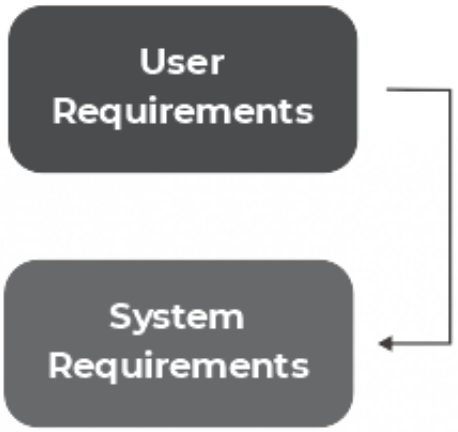
Niciun proces de dezvoltare software nu poate fi imaginat fără etapa numită specificaţie software. Este vorba de una dintre cele patru etape de bază ale dezvoltării software, pe care le au toate modelele de dezvoltare. Principalul subiect al specificaţiei software îl reprezintă cerinţele software. Prin urmare, etapele introductive ale dezvoltării tuturor modelelor de dezvoltare sunt axate pe lucrul cu cerinţele software, în primul rând pe colectarea şi procesarea acestora. Aceasta este cea mai importantă etapă de lucru la un proiect software, care precede codarea şi programarea. În această lecţie veţi avea ocazia să vă familiarizaţi cu toate aspectele lucrului cu cerinţele software în procesul de dezvoltare a software-ului.

Ce sunt cerinţele software şi cine le defineşte?

Cerinţele software (*software requirements* în engleză) sunt o descriere a sistemului care trebuie construit cu toate caracteristicile şi limitările aşteptate. Acestea sunt definite de toate părţile interesate care participă la dezvoltarea proiectului. În funcţie de natura proiectului, la crearea cerinţelor software pot participa diferite entităţi. Utilizatorii finali, clienţii, finanţatorii, inginerii care participă la dezvoltare - acestea sunt doar câteva exemple de entităţi care pot participa la crearea cerinţelor.

Aproape întotdeauna, crearea cerinţelor software începe cu utilizatorii cărora le este destinat viitorul produs software. Cerinţele definite de utilizatori se numesc cerinţe de utilizator (*user requirements* în engleză). Cerinţele de utilizator sunt primite de echipa de dezvoltare şi extinse cu cerinţe legate de aspectele tehnice ale dezvoltării. Aşa sunt create cerinţele de sistem (*system requirements*, engleză). Cerinţele de utilizator şi de sistem sunt două grupuri de bază de cerinţe (imaginea 8.1.).



Imaginea 8.1. Cerinţe de utilizator şi de sistem

Cerinţele de utilizator sunt formate de utilizatori care nu înţeleg modul în care va fi realizat software-ul din punct de vedere tehnic. Prin urmare, cerinţele lor se referă la comportamentul extern al sistemului.

Cerinţele de sistem sunt o versiune extinsă a cerinţelor de utilizator. Cerinţele de utilizator pot fi observate ca un punct de plecare pe baza cărora echipa de dezvoltare creează cerinţe de sistem. Cerinţele de sistem descriu modul în care sistemul va implementa cerinţele de utilizator.

Caracteristicile cerinţelor software

Cerinţele software stau la baza oricărui proiect care implică dezvoltarea software. Prin urmare, este foarte important ca fiecare cerinţă software să aibă anumite caracteristici, care o fac utilizabilă în ceea ce priveşte dezvoltarea ulterioară a produsului software.

Indiferent dacă este vorba de o cerinţă de utilizator sau de un sistem, fiecare cerinţă software ar trebui să fie:

- **corectă** - dacă cererea conţine un fapt, acel fapt trebuie să fie adevărat;
- **lipsită de ambiguitate** – trebuie să existe un singur mod în care cererea poate fi interpretată;
- **completă** – fiecare cerinţă software ar trebui să fie completă, adică trebuie să proceseze toate condiţiile şi posibilităţile care pot apărea în situaţia descrisă de cerere;
- **consecventă** – cerinţa software trebuie să fie clară şi simplă şi să nu conţină informaţii inutile care pot îngreuna interpretarea acesteia;
- **verificabilă** – persoanele care se ocupă de testare trebuie să poată verifica dacă o cerinţă specifică a fost implementată în mod corect;
- **necesară** - dacă niciuna dintre părţile interesate nu are nevoie de o cerinţă anume sau dacă înlăturarea acesteia nu ar schimba în niciun fel comportamentul sistemului, este o cerinţă care nu este necesară;
- **independentă** - înţelegerea unei cerinţe nu trebuie să fie condiţionată în niciun fel de înţelegerea altei cerinţe;
- **fezabilă** – cerinţa trebuie să corespundă limitărilor realiste în ceea ce priveşte timpul, banii şi resursele disponibile, cu alte cuvinte, fiecare cerinţă trebuie să fie realistă şi posibil de realizat în circumstanţele actuale.

