**Rendszertervezés és a szoftverfejlesztés**

UML: felhasználói, szekvencia és egyedkapcsolati diagram. Verziókezelés: Git alapjai (branch, pull,push, commit, tag, merge fogalma, gyakorlati hasznuk). Agilis módszertanok (pl. SCRUM). Prototípus-modellek. Szoftver dokumentáció készítése.  
Feladat: Saját, egyedi példa a fent említett UML diagramokra, azok bemutatása, magyarázata

**I. UML: (még az activity diagramról is lehet beszélni)**

* Jelentése: Unified Modeling Language
* Fontos elemei a vizualizáció, áttekinthetőség, egységesség, könnyű értelmezhetőség
* Integrált diagramkészletből álló szabványos modellezési nyelv
* Segít a szoftverrendszerek részeinek meghatározásában, megjelenítésében, felépítésében és dokumentálásában, illetve az üzleti modellezésben és nem szoftveres tervezésben is

a, Felhasználói diagram - ez valószínűleg use case

* Célja, hogy bemutassa, milyen különböző módokon léphet kapcsolatba a felhasználó (actor) a rendszerrel és milyen „tulajdonságaik vannak”
* Bemutatja, hogy az alkalmazás hogyan lép kölcsönhatásba emberekkel, szervezetekkel vagy külső rendszerekkel, illetve a modellezett rendszer hatókörét
* Generalizáció: A leszármaztatott actor rendelkezik az „ősactor” minden tulajdonságával is a sajátjai mellett.
* Include
  + kötelező és az alaphasználati eset része
  + szaggatott nyíl jelzi az included használati eset irányába
* Extend
  + nem kötelező, és az alaphasználati eset után jöhet
  + szaggatott nyíl jelzi az alaphasználati eset irányába

A képen szöveg, képernyőkép, Betűtípus, diagram látható

Automatikusan generált leírás

b, Szekvencia diagram

* Idő központú és az interakciók sorrendjét mutatja a diagram függőleges tengelyének (időtengely) használatával.
* Vagyis leírja, hogyan és milyen sorrendben működik együtt objektumok egy csoportja, milyen üzeneteket küldenek egymásnak és mikor
* Egy objektum élettartamát a függőleges vonalának hossza jelzi
* Ezek az objektumok üzenet fogadásakor aktív állapotba kerülnek és némely objektumok élettartam csak ilyenkor kezdődik.
* Aszinkron folyamatnál a küldő nem várja meg a választ.
* Lehet feltételes, elágazás, ciklus vagy párhuzamosítás.

