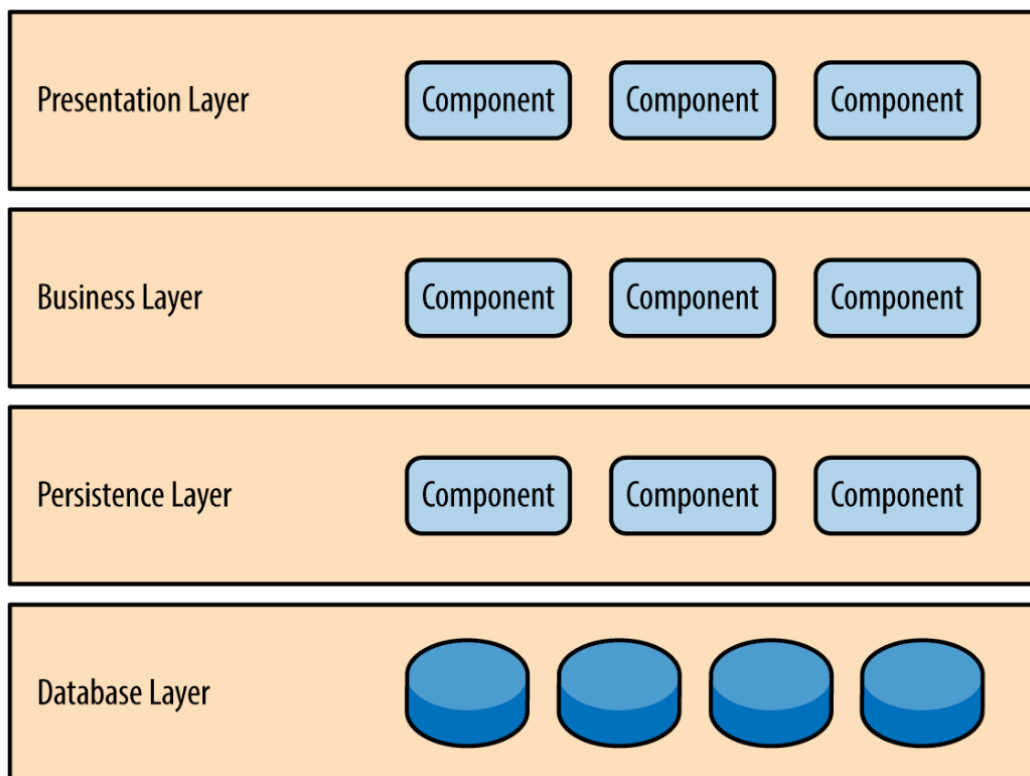
Scalability:

- scale out/in = αύξηση ή μείωση κόμβων
- scale up/down = αύξηση ή μείωση πόρων ανά κόμβο

Αρχιτεκτονικά πρότυπα:

- Client-server:
 - server-based
 - 1 server – N clients
 - επικοινωνούν με κάποιο πρωτόκολλο για υλοποίηση εφαρμογής
 - www, ssh, ftp...
- Peer-to-peer (P2P):
 - δίκτυο ομότιμων κόμβων
 - όλοι είναι client & server ταυτόχρονα
 - επικοινωνούν με κάποιο πρωτόκολλο για υλοποίηση εφαρμογής
 - file-sharing networks, blockchain, cryptocurrencies...
- Component based:
 - συστατικά αλληλεπιδρούν μέσω interfaces
 - κάθε component παρέχει μία διεπαφή και απαιτεί άλλες
 - loose coupling & separation of concerns
 - Application server = το λογισμικό που κάνει host όλες αυτές τις λειτουργίες
- Layered/N-tier

Layered/N-tier



- server-based
 - λογική και/ή φυσική αρχιτεκτονική
 - εφαρμογές διαδικτύου
 - υπάρχουν έτοιμα frameworks
- Model-View-Controller (MVC):
 - διαχωρισμός ενδιαφερόντων
 - Controller:

- user input, request/response, επιβλέπει τα άλλα
- Model:
 - data model, business logic
- View:
 - data display
 - υπάρχουν κι εδώ frameworks
- Master-Slave / Master-Replica:
 - N slaves, 1 master
 - master = authority, slaves = redundancy
 - βοηθάει σε διαθεσιμότητα, ισομοιρασμό φορτίου κτλ
 - data replication για κάθε slave
 - load balancing, δλδ ο master έχει κάποιο dispatching logic
- Share-Nothing:
 - ανεξάρτητοι & αυτοτελείς κόμβοι
 - δεν μοιράζονται κανέναν απολύτως πόρο
 - sharding = δεδομένα σπάνε οριζόντια στους επιμέρους κόμβους
 - horizontal scalability – απλά προσθέτω κόμβους και το σύστημα βελτιώνεται
 - εφαρμόζεται σε πολλά NoSQL συστήματα
- Message-Driven/Publish-Subscribe:
 - Loose coupling
 - publisher (producer) = αποστολή μηνυμάτων
 - subscriber (consumer) = λήψη
 - topics (channels) = τύποι μηνυμάτων
 - message bus (broker) = δρομολόγηση μηνυμάτων (με ρυθμιζόμενες εγγυήσεις λειτουργιών)
 - Εφαρμογές:
 - middleware ετερογενών συστημάτων
 - υψηλή απόδοση/κλιμάκωση σε κατανεμημένα
 - ΑΛΛΑ: δύσκολη αλλαγή δομής μηνυμάτων
- Event-Driven:
 - events & event handlers
 - implicit invocation / inversion of control

(The idea behind implicit invocation is that instead of invoking a procedure directly, a component can announce (or broadcast) one or more events. Other components in the system can register an interest in an event by associating a procedure with the event. When the event is announced the system itself invokes all of the procedures that have been registered for the event. Thus an event announcement implicitly causes the invocation of procedures in other modules. - Wikipedia)
 - event thread/loop
 - GUI & server side εφαρμογές
- Pipeline/Pipe-filter
 - data streams, pipes and filters
 - συναρτησιακός προγραμματισμός
 - επαναχρησιμοποίηση & παραλληλοποίηση