Erorile sunt o parte integrantă a fiecărui produs software, ceea ce înseamnă practic că este aproape imposibil să găsiţi un program fără erori. Programatorii şi testerii sunt în permanenţă în căutarea unor astfel de erori şi se străduiesc să menţină numărul lor cât mai scăzut posibil în produsul final.

Este important să înţelegeţi că erorile software influenţează direct costul unui produs software. Fiecare eroare necesită o anumită perioadă de timp pentru a fi detectată şi corectată. În plus, greşelile pot fi făcute şi descoperite în diferite etape de dezvoltare a unui produs. Cu cât sunt descoperite mai devreme, cu atât costul corectării lor este mai mic. Tabelul 8.1. încearcă să transmită acest lucru.

Etapa	Coeficientul preţului

Fezabilitate/Analiză	1
Design	3–6
Codarea/Testarea unităţilor	10
Testarea în curs de dezvoltare	14–40
Testarea acceptabilităţii	30–70
Etapa operativă	40–1000

Tabelul 8.1. Preţurile corectării erorilor pe etape

Pe baza datelor din tabelul 8.1. puteţi observa că cea mai scumpă eroare software este cea care se face în etapa de analiză a cerinţelor, care este descoperită abia în etapa operativă. Corectarea sa poate costa de până la 1000 de ori mai mult decât corectarea din timpul etapei de analiză a cererii. Tocmai din cauza acestor fapte, o bună formulare a cerinţelor software stă la baza fiecărui produs software.

În cele din urmă, este important să spunem că lucrul cu cerinţele software diferă uşor în funcţie de modelul de dezvoltare aplicat. Lucrul cu cerinţele este cel mai simplu atunci când unul dintre modelele agile este folosit pentru dezvoltare. Datorită împărţirii proiectului într-un număr de unităţi mai mici, este mult mai simplu de gestionat cerinţele şi mai greu de făcut o greşeală datorită analizei modulare a întregului sistem.

Tipuri de cerinţe software

Până acum, în povestea cerinţelor software am menţionat cerinţele de utilizator şi de sistem. Cerinţele de sistem se bazează pe cerinţele de utilizator, dar aceasta nu este singura împărţire care este important de ştiut. Toate cerinţele software pot fi împărţite în două grupuri de bază şi astfel există:

- **cerinţe funcţionale** – se referă la serviciile pe care sistemul ar trebui să le ofere utilizatorilor, adică la funcţionalitatea pe care

o va avea sistemul,
- **cerinţe nefuncţionale** – cerinţe legate de calităţile operaţionale ale produsului software, cum ar fi performanţa şi scalabilitatea, securitatea, gradul de utilizare, fiabilitatea, compatibilitatea.

Exemple de cerinţe funcţionale

- *Sistemul trebuie să trimită clientului un e-mail de confirmare după efectuarea achiziţiei.*
- *Numărul de telefon al utilizatorului trebuie să fie implicat în procesul de verificare a conturilor de utilizator.*
- *Sistemul trebuie să aibă o modalitate de a crea un cont prin conectarea la un cont de utilizator existent pe Gmail.*
- *Sistemul va trebui să aibă funcţionalitatea de a reseta parolele în cazul în care utilizatorul a uitat parola.*

Exemple de cerinţe nefuncţionale

- *Sistemul ar trebui să funcţioneze pe dispozitive mobile şi desktop.*
- *Sistemul ar trebui să poată procesa 100 de cereri într-o secundă.*
- *Sistemul nu trebuie să utilizeze mai mult de un gigabyte (GB) de memorie RAM.*
- *În condiţii normale de reţea, site-ul ar trebui să se încarce complet pe dispozitivul utilizatorului în cel mult 3 secunde când numărul de conexiuni simultane este mai mic de 10.000.*

Pe lângă cerinţele funcţionale şi nefuncţionale, există şi aşa-numitele **cerinţe de domeniu**. Cerinţele de domeniu sunt caracteristice domeniului, adică domeniului specific de afaceri pentru

care este destinat produsul software. Software-ul poate fi destinat diverselor domenii de specialitate, precum managementul traficului feroviar, înregistrarea dosarelor medicale, gestionarea echipamentelor militare, funcţionarea ca parte a serviciilor web... Software-ul destinat condiţiilor specifice de afaceri include aproape întotdeauna cerinţe care sunt caracteristice domeniului de afaceri. Astfel de cerinţe sunt altfel numite cerinţe de domeniu. Cerinţele de domeniu sunt deosebit de problematice pentru echipa de dezvoltare, deoarece necesită un angajament special din partea acestora pentru a fi interpretate corect. Un exemplu de cerinţă de domeniu ar putea arăta astfel: *un program care manipulează un dispozitiv medical trebuie să îndeplinească standardul IEC 60601-1, care se referă la cerinţele generale de siguranţă şi performanţă de bază*.