A képen szöveg, képernyőkép, diagram, Tervrajz látható

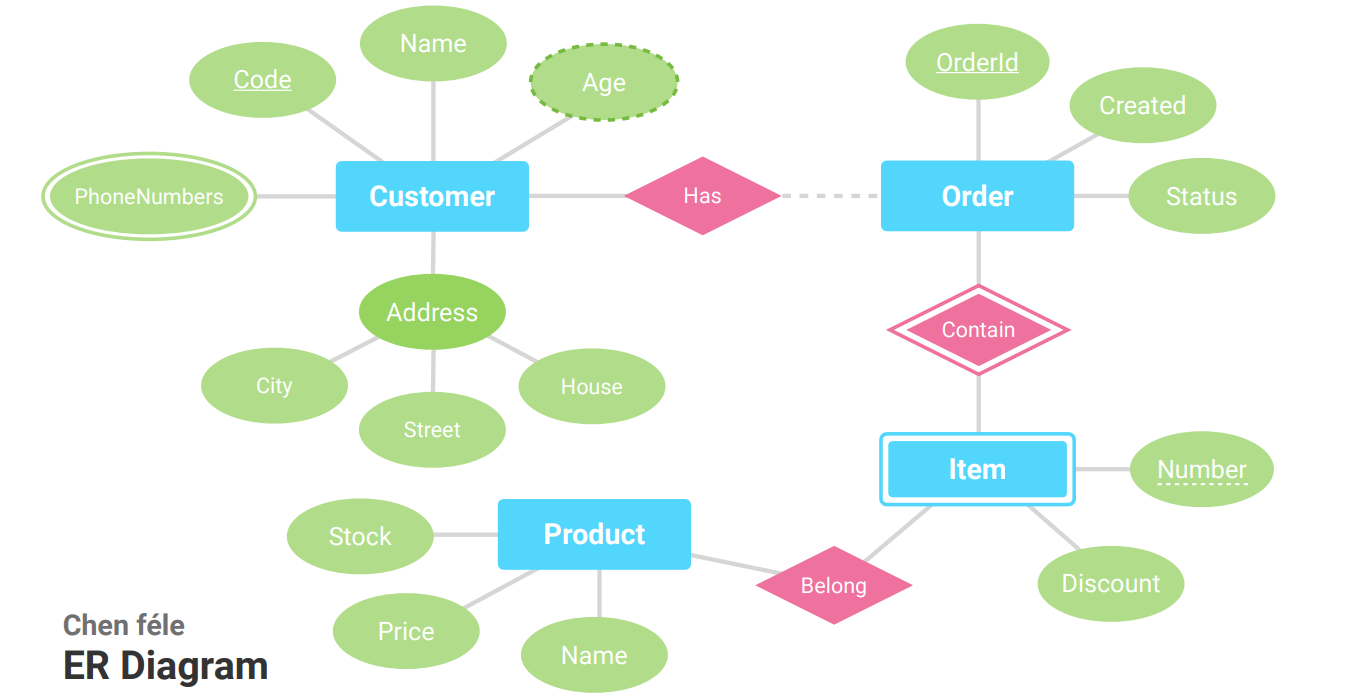
Automatikusan generált leírás

c, Egyedkapcsolati diagram

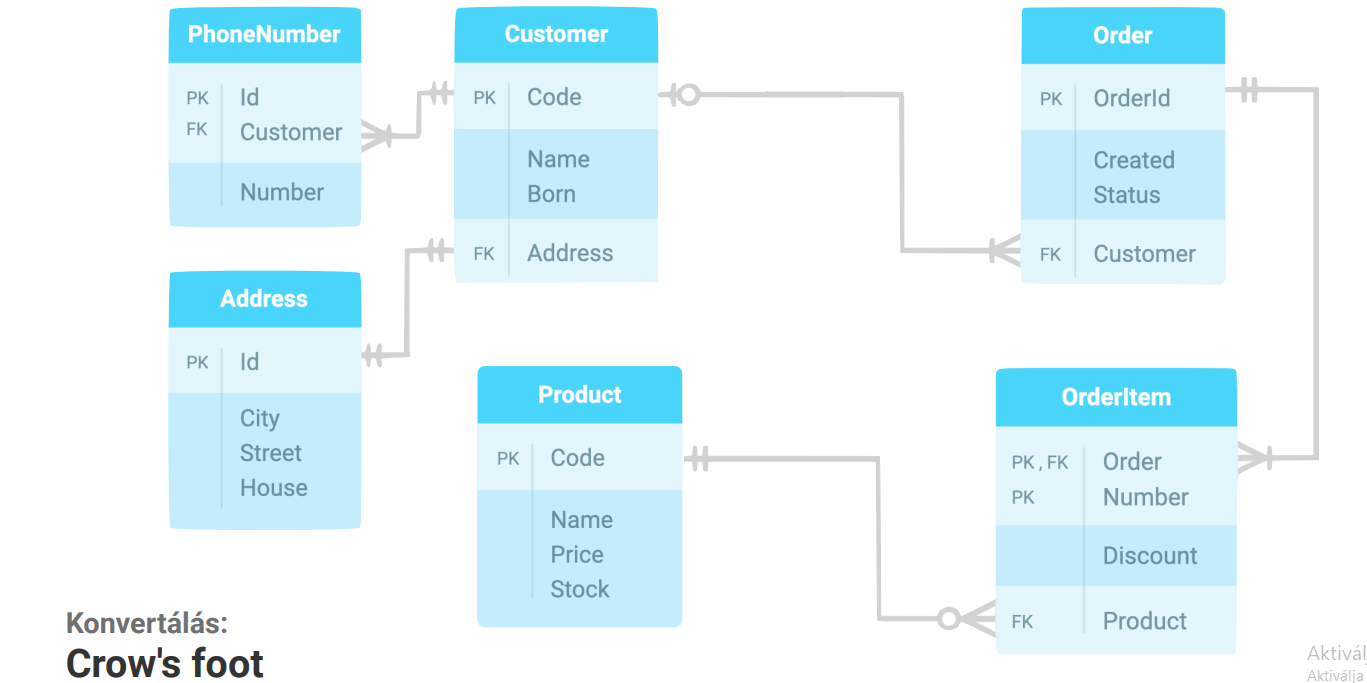
Az(ER) diagram egyfajta folyamatábra, amely bemutatja, hogy az „entitások”, például emberek, objektumok vagy fogalmak hogyan viszonyulnak egymáshoz egy rendszeren belül.