Δομημένη ανάλυση και σχεδίαση:

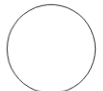



- στοιχεία εφαρμογής = δεδομένα και μετασχηματισμοί
- ταυτοποίηση δεδομένων (λεξικά, ορολογία)
- εύκολη μετάβαση σε υλοποίηση δομημένου προγραμματισμού (όχι go-to, και functions)

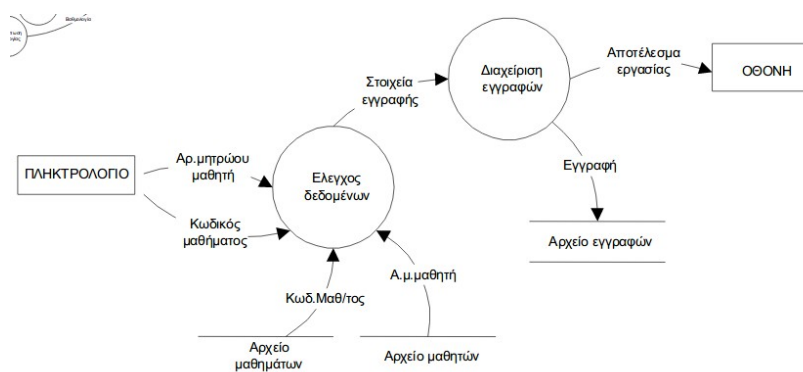
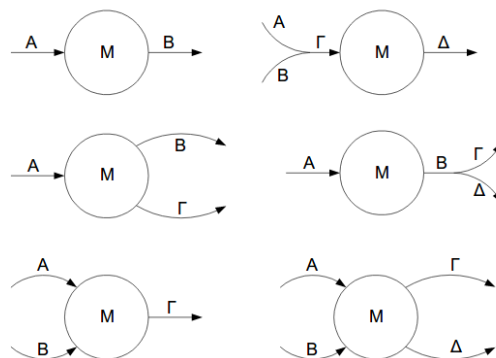
Αρχές:

- δεδομένα ανεξάρτητα μονάδων διαχείρισής τους
- Περιγραφή σε:
 - εννοιολογικό επίπεδο (RED)
 - κατασκευαστικό επίπεδο (RDB)
- συμπεριφορά μονάδων λογισμικού προκύπτει κυρίως από περιγραφή λειτουργικών απαιτήσεων
- μονάδες ανταλλάσσουν δεδομένα για την επίτευξη του στόχου τους
- επιπλέον περιγραφές όπου απαιτείται επεξήγηση

Διάγραμμα ροής δεδομένων:

- δίκτυο που “ρέουν” δεδομένα μέσω μετασχηματισμών
- διαδοχικά μέχρι να βγει το αποτέλεσμα

Συμβολισμοί διαγραμμάτων ροής δεδομένων	
	Διαδικασία / μετασχηματισμός δεδομένων
	Εξωτερική πηγή ή αποδέκτης δεδομένων
	Ροή δεδομένων
	Αποθήκη δεδομένων



Εννοιολογικά μοντέλα δεδομένων (ERD): οντότητες και συσχετίσεις ανάμεσά τους.

Διαγράμματα μετάβασης καταστάσεων:




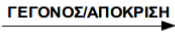

- Γεγονός (στιγμιαία εξωτερική αλλαγή στο περιβάλλον)
- Απόκριση (λειτουργία λόγω γεγονότος)
- Κατάσταση (λογισμικό κάνει wait για γεγονός)

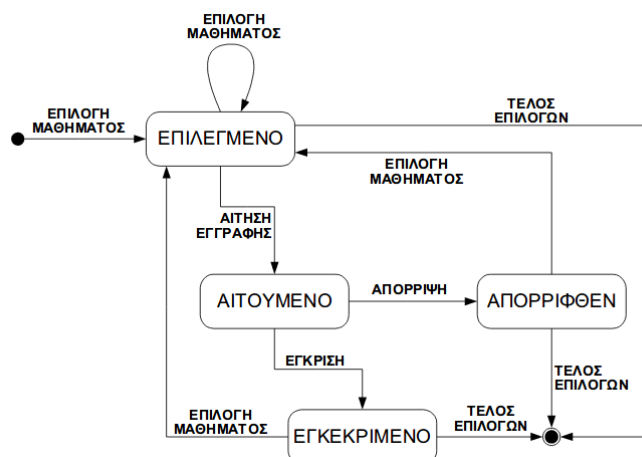
Κύκλος: Κατάσταση – Γεγονός – Απόκριση – Νέα κατάσταση

Το διάγραμμα έχει μία κατάσταση αρχής και μία κατάσταση τέλους, περιγράφει όλο ή τμήμα του συστήματος.

Έχει νόημα να το φτιάξουμε όταν:

- αποσαφηνίζει τη συμπεριφορά του λογισμικού
- διευκολύνει την περιγραφή της υλοποίησής του
- περιγράφει την κατάσταση των δεδομένων

Συμβολισμοί διαγραμμάτων μετάβασης καταστάσεων	
	Κατάσταση
	Κατάσταση έναρξης
	Κατάσταση τέλους
	Μετάβαση σε άλλη κατάσταση / λειτουργία που εκτελείται
	Μετάβαση στην ίδια κατάσταση / λειτουργία που εκτελείται



Λεξικό δεδομένων = πίνακας ή ΒΔ, που για κάθε οντότητα περιέχει:

- ονομασία
- βοηθητικές ονομασίες (aliases)
- πού (από ποιους μετασχηματισμούς) χρησιμοποιείται
- πώς χρησιμοποιείται
- τι περιέχει
- όρια τιμών
- αρχική τιμή
- λοιπά στοιχεία

Ανάλυση λογισμικού: τι θέλουμε να κάνει και τι χαρακτηριστικά να έχει.

Σχεδίαση λογισμικού: πώς θα το πετύχουμε και με τι απόδοση.

Για τη σχεδίαση υπάρχουν:

- function-oriented
 - process-oriented
 - data-oriented
- object-oriented

μεθοδολογίες.