Etape de lucru cu cerinţele software

Până acum am vorbit despre caracteristicile de bază ale cerinţelor software şi despre diferite categorii în care pot fi clasificate. Cu toate acestea, este important de ştiut că lucrul cu cerinţele software implică mai multe etape cronologice cum ar fi:

- **colectarea cerinţelor** - procesul de descoperire a cerinţelor în care participă clienţii, inginerii, utilizatorii, managerii şi toate părţile interesate care participă la colectarea cerinţelor sunt denumite altfel părţi interesate (*stakeholders* în engleză);
- **analiza cerinţelor** – procesul de înţelegere a cerinţelor care au fost colectate în etapa anterioară;
- **specificaţia cerinţelor** – un proces în care cerinţele software sunt înregistrate într-o formă ceva mai formală, rezultând un aşa-numit document SRS (*software requirements specification document*, în engleză);
- **validarea cerinţelor** – procesul de determinare dacă cerinţele definesc un sistem cu caracteristici de care au nevoie utilizatorii.

În continuare se va acorda puţin mai multă atenţie fiecărei etape

tocmai menţionate de lucru cu cerinţele software.

Cum se colectează cerinţele?

Prima etapă în lucrul cu cerinţele software este colectarea acestora. Aceasta este o etapă care este adesea numită etapa de descoperire a cerinţelor software. Am spus deja că cerințele pot fi colectate de la toate părţile interesate, fie că este vorba de utilizatori, clienţi, finanţatori, echipa de dezvoltare şi altele.

După ce ştim de la cine pot fi colectate cerinţele, apare întrebarea logică cu privire la modul în care sunt colectate cerinţele. Există numeroase modalităţi prin care este posibilă colectarea cerinţelor software: prin interviu, observaţie, chestionar, analiza sistemelor conexe existente, prin sarcini temporare, prin revizuirea documentelor.

- **Prin interviu** – cel mai important mod de a obţine cerinţele software este de a efectua interviuri cu părţile interesate. Un interviu este, de fapt, o întâlnire al cărei unic scop este obţinerea informaţiilor, iar în timpul interviului, printr-o conversaţie directă cu părţile interesate, este posibil să ajungeţi la cerinţele software ale viitorului produs software.
- **Prin observaţie** – această metodă implică observarea muncii angajaţilor sau observarea procesului de producţie pentru care trebuie creat un produs software. Această abordare se poate referi şi la observarea lucrului asupra sistemului existent care trebuie îmbunătăţit şi, în orice caz, prin observare se pot trage numeroase concluzii şi astfel se poate forma un anumit număr de cerinţe.
- **Prin chestionar** – chestionarul presupune întrebări pregătite în prealabil care sunt în mare parte distribuite părţilor interesate în formă scrisă; analizând chestionarele completate, putem obţine informaţii pe care le-am ratat în colectarea anterioară şi folosind abordările menţionate până acum.
- **Prin analiza sistemelor conexe existente -** uneori, surse de

informaţii valoroase pot fi sisteme software care există deja şi efectuează lucrări identice sau similare cu sistemul care trebuie creat. Desigur, dacă există astfel de sisteme conexe, ele pot fi o sursă de idei excelente pentru rezolvarea problemelor utilizatorilor sistemului care trebuie creat.

- **Printr-o sarcină temporară** – deseori puteţi auzi că nu există un înlocuitor pentru experienţa dobândită prin munca practică şi tocmai din această cauză, una dintre tehnicile de colectare a cerinţelor poate implica preluarea unor sarcini temporare într-o organizaţie şi, în acest fel, se poate dobândi experienţă de primă mână legată de funcţionarea unui sistem, care în cele din urmă poate facilita formarea cerinţelor.
- **Prin revizuirea documentelor** – o tehnică de colectare a cerinţelor adesea folosită este analiza documentelor de afaceri ale unei companii pentru care este dezvoltat un sistem software. Compania poate deţine şi documentaţia sistemului existent, dacă acesta există; astfel de documente pot facilita înţelegerea noilor cerinţe de utilizator.

Analiza cerinţelor software

Analiza cerinţelor software se referă la procesul de familiarizare a cerinţelor colectate pentru a le înţelege mai bine. În etapa de analiză a cerinţelor software, cel mai important este să se determine eventualele anomalii, inconsecvenţe sau incompletitudini ale cerinţelor colectate.

Există diferite tipuri de analiză a cerinţelor, iar cea mai semnificativă clasificare este în următoarele două grupuri de bază:

- analiza orientată pe proces,
- analiza datelor.

Analiza orientată pe proces presupune prezentarea părţilor unui produs software prin arătarea detaliilor acestora prin procesul de

descompunere. Într-o astfel de analiză, accentul este pus pe ceea ce face software-ul, adică analiza este axată pe funcţionalitatea software-ului, care este reprezentată de procese, precum şi pe relaţia cu entităţile externe.