* Egyedtípus: meghatározott szerkezetű dolog, információt tárolunk róla, több példány is jelen lesz
* Attribute
  + jellemző, amit tárolunk az egyedeknél
  + egy egyednek több tulajdonsága is lehet
* Kapcsolat
  + egyedtípusok összekapcsolása valami alapján
  + több egyedet is össze lehet kapcsolni, akár attribútumok is tartozhatnak hozzá, lehet 1-1, 1-M, M-1, M-M
* Elsődleges kulcs
  + az egyedek azonosítására szolgál
  + általában valami szám vagy rövid szöveg
* Többértékű
  + listaként működő tulajdonság
  + ugyanolyan jellegű és szerkezetű adatokat tárolunk benne
* Származtatott
  + valószínűleg nem fogjuk tárolni, más értékből számítható
* Gyenge egyed
  + önmagában nem alkalmas egyed azonosítására
  + erős egyeddel áll kapcsolatban
* Parciális kulcs
  + gyenge egyed azonosítására szolgál

A **Chen ER diagram** a következőképpen ábrázolja az entitásokat, illetve a kapcsolatok számosságát egyszerűen 1 vagy M jelzővel mutatja be.



A **Crow’s foot ER diagramban** az egyedtípusokat táblákként említjük, a speciális mezőket külön megjelöljük és egyéb információkat is megjelenítünk., illetve a kapcsolatok számosságas is más, megjelenik a kötelező vagy opcionális.



**II. Verziókezelés: git alapjai**

A Git egy ingyenes és nyílt forráskódú elosztott verziókezelő rendszer, amely a kicsitől a nagyon nagy projektig mindent gyorsan és hatékonyan kezel. (elég csak felszínesen beszélni a többi fogalomról, mint pl. fork, cherrypick stb)

* Branch
  + main repository új/különálló ága
    - régen master volt main helyett, csak megváltoztatták
  + lehetővé teszi, hogy a projekt különböző részein dolgozz anélkül, hogy a fő ágat érintenéd, ezeket később mergelheted
* Pull
  + frissíti a jelenlegi checked out local branchet a remote alapján, vagyis fetcheli az új commitokat majd mergeli a local branchedbe
* Push
  + lehetővé teszi a commitok elküldését a local repository local branchéből a remote repositoryba
* Commit
  + nyomon követhetjük a munkánkat
  + tulajdonképpen a commit egy mentési pontként működik
  + vagyis vissza is lehet térni ezekhez, ha valamilyen változtatást szeretnénk eszközölni
  + mindig kell csatolni egy üzenetet mellé
* Tag
  + egy adott verziót „elmentünk” majd a branchekkel ellentétben ehhez már nem lehet commitolni, hanem mindig így marad
  + szoftverkiadási ciklusoknak megfelelő verziószám azonosítók létrehozására használatos
* Merge
  + két branch egy branchbe való összefésülése
  + különválástól kezdve az összes commitot egybeolvassa/fésüli
  + lehetnek conflictok vagyis eltérések, amelyeket általában a fejlesztők kezelnek és egy végleges változatot adnak.

**III. Agilis módszertanok (érdemes megemlíteni a vízesést is)**

* Scrum: hangsúly a kísérletezésen és értékelésen van, célja az eredmények nyomon követése és a problémák korai felismerése, a szükséges javíthatóságok felismerése
* A Scrum csapat általában három szerepkörből áll: a scrum masterből, a product ownerből és a fejlesztői csapatból (bár ez a felállás kicsit eltérhet)
* Scrum master: felelős azért, hogy a csapat kövesse a Scrum módszertant
* Product owner: felelős a projekt eredményéért, arra törekszik, hogy maximalizálja a termék értékét a backlog kezelésével és optimalizálásával
* Folyamatos kapcsolat a megrendelővel
* Menedzsment szervezi és felügyeli a folyamatokat majd kiosztásra kerülnek a szerepek és a feladatok
* Különböző szerepek pl: PM, DEV, QA, BA stb.
* Sprint: 1-4 hetes időkeret, ami alatt egy feladat remélhetőleg végrehajtódik. Az előző Sprint befejezése után azonnal indul egy új Sprint
* Fontos, hogy legyen valami bemutatható eredmény
* Változásokra gyorsan tud reagálni mivel gyakran jelent a megrendelőnek demo során
* Sprintek során több meeting is lehet pl. standup
* A sprintek elején mindig tervezés a követelmények alapján
* Ezután beütemezik a sprinteket és a feladatokat
* Utána végrehajtás, fejlesztés
* Tesztelés és az eredményeik összegyűjtése
* A sprint végén retro vagyis kiértékelés