Εργασίες δομημένης σχεδίασης:

- Αρχιτεκτονική σχεδίαση:
 - διάγραμμα δομής προγράμματος
 - διάγραμμα διάταξης
- Σχεδίαση διεπαφών
- Λεπτομερής σχεδίαση μονάδων (ψευδοκώδικας)
- Σχεδίαση δεδομένων (σχεσιακό μοντέλο)

Αρχιτεκτονική σχεδίαση:

- ορισμός υποσυστημάτων
- διαδοχικά επίπεδα λεπτομέρειας
- γενικό περίγραμμα εφαρμογής

Σχεδίαση διεπαφών:

- Επικοινωνία
 - μεταφορά ελέγχου ροής
 - μεταφορά δεδομένων με παραμέτρους
- Καθορίζεται:
 - τύπος παραμέτρων κάθε μονάδας
 - φύση επικοινωνίας μεταξύ υποσυστημάτων
 - λεπτομέρειες επικοινωνίας με εξωτερικές συσκευές
 - UI

Λεπτομερής σχεδίαση μονάδων:

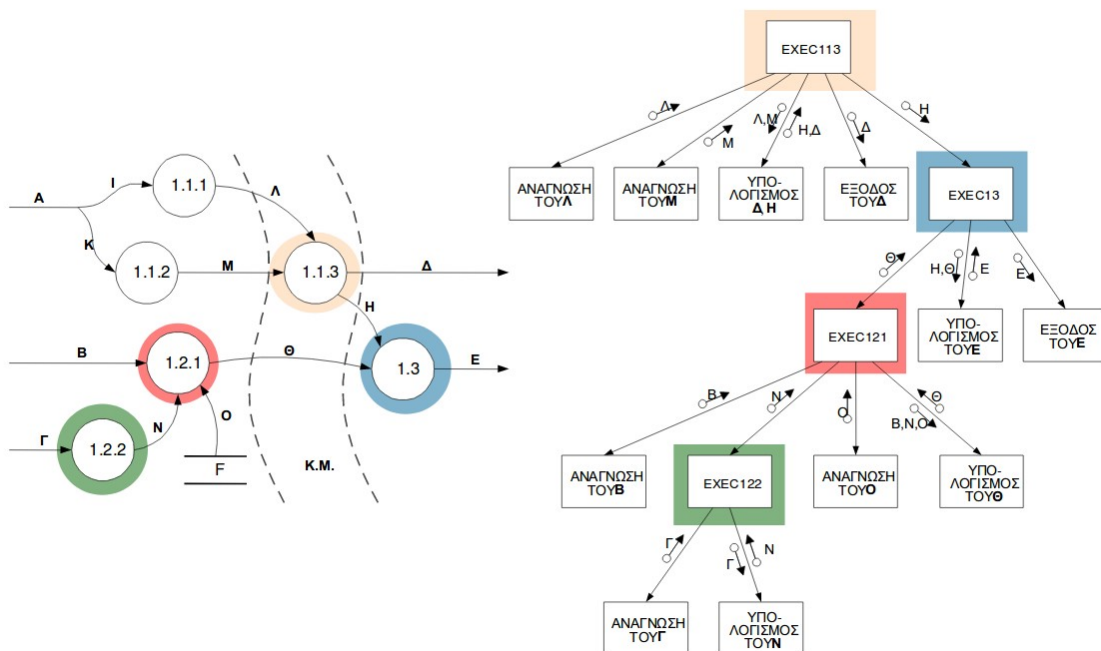
- εστίαση στο εσωτερικό κάθε μονάδας
- εύκολο να υλοποιηθεί προγραμματιστικά
- Υπόψη:
 - όλα τα προϊόντα σχεδίασης μέχρι στιγμής
 - όλα τα σημεία προδιαγραφών που περιγράφουν την επιθυμητή συμπεριφορά

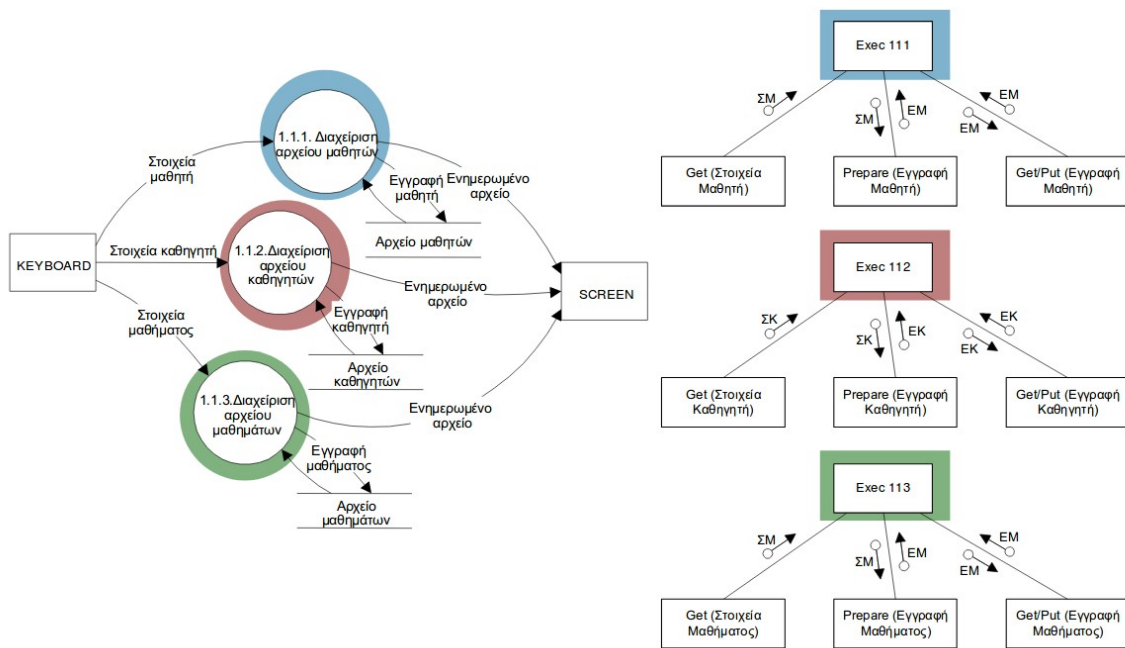
Σχεδίαση δεδομένων:

- επαλήθευση μοντέλου οντοτήτων – συσχετίσεων
- βελτιστοποιήσεις + κανονικοποίηση
- καθορισμό τύπων πεδίων
- καθορισμός δεικτών και όψεων
- εξαρτάται και από DBMS

Διάγραμμα δομής:

- εντοπισμός κεντρικού μετασχηματισμού/κέντρου δοσοληψιών
- απεικόνιση σε διάγραμμα δομής
- παραγοντοποίηση (επανάληψη για δεξί και αριστερό τμήμα του κέντρου)
- συνένωση τμημάτων





```

/*-----*/
PROCEDURE Exec111
/*-----*/
LOCAL VAR στοιχεία_μαθητή, εγγραφή_μαθητή
Αρχικοποίησε στοιχεία_μαθητή, εγγραφή_μαθητή

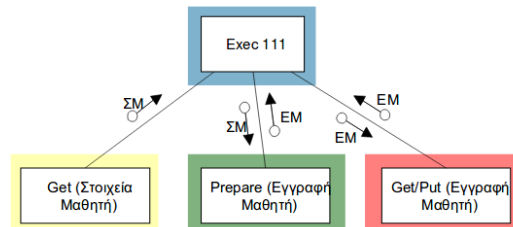
WHILE στοιχεία_μαθητή <> κενό DO

    CALL Get_ΣΜ(στοιχεία_μαθητή)
    IF στοιχεία_μαθητή <> κενό THEN
        CALL Prepare_ΣΜ(στοιχεία_μαθητή, εγγραφή_μαθητή)
        CALL Put_EM(εγγραφή_μαθητή)
    END_IF

END_WHILE

END_PROCEDURE

```



Αντικειμενοστραφής σχεδιασμός: Αντικείμενο := βασική μονάδα λογικών abstractions

Τεχνικό χρέος:

Κόστος πρόσθετης δουλειάς λόγω επιλογής quick-and-dirty λύσεων έναντι σωστών υλοποιήσεων.

Αντικείμενα:

- encapsulation
- state
- behavior
- instantiation
- constructors
- self-reference

- this, self etc.
- message passing

Classification – Instantiation:

- σχέση αντικειμένου με κλάση του
- όλα τα στιγμιότυπα μιας κλάσης μοιράζονται κοινά χαρακτηριστικά
- κλάση = abstraction όλων των δυνατών της αντικειμένων

Aggregation – Decomposition:

- συνάθροιση επιμέρους εννοιών για σύνθεση νέας
- σχέσης μέρους-όλου

Generalization – Specialization:

- σχέση μεταξύ κλάσεων
- η γενική κλάση έχει την τομή των εξειδικεύσεων της
- η κληρονομικότητα υλοποιεί την εξειδίκευση

Κληρονομικότητα:

- αντικείμενο κληρονομεί πεδία ή/και μεθόδους των προγόνων του
 - είτε για εξειδίκευση
 - είτε για επαναχρησιμοποίηση

Private state/behavior = information hiding

Grouping – Individualization:

- ομαδοποίηση βάσει extensional, όχι intensional, χαρακτηριστικού
- πχ
 - αγαπημένα χρήστη
 - πιο δημοφιλή
 - search results

Polymorphism:

- είτε κοινή διεπαφή για αντικείμενα άλλων τύπων
- είτε ένα αντικείμενο έχει πολλούς τύπους ταυτόχρονα
- Κατηγορίες:
 - Universal
 - parametric (πχ generics/templates <T>)
 - inclusion (πχ implements {some_interface} σε Java)
 - Ad-hoc
 - overloading (πχ operator overloading)
 - coercion (πχ αυτόματα cast από int σε double λόγω πράξεων)

UML concepts:

- system
- model (abstraction of actual system)
- view (set of selected aspects of the model)

Το ποια views χρειάζονται ή όχι εξαρτάται από την εφαρμογή.

Stereotypes = a mechanism for extending UML's vocabulary.

UML modeling:

- use cases
- domain model
- design model
- Diagram types:
 - use case
 - class / package
 - interaction
 - sequence
 - collaboration / communication
 - activity / state transition
 - component / deployment

Use case diagrams:

- use cases: interactions with functionality
- actors: roles
- used during requirement specification for external behavior

Use cases – γκρουπάρουν απαιτήσεις για να τρέξουν.

Actors:

- user
- external system
- physical environment

Use case diagrams:

- Include
 - επαναχρησιμοποιήσιμη συμπεριφορά
 - βέλος προς το use case που την χρησιμοποιεί
- Extend
 - σπάνιο use case
 - βέλος προς αυτό που επεκτείνεται

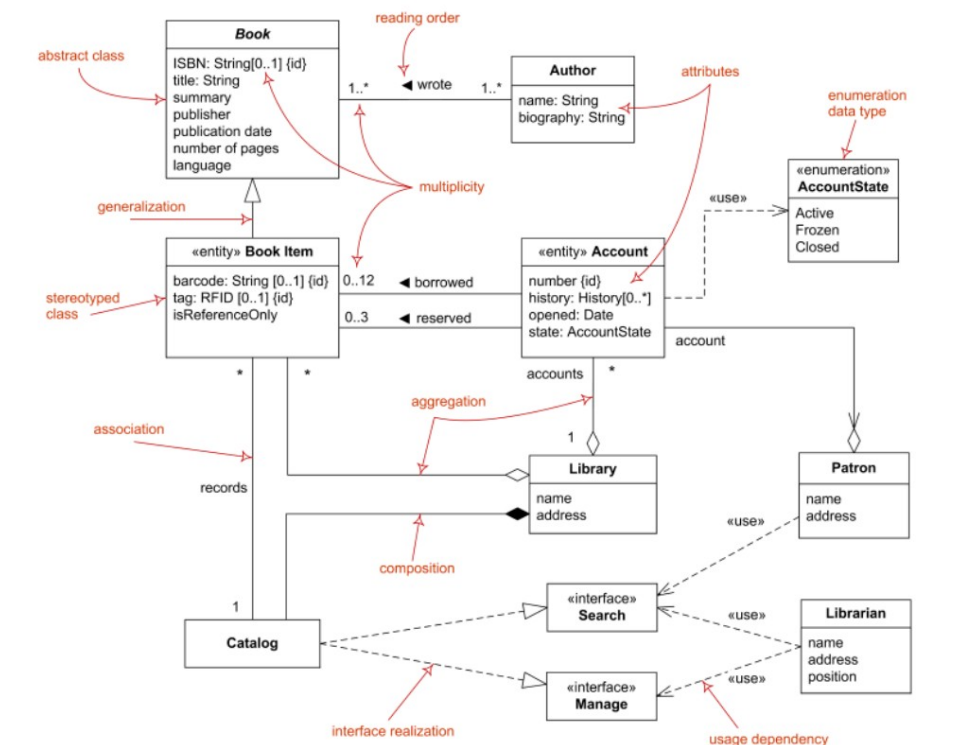
Χρήση για:

- αποσαφήνιση απαιτήσεων
- επικοινωνία με πελάτες
- δημιουργία test cases

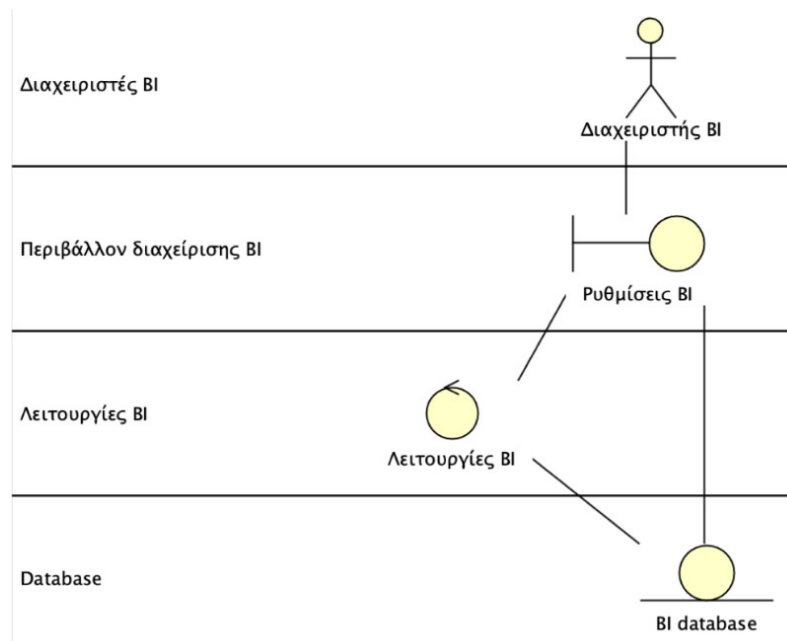
Class diagrams:

- περίληψη συστήματος δείχνοντας κλάσεις και συσχετίσεις μεταξύ τους
- στατικό, περιγράφει τις συνδέσεις, όχι τι κάνουν όταν ενεργοποιούνται
- δείχνει και attributes/operations για κάθε κλάση
- 3 Perspectives:
 - conceptual (εντελώς αφαιρετικό, ανεξάρτητο γλώσσας υλοποίησης)
 - specification (interface level)
 - implementation (full specs)
- Notation:
 - private: -
 - public: +
 - protected: #
 - static: underlined
 - abstract: *italics*
 - association: line/arrow
 - aggregation (association + one class belongs to a collection): line w/ empty diamond
 - composition (aggregation++): same, but filled diamond
 - inheritance: is-a, arrows pointing to parent

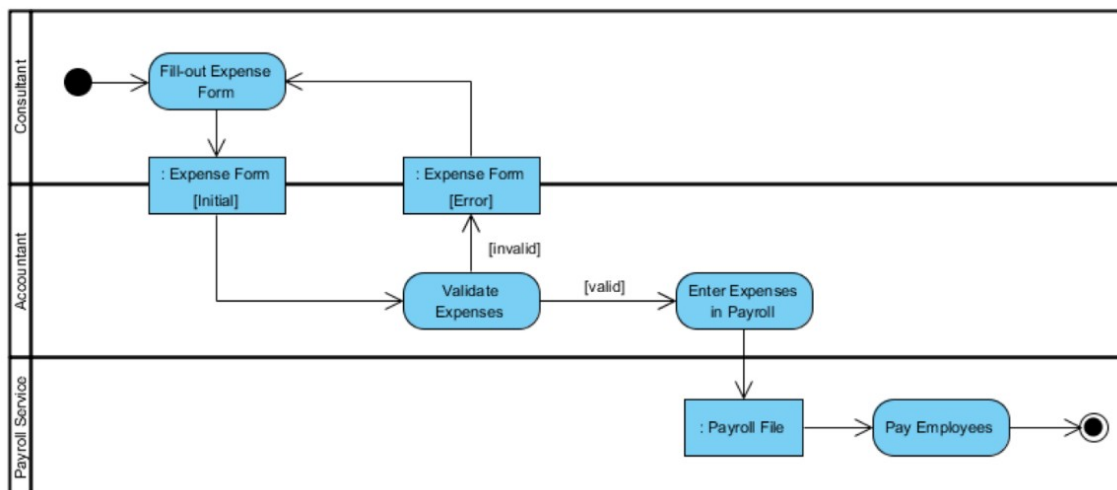
Έχει και multiplicities, δηλ πόσες οντότητες συμμετέχουν σε ένα association, νουμεράκια πάνω από τα βέλη (βάζει * για 0,1,2... ή και 1..* για τουλάχιστον 1).