Analiza datelor se concentrează pe date, mai degrabă decât pe procesele care vor exista într-un sistem software. De altfel, datele sunt parte integrantă a oricărui sistem software şi putem spune că acestea sunt figura sa centrală. Prin urmare, un segment foarte important al analizei cerinţelor software este determinarea tuturor datelor pe care sistemul le va folosi, precum şi relaţiile lor reciproce. Rezultatele obţinute în timpul unei astfel de analize sunt aplicabile implementării modelului relaţional şi obiectual de producţie.

Specificaţia cerinţelor

Specificaţia cerinţelor software (*software requirements specification* în engleză, abreviat SRS) este o descriere a sistemului software care trebuie dezvoltat. De fapt, este vorba de produsul final al colectării şi analizei cerinţelor, care creează aşa-numitul document de cerinţe (*requirements document* în limba engleză). SRS reprezintă baza pentru următoarea etapă de dezvoltare, care se referă în primul rând la procesul de proiectare şi programare.

SRS este creat prin notarea formală a cerinţelor software şi o astfel de notare se poate face în mai multe moduri diferite, unde se poate utiliza:

- **limbajul natural** – presupune notarea cerinţelor în funcţie de teze, folosind una dintre limbile naturale;
- **limbajul natural structurat** – din nou cerinţele sunt notate folosind limbajul natural, dar de data aceasta folosind o formă standard, adică un şablon pregătit în avans;
- **notaţiile grafice** – diferite tipuri de modele grafice, care descriu proprietăţile funcţionale ale sistemului, pot fi diferite tipuri de diagrame, pe care le vom aminti puţin mai târziu.

Validarea cerinţelor software

După crearea unui document care reprezintă specificaţia cerinţelor, se trece la procesul de validare a acestora. Cerinţele sunt verificate încă o dată, pentru a se asigura că îndeplinesc toate calităţile bune pe care ar trebui să le aibă o cerinţă de software, despre care am discutat mai devreme.

Cea mai importantă caracteristică a fiecărei cerinţe este că răspunde nevoilor părţilor interesate care vor folosi programul. Prin urmare, procesul de verificare a cerinţelor determină dacă există vreo problemă cu cerinţele care au fost colectate şi procesate în etapele anterioare.

Procesul de validare a cerinţelor software este o etapă foarte importantă a gestionării cerinţelor, deoarece permite detectarea timpurie a erorilor. Detectarea timpurie a erorilor contribuie în mare măsură la reducerea costului total al dezvoltării unui produs software

Utilizarea diagramelor în procesul de lucru cu cerinţele

În procesul de analiză a cerinţelor software, este posibilă utilizarea diferitor tipuri de diagrame, care ajută la vizualizarea caracteristicilor sistemului şi astfel le fac mai clare şi mai uşor de înţeles. Pentru început, puţin mai devreme am vorbit despre analiza orientată pe proces, în timpul căreia sunt determinate proprietăţile funcţionale ale sistemului. Pentru a uşura acest tip de analiză, se folosesc adesea două tipuri de diagrame:

- diagrama de context,
- diagrama fluxului de date.

Diagrama de context este utilizată pentru a reprezenta limitele sistemului modelat şi relaţia acestuia cu mediul, adică cu părţile interesate. Părţile sale sunt, pe lângă sistemul în sine, entităţi externe şi fluxuri de date. Un exemplu de diagramă de context ar putea arăta

ca în imaginea 8.2.

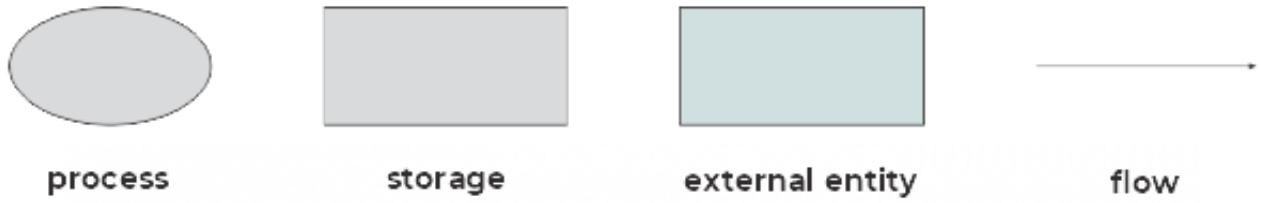
Imaginea 8.2. Diagrama de context

În imaginea 8.2. puteţi vedea diagrama de context a sistemului software de cumpărături online. Figura centrală a diagramei este sistemul propriu-zis. Totuşi, pe lângă acesta puteţi să vedeţi entităţi externe, adică părţi interesate care vor folosi un astfel de sistem. Este vorba despre clienţi şi administratori. Liniile care leagă astfel de părţi interesate de sistem, ilustrează interacţiunea lor. Astfel, clientul îşi poate crea un cont, plasa o comandă, vizualiza produse, în timp ce administratorul poate adăuga produse noi, confirma comenzi, vizualiza produse etc.

Diagrama fluxului de date este utilizată pentru a reprezenta modul în care datele se deplasează prin sistem. Cele trei elemente de bază ale unor astfel de diagrame sunt:

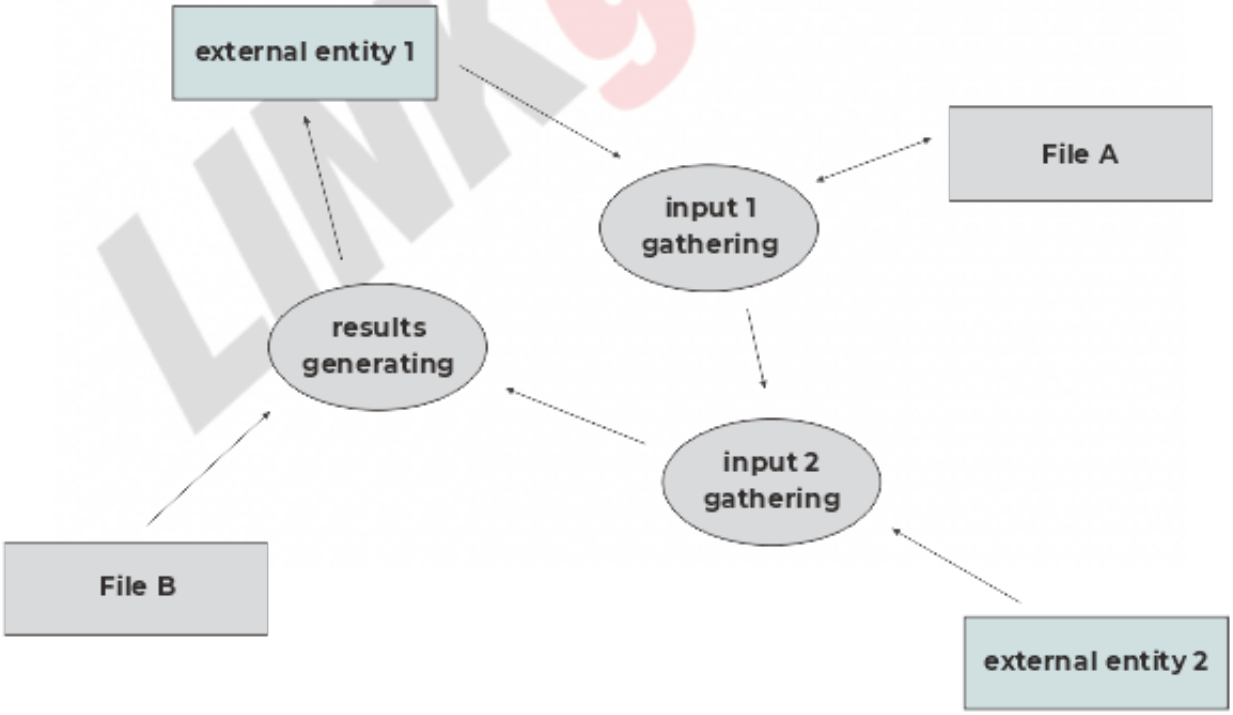
- fluxurile de date,
- spaţiile de stocare,
- procesele,
- entităţile externe.

Fiecare dintre aceste elemente este reprezentat printr-o notaţie specifică (imaginea 8.3.).



Imaginea 8.3. Simboluri cu ajutorul cărora se creează diagramele fluxului de date

Exemplul unei diagrame a fluxului de date poate arăta ca în imaginea 8.4.



Imaginea 8.4. Diagrama fluxului de date

În imaginea 8.4. puteţi vedea o diagramă simplă a fluxului de date a funcţionalităţii unei aplicaţii. Aşa ajungem la prima caracteristică a diagramei fluxului de date. Astfel de diagrame sunt foarte des folosite pentru a realiza **descompunerea**. Descompunerea înseamnă afişarea detaliilor, spre deosebire de afişarea funcţionării întregului sistem. La urma urmei, putem vedea că în exemplul din imaginea 8.4., acesta ilustrează fluxul de date dintr-un singur segment al operaţiunii programului.

În imaginea 8.4. puteţi vedea diagrama fluxului de date a funcţionalităţii pentru preluarea a două intrări şi generarea rezultatelor. La o astfel de operaţiune participă două entităţi externe şi două spaţii de stocare a datelor (File A şi File B). Liniile care leagă elementele diagramei ilustrează mişcarea datelor.