- message passing – collaboration diagram
 - object links – συνεχείς γραμμές μεταξύ αντικειμένων που αλληλεπιδρούν
 - εμπεριέχουν και τα μηνύματα (με κατεύθυνση) που ανταλλάσσονται



- activity diagrams (?)
 - activity / action
 - flow (control / object)
 - initial / final node
 - decision / merge
 - fork / join



- UML state diagrams
 - δείχνουν συμπεριφορά ενός αντικειμένου μόνο από αρχή μέχρι τέλος
 - περιγράφει όλες τις πιθανές μεταβάσεις του
 - Συμβάντα μπορεί να είναι:
 - signal
 - call
 - time
 - change

Κύκλος μετάβασης κατάστασης: source state – event occurs – action performed – element enters new state

- Component/Package diagrams
 - εξηγούν δομή συστήματος
 - συλλογή από κλάσεις
- Component: όλα τα elements δείχνουν και τα private, με κάποια στο public interface
- Package: δείχνει μόνο τα public

- Operation – Deployment diagram
 - περιέχει σαν μαύρα κουτιά components ή packages
 - περιγράφει και διασυνδέσεις hardware – software

AJAX (Asynchronous Javascript And ~~XML~~ JSON):

στέλνει HTTP ασύγχρονα σε ξεχωριστό thread και έχει callback hook για το αποτέλεσμα.

Observable:

- push paradigm
- observer ενημερώνεται για αλλαγή τιμής του observable
- σαν να είναι event

MVVM:

- View: UI
- Model: data
- ViewModel (Presenter/ViewController):
 - data bindings (model elements ~ UI elements)
 - change notifications

To ViewModel ενθυλακώνει model elements ως observables και θέτει ως observers τα UI elements (bind).

JSON – Mimetype = application/json

JSON – benefits:

- απλό
 - αναγνώσιμο από άνθρωπο
 - επεξεργάσιμο από ό,τι γλώσσα θέλω
 - αναπαριστά εύκολα ό,τι δομές χρειάζονται οι εφαρμογές
- σταθερό
 - δεν έχει εκδόσεις
 - δεν αναμένεται να αλλάξει το συντακτικό
 - απλοί και γρήγοροι parsers
- ταχύτερο και απλούστερο από XML
- υποστηρίζει UNICODE

JSON – τύποι:

- πρωτογενείς
 - string (διπλά εισαγωγικά με ‘\’ escapes)
 - number (signed δεκαδικός, ακρίβεια double)
 - boolean (true ή false)
- null (duh)
- σύνθετοι
 - array (μέσα σε [], χωρίζονται με ‘,’)
 - object (μέσα σε { }, σαν dictionary Python)

Service-oriented architecture (SOA):

- Υπηρεσίες με επικοινωνία πρωτοκόλλου:
 - κατανεμημένες
 - αυτοτελείς
 - loosely-coupled
 - ανεξάρτητες τεχνολογιών υλοποίησης
 - ανεξάρτητες κατασκευαστή
- Σύνθεση εφαρμογής μέσω ολοκλήρωσης υπηρεσιών (integration).

Αρχές SOA:

- service contract
- metadata
- composition
- autonomy
- discovery
- reusability

Representational State Transfer (REST):

Γενικό αρχιτεκτονικό στυλ για λειτουργία κατανεμημένων συστημάτων, οριοθετεί αρχές, περιορισμού και βασικές λειτουργίες, χωρίς εξάρτηση από γλώσσες προγραμματισμού, πρωτόκολλα επικοινωνίας ή είδη δεδομένων.

4 έννοιες:

- Resources
- Representations
- Requests
- Responses

6 αρχές:

- client-server: διαχωρισμός ενδιαφερόντων μεταξύ client και server και ομοιόμορφη διεπαφή ανάμεσά τους
- stateless: καμία πληροφορία δεν αποθηκεύεται από τον server για το state του client όταν επικοινωνούν (μόνο user *session* data)
- cacheable: client μπορεί να αποθηκεύει προσωρινά κάποια responses, όποια του επιτρέπει ο server (πχ μέσω HTTP headers)
- layered system: client συνδέεται με end server είτε με ενδιάμεσο χωρίς καμία διαφορά, και οι ενδιάμεσοι προσθέτουν στρώσεις λειτουργικότητας (πχ caching, authentication...)
- uniform interface: resource identification, διαχείριση πόρων μέσω representations – HATEOAS (Hypermedia As The Engine Of Application State)
- code on demand (προαιρετικό): client εκτελεί κώδικα από server δυναμικά

Σύνοψη:

- client sends request with URIs
- resources διαχωρίζονται από representations (representation = JSON/XML/... , resource = το actual entity μέσα στον server)
- client έχει representation μαζί με metadata = μπορεί να τροποποιήσει το αντίστοιχο resource
- responses του server έχουν όλη τη πληροφορία που απαιτείται για χειρισμό client-side
- HATEOAS: client επικοινωνεί μέσω hypermedia (πχ hyperlinks), οπότε δεν χρειάζεται να ξέρει κάποιο interface εκ των προτέρων

Οφέλη:

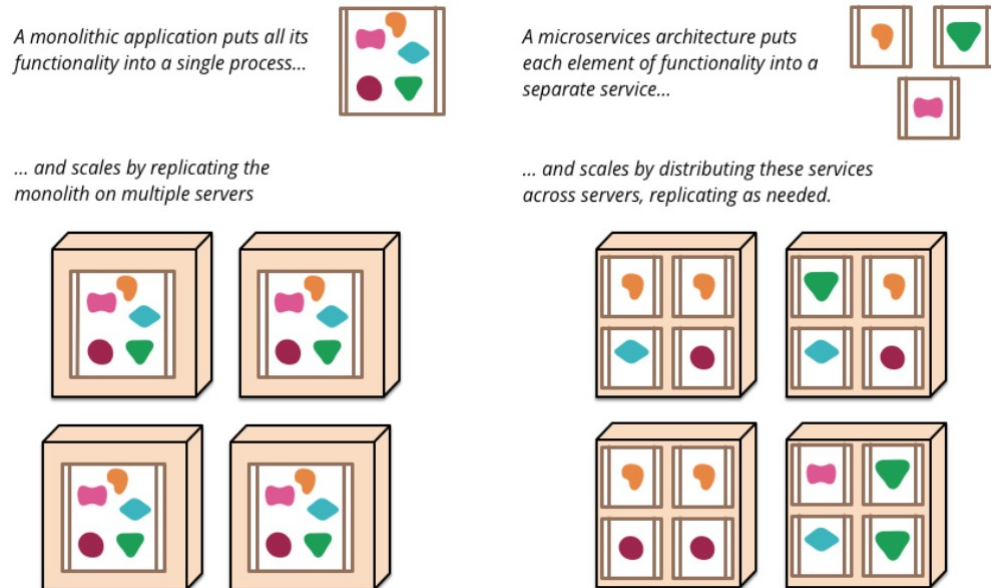
- απόδοση
- κλιμάκωση
- απλότητα
- επεκτασιμότητα
- αξιοπιστία