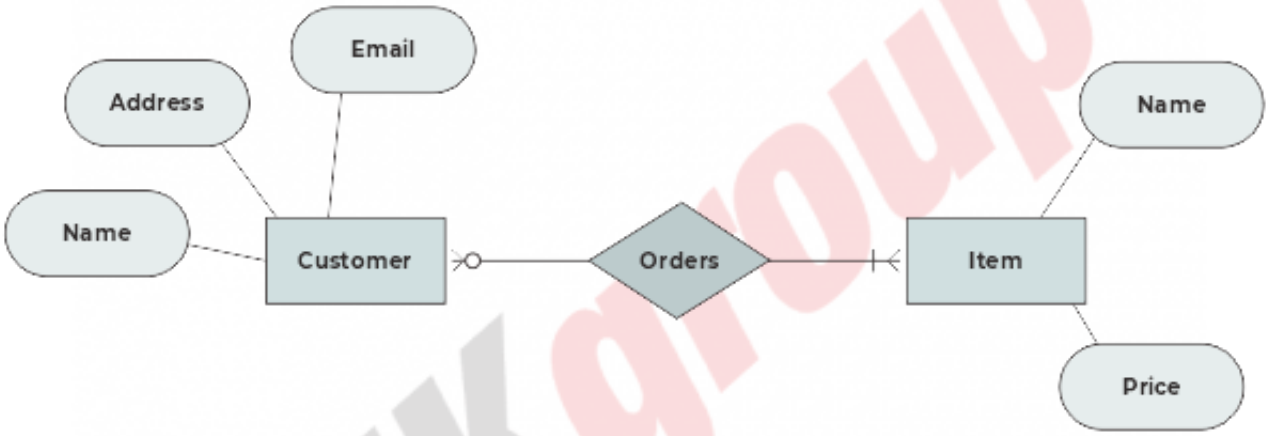
Puţin mai devreme în această lecţie am vorbit despre analiza cerinţelor software, cu un accent deosebit pe date. La efectuarea unei astfel de analize se recurge adesea la crearea aşa-numitelor diagrame de entităţi şi relaţii (*entity/relationship diagrams* în engleză).

Diagramele entitate-relaţie reprezintă entităţi şi atributele acestora, precum şi relaţiile dintre astfel de entităţi. Ele ajută la modelarea datelor pe care le va folosi sistemul, deci reprezintă clar entităţile şi proprietăţile (atributele) acestora, precum şi relaţiile care există între entităţi. Acest tip de diagramă este format prin combinarea simbolurilor grafice şi a textului (imaginea 8.5.).

Imaginea 8.5. Simboluri pentru crearea diagramei entitate-relaţie

În imaginea 8.5. puteţi vedea simbolurile de bază folosite pentru a forma diagrame de entități şi relaţii. Entităţile, atributele, entităţile asociative şi relaţiile sunt prezentate sub diferite forme.

Un exemplu de diagramă entitate-relaţie ar putea arăta ca în imaginea 8.6.



Imaginea 8.6. Diagrama entitate-relaţie (entity/relationship diagram)

Diagrama entitate-relaţie din imaginea 8.6. prezintă cele două entităţi care vor exista în sistemul modelat – consumator (*Customer*) şi articol/element (*Item*). Fiecare dintre aceste entităţi are propriile sale caracteristici. Consumatorul are numele, adresa şi adresa de e-mail, în timp ce articolul are un nume şi un preţ. Relaţia dintre aceste două entităţi construieşte o altă entitate asociativă suplimentară – comenzile (*Orders*). Ele definesc aşa-numita relaţie mai mult la mai mult, care descrie relaţia dintre consumator şi articol. Şi anume, fiecare consumator poate achiziţiona mai multe articole diferite, iar fiecare articol poate fi achiziţionat de mai mulţi consumatori diferiţi.

UML

Desenarea diferitor diagrame nu se limitează la procesul de analiză a cerinţelor software. Se poate spune că modelarea este o operaţiune universală care urmăreşte diverse aspecte ale proiectelor în care se dezvoltă software-ul de calculator.

Un model este o reprezentare simplificată a realităţii, care permite o descriere completă a sistemului dintr-o anumită perspectivă. Modelele sunt realizate pentru a înţelege mai bine sistemul. Sistemele complexe sunt modelate deoarece nu putem înţelege aceste sisteme ca un întreg. Astfel, modelele ne ajută să vizualizăm sistemul aşa cum este sau aşa cum ne dorim să fie, ne permit să definim structura sau comportamentul sistemului, să furnizăm modele în construcţia sistemului şi să documentăm deciziile pe care le-am luat.

Unified Modelling Language (UML) este un limbaj de modelare vizuală utilizat pentru a specifica, vizualiza, construi şi documenta sisteme software. Este folosit pentru a analiza, proiecta, căuta, configura, întreţine şi controla informaţii despre astfel de sisteme.