RESTful Web Service (API):

- HTTP base URL (REST endpoint)
- ≥ 1 MIMEType για representations
- HTTP requests για collections:
 - GET: εμφάνιση λίστας με elements της συλλογής
 - PUT: αντικατάσταση υπάρχοντος collection με νέο
 - POST: προσθήκη νέου element
 - DELETE: διαγραφή υπάρχοντος collection

- HTTP requests για elements:
 - GET: ανάκτηση μιας αναπαράστασης του item
 - PUT/PATCH: τροποποίηση item
 - POST: δε χρησιμοποιείται ευρέως για item
 - DELETE: διαγραφή item

MicroServices = SOA + Unix Principles + Agile + DevOps



Χαρακτηριστικά:

- components via services
- organized around business capabilities
- products, not projects
- smart end-points, dumb pipes
- decentralized governance
- decentralized data management
- infrastructure automation
- design for failure

Testing & Validation:

- Testing
 - Reviews (static)
 - Static analysis
 - Test cases (dynamic)

Επαλήθευση = παράγουμε το λογισμικό σωστά?

Επιβεβαίωση = παράγουμε το σωστό λογισμικό?

Bugs:

- Critical
- Major
- Minor
- Trivial

Test life-cycle:

- επισκόπηση απαιτήσεων
- σχεδιασμός δοκιμασιών
- καθορισμός περιβάλλοντος δοκιμασιών
- εκτέλεση δοκιμασιών
- παραγωγή αναφορών

Test plan: ένα ή περισσότερα έγγραφα, περιγράφει τι και πώς θα ελεγχθεί, ποια είναι τα παραδοτέα και πόσο χρόνο και χρήμα θα κοστίσει. Επίσης προγραμματίζει σε επίπεδο προσωπικού τη διαδικασία που θα εκτελεστεί.

Test case:

Συνθήκες/μεταβλητές που καθορίζουν ορθή συμπεριφορά, γράφεται σε κάποιο εργαλείο.

Test scripts:

Αυτοματοποίηση tests – CI

Μέθοδοι testing:

- black box (behavioral): public interface tests
- white box (structural): internal state tests
- gray box (συνδυασμός)

Επίπεδα δοκιμασιών (μικρό προς μεγάλο):

- unit test
 - ελάχιστο στοιχείο που μπορεί να ελεγχθεί
 - white box testing
 - πιθανές είσοδοι, test αν κρασάρουν ή βγάζουν λάθος τιμές
- integration test
 - διαλειτουργικότητα ενοτήτων (όχι απαραίτητα units)
 - black/white/gray (διαλέγουμε)
- system test
 - όλο το σύστημα ως προς το λειτουργικό περιβάλλον
 - δικτύωση, υπηρεσίες, αποθήκευση, υλικό κτλ
- acceptance test
 - customer/user acceptance
 - black box
 - ad hoc

Τύποι δοκιμασιών:

- Smoke (integration, system, acceptance)
 - να δουλεύει έστω ένα πράγμα
 - καλύπτουμε οριζόντια κάποιες βασικές λειτουργίες, μπαίνουμε σε βάθος μετά
 - προχωράμε σε βάθος ή δεν είναι OK το σύστημα?
- Funcional (unit, system, acceptance)
 - επαλήθευση συμβατότητας με προδιαγραφές & απαιτήσεις
 - εκτελούνται από “χρήστες”
 - domain-driven design & behavior-driven development
 - automated tests/specs
- Usability testing (system, acceptance)
 - ad-hoc
 - A/B testing (μέτρηση αποτελεσματικότητας UI/UX)
- Security testing (δύσκολο)
- Performance / load testing
 - load: stress, spike, soak testing
 - max cap: concurrent users, throughput
 - acceptance response time: latency
 - stress: έλεγχος σε συνθήκες πέρα από το νορμάλ
 - spike: έλεγχος για burst από χρήστες που μπαίνουν ταυτόχρονα
 - soak: έλεγχος με απλές συνθήκες για μεγάλο χρονικό διάστημα
- Regression testing:
 - όταν έρχεται νέα λειτουργικότητα, είναι πραγματικά backwards compatible?
- Bebugging:
 - βάζω bug στον κώδικα και βεβαιώνομαι ότι οι δοκιμασίες τα πιάνουν (test the tester)
- Mutation testing:
 - αλλαγή σε λογικά ισοδύναμο κώδικα με άλλη σύνταξη – τι γίνεται με τα tests σε αυτήν την περίπτωση?

Διαδοχικοί server για artifact testing/releases:

- developer workstation (local tests)
- testing server (CI)
- staging server (user acceptance tests)
- production server (production use)

Κατασκευαστικά πρότυπα σχεδίασης:

- Singleton:
 - διασφαλίζει ότι μία κλάση έχει 1 το πολύ ενεργά στιγμιότυπα ανά πάσα χρονική στιγμή
 - πχ μόνο ένα data access handler
 - private constructor (άρα έχω θέμα στην κληρονομικότητα)
 - don't overuse this

- Factory Method (virtual constructor)
 - extra level of indirection
 - δλδ κάνω virtual τον constructor, οπότε μία κλάση που κληρονομεί την αρχική τον υλοποιεί με custom λογική
 - μία κλάση μπορεί να μην θέλει να ξέρει τι στιγμιότυπά της θα δημιουργηθούν
 - κάνει delegate αρμοδιότητες σε υποκλάσεις της
 - μειώνεται η πιθανότητα σφάλματος στον constructor λόγω κακού initialization
- Abstract Factory (interface)
 - ορίζω ένα interface, όσες κλάσεις το υλοποιούν έχουν τα απαραίτητα χαρακτηριστικά για να τις θεωρώ “παρόμοιες”
 - είναι ανεξάρτητη μία container κλάση του πώς δημιουργούνται ή αναπαρίστανται τα στοιχεία που περιέχει
 - επιτρέπεται να γίνει το specification του πραγματικού τύπου κάτω από το interface στο runtime
- Builder
 - απλουστεύει τη δημιουργία σύνθετων αντικειμένων
 - λιγότερα argument στον constructor μιας κλάσης
 - κρατάει κάποιες default τιμές αν ο χρήστης δεν τις προσδιορίσει όλες ρητά
 - η διαδικασία “κατασκευής” πρέπει να υποστηρίζει διαφορετικές αναπαραστάσεις για τα υπό κατασκευή αντικείμενα.
- Object pool
 - κρατάει ζωντανά cached αντικείμενα που έχουν δύσκολους/ακριβούς constructors
 - thread pools, connection pools etc
 - borrow & return μοτίβο
 - επαναχρησιμοποίηση runtime οντοτήτων
 - αποφυγή επιθέσεων DoS
 - αν είναι μικρό το pool, φράσσεται η απόδοση του συστήματος
- Prototype (cloneable)
 - δημιουργία νέου στιγμιότυπου κλωνοποιώντας ένα υπάρχον
 - εναλλακτικά με copy constructors

Software quality assurance:

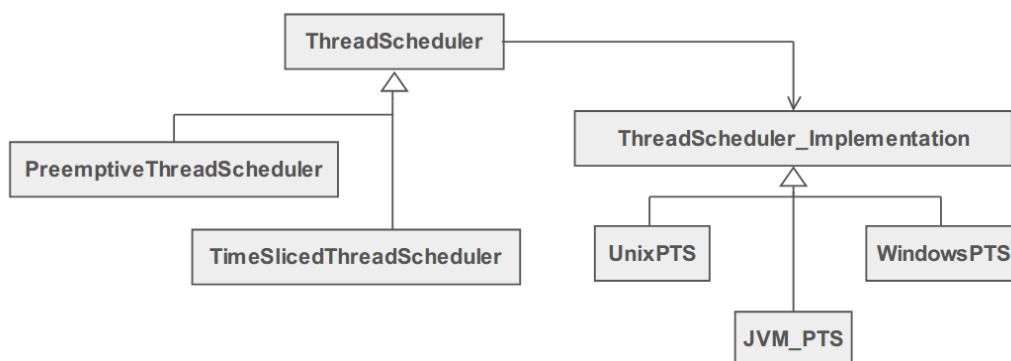
- Introduction
 - scope
 - revision process
 - abbreviations
 - distribution list
 - reference
- Project description
- Project management
 - roles
 - team structure
 - scheduling
 - QA audits
- Deliverables

- Identification and classification of deliverables
- Software documentation templates
 - requirements
 - design/architecture
 - user manual
 - test plan
 - test reports
- Software development process
 - life cycle model
 - development tasks
 - requirements engineering process
 - design/architecture process
 - source code requirements
- Subcontractors
 - testing and acceptance process
 - documentation requirements
 - other requirements
- Internal quality audits
 - process
 - documentation
- Development team update

ISO/IEC/IEEE standards - kill me now... : ^)

Δομικά πρότυπα σχεδίασης:

- Adapter (wrapper):
 - καθιστά την εσωτερική του κλάση συμβατή με κάποιο interface
 - δύο πρώην ασύμβατα συστατικά τώρα δουλεύουν μαζί
- Bridge:
 - διαχωρίζει interface από class
 - έχω ένα super interface, αριστερά βάζω άλλα interface που το συγκεκριμενοποιούν, δεξιά τα class που υλοποιούν κάτι από αυτά



- έμφαση στην σύνθεση, όχι την κληρονομικότητα
- χρήση δύο ιεραρχιών, μία “δημόσια”, μία “ιδιωτική”
- “δημόσιες” αφαιρέσεις χρησιμοποιούν “ιδιωτικές” υλοποιήσεις
- οι ιεραρχίες εξελίσσονται ανεξάρτητα

- Composite
 - κλάση-χρήστης χειρίζεται σύνθετα αντικείμενα ομοιόμορφα και απλά
 - πχ έχω μία συλλογή από class που ανοίγουν αρχεία ή URL, γενικά ένα resource
 - φτιάχνω ένα άλλο class, που παίρνει από τον χρήστη το όνομα ενός resource και κρατάει εσωτερικά ένα instance από όλες τις μικρές κλάσεις που είχα ορίσει πριν
 - τις δοκιμάζει όλες μέχρι να βρει μία που πετυχαίνει
 - ο χρήστης τελικά βλέπει ανοιχτό το resource
 - δεν ξέρει πώς ανοίχτηκε, ούτε τον νοιάζει
- Decorator
 - προσθέτει δυνατότητες σε ένα αντικείμενο χωρίς κληρονομικότητα
 - στην Java πετάω new μέσα σε new μέσα σε new
 - συγκεκριμενοποιώ στον constructor τι θέλω να υλοποιεί, αλλά ο τελικός τύπος είναι αυτός του πρώτου super class του οποίου τον constructor καλώ
 - πιο ευέλικτο από subclassing
 - recursive wrapping
- Facade
 - απλό interface για δύσκολο υποσύστημα
 - έχω μία πολύ περίπλοκη κλάση με 100 public μεθόδους
 - φτιάχνω μία άλλη που κρατάει ένα στιγμιότυπο της πρώτης μέσα της
 - αφήνω public μόνο 1-2 μεθόδους που θέλει ο χρήστης, που τις υλοποιώ πχ καλώντας κατευθείαν τις αντίστοιχες του εσωτερικού, πολύπλοκου αντικειμένου
- Proxy
 - ενδιάμεσος για ένα άλλο αντικείμενο
 - προσθέτει λειτουργίες, πχ:
 - lazy initialization (virtual proxy)
 - access proxy
 - remote proxy
 - cache proxy
 - logging proxy