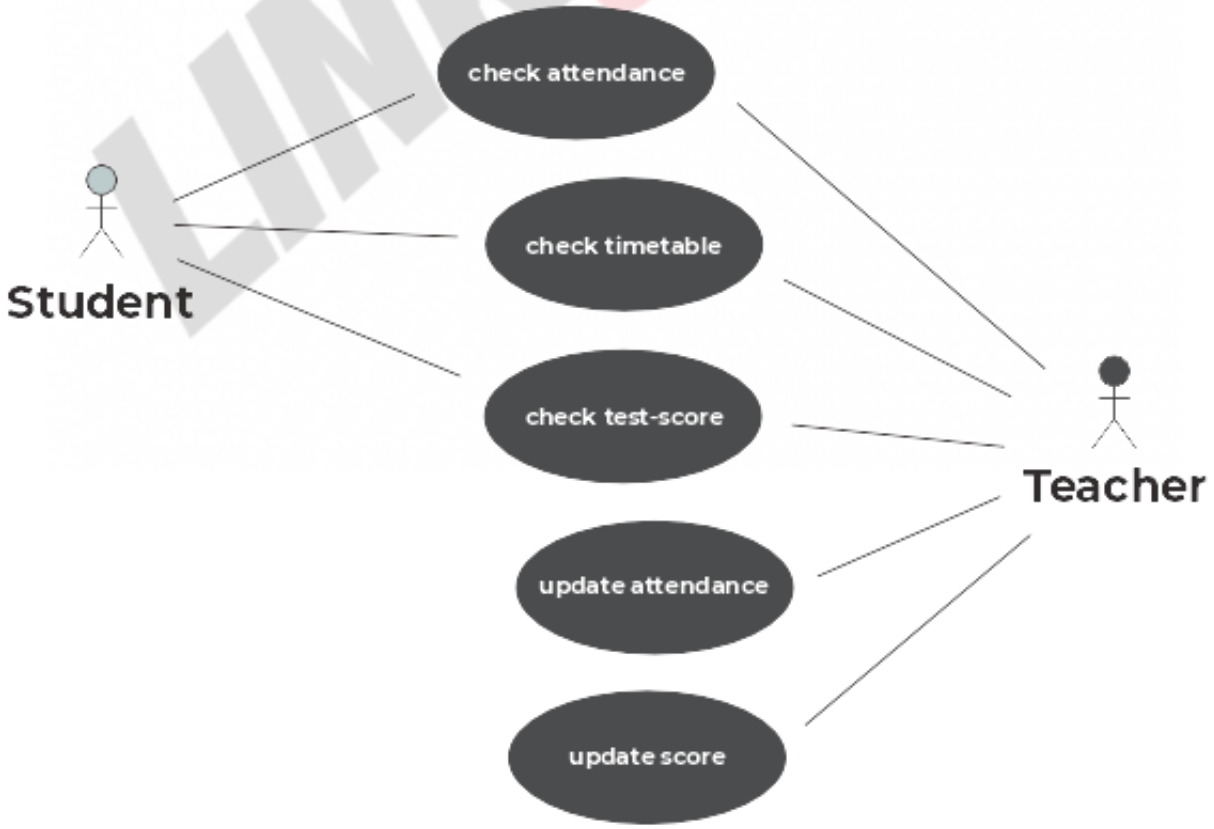
UML este un limbaj de modelare foarte puternic, care este folosit la multe niveluri diferite şi în multe stări ale ciclului de dezvoltare. Are un număr mare de diagrame diferite care pot fi împărţite aproximativ în două categorii:

- **diagrame structurale** - folosite pentru a reprezenta structura sistemului, iar acest tip de diagramă include:
    - diagramele componente (Component Diagrams),
    - diagramele obiectelor (Object Diagrams),
    - diagramele de clase (Class Diagrams),
    - diagramele de implementare (Deployment Diagrams) şi
- **diagrame de comportament** - reprezintă aspectele dinamice ale sistemului care se modelează, iar în acest grup de diagrame sunt incluse următoarele diagrame:
    - diagramele de cazuri de utilizare (Use Case Diagrams),
    - diagramele de stare (State Diagrams),

- diagramele de activitate (Activity Diagrams),
- diagramele de interacţiune (Interaction Diagrams).

În această lecţie nu încercăm în niciun caz să ne familiarizăm în detaliu cu limbajul UML, având în vedere că nici acest curs complet nu ar fi suficient pentru aşa ceva. Scopul nostru în acest moment este o înţelegere de bază a acestui termen şi a rolului său în lumea dezvoltării software. Prin urmare, în continuare vom prezenta câteva diagrame UML care sunt cele mai accesibile pentru începători în lumea dezvoltării software.

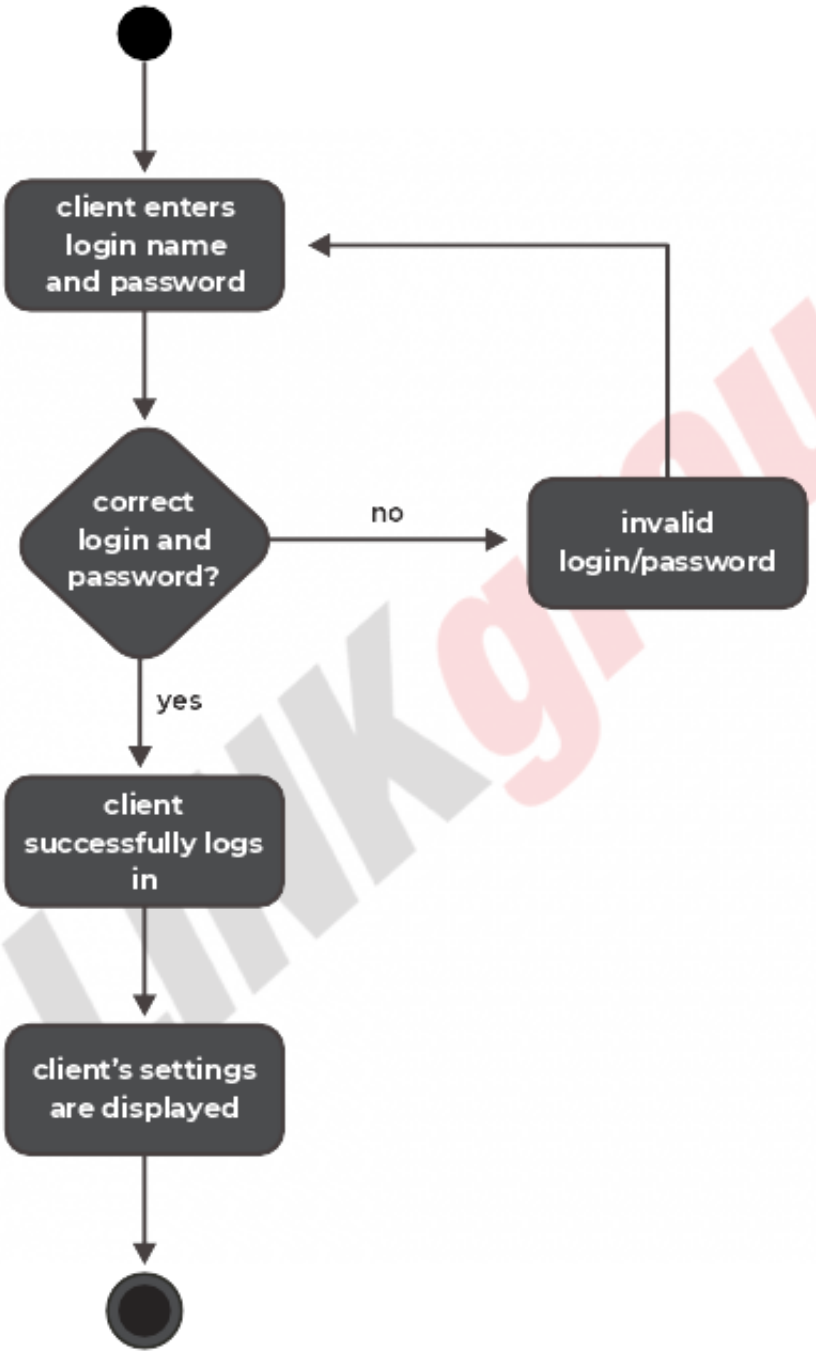
Diagrama cazurilor de utilizare (*Use Case* în engleză) ilustrează interacţiunea utilizatorului cu sistemul. Aceasta este, cu siguranţă, una dintre cele mai cunoscute diagrame UML. În acelaşi timp, este mai accesibilă pentru începători şi o astfel de diagramă se poate vedea în imaginea 8.7.



Imaginea 8.7. Exemplu de diagramă a cazurilor de utilizare (Use case)

Imaginea 8.7. ilustrează o diagramă a cazurilor de utilizare. Aceasta are doi actori şi un număr diferit de cazuri de utilizare care ilustrează modalităţile în care poate fi utilizat sistemul (de exemplu, un student poate verifica prezenţa, un profesor poate introduce o notă...).

Diagrama de activitate este un alt tip de diagramă UML despre care se poate spune că este accesibilă începătorilor în lumea programării. Acesta ilustrează fluxul de activităţi atunci când se efectuează o operaţiune a sistemului (imaginea 8.8.).



Imaginea 8.8. Exemplul diagramei de activitate pentru autentificarea la sistem

Imaginea 8.8. ilustrează diagrama de activitate pentru procesul de autentificare la un sistem software. Blocul principal al acestui tip de diagramă îl constituite dreptunghiurile cu margini rotunjite care reprezintă activităţile. Simbolul romb indică deciziile. Cercul negru ilustrează începutul diagramei, iar cercul înrămat cu negru sfârşitul acesteia.

Diagrama din imaginea 8.8. ilustrează activităţile efectuate atunci când utilizatorii existenţi se autentifică la un sistem software. Utilizatorul introduce mai întâi un nume de utilizator şi o parolă. Apoi se ia o decizie care verifică validitatea datelor introduse. Dacă datele sunt corecte, utilizatorul se autentifică şi sunt afişate setările utilizatorului. În caz contrar, autentificarea utilizatorului eşuează, iar utilizatorul revine la reintroducerea datelor de acces.