A képen szöveg, képernyőkép, kör, embléma látható

Automatikusan generált leírás

* A Kanban egy vizuális menedzsment módszer a munkafolyamatok menedzselésére és optimalizálására
* Valós idejű, vizuális megjelenítést biztosít a munkaelemekről, amint azok a folyamat különböző szakaszain haladnak keresztül
* Kanban tábla: A Kanban tábla a munkafolyamat vizuális ábrázolása
  + Oszlopokból áll, amelyek különböző szakaszokat reprezentálnak, és minden oszlop az egyes munkaelemeket jelző kártyákat tartalmaz, amelyek információt tartalmaznak
  + Gyakori oszlopok: a „Teendő”, „Folyamatban” és a „Kész”.
* Limit WIP: Korlátozza az adott időben folyamatban lévő munkaelemek számát túlterhelés elkerülése céljából
* Kanban a folyamatos fejlesztést ösztönzi a folyamat rendszeres felülvizsgálatával és adaptálásával
* Vizuális jelek, például színkódolt kártyák, további információk közvetítésére használhatók a munkaelemekről, például a prioritásról vagy a típusról

**IV. Prototípus modellek**

**A képen szöveg, képernyőkép, Betűtípus, embléma látható

Automatikusan generált leírás**

**V. Szoftver dokumentáció készítése - erről nem volt tananyag, úgyhogy internet**

* A program egy termék, és egy termékhez mindig tartozniuk kell különböző leírásoknak.
* Több fajtája is van
* Fejlesztői dokumentáció
  + minden más szempontnál fontosabb a pontosság
  + specifikációk, követelményanalízis. A feladat és a megoldástól elvárt követelmények meghatározása.
  + fejlesztői környezet leírása: választott programnyelv(ek), és verziószáma(i) stb.
  + algoritmusok és az adatok (típusok, osztályok, programkonstansok) leírása, ezek kapcsolata
  + kód, implementációs szabványok
  + tesztesetek
  + a program hiányosságait szemlélteti
  + hatékonysági mérések
  + fejlesztési lehetőségek
  + fejlesztő adatai
  + koncepcióterv: feladata a probléma lehetséges megoldásainak bemutatása és értékelése. Felhasználásával a feladat kitűzője választhat megoldási lehetőségek között.
  + rendszerterv: Tartalmaznia kell a szükséges hardver-szoftver környezet leírását, amely meghatározza a rendszer használhatósági körét. Itt konkrét döntéseket kell megfogalmazni, következményeikkel, indoklásukkal együtt. Ez az a dokumentum, amelynek alapján a tényleges programozási munka elkezdődhet.
    - itt lehet megemlíteni a wireframeket is esetleg
* Felhasználói dokumentáció
  + a felhasználó, az üzembe helyező, a betanító használja
  + időnként lehet „szájbarágó”
  + a feladat – rövid áttekintő és részletes a használathoz
  + a futási környezet leírása
  + a használat leírása
  + bemenő adatok, eredmények, szolgáltatások részletes leírása: mit, mikor, milyen sorrendben kell megadni
  + mintaalkalmazás – példafutás, ezek alapján tudják elképzelni a programot
  + hibaüzenetek és a hibák lehetséges okai, mi a teendő
* Programismertető
  + célja a vásárló, programkereső ember meggyőzése arról, hogy e program felel meg leginkább igényeinek
  + dicsérni kell a programot, kiemelve jó tulajdonságait
  + a feladat rövid, szöveges leírása, áttekintési céllal
  + a program tulajdonságainak rövid leírása
  + minimális hardver és szoftver környezet
* Installálási kézikönyv, operátori kézikönyv
  + szerepel mindaz az információ, aminek segítségével a program elhelyezhető a gépünkön úgy, hogy az aktuális környezetben optimálisan működjön
* Szerkezeti szabályok
  + ne legyen túl hosszú se rövid
  + legyen világosan tagolt és segítse a gyors keresést
  + legyen tömör, ne vesszen el az olvasó a részletekben
  + legyen olvasható, ne túl formális
  + legyen pontos, minden kérdésre legyen válasz
  + legyen tartalomjegyzék is

**Források:**

Rendszertervezés teams csoport

<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/>

<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>

<https://www.geeksforgeeks.org/use-case-diagram/>

<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>

<https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/>

<https://www.gleek.io/blog/crows-foot-chen>

<https://www.simplilearn.com/tutorials/git-tutorial/what-is-git>

<https://www.freecodecamp.org/news/10-important-git-commands-that-every-developer-should-know/>

<https://www.scrum.org/resources/what-scrum-module>

<https://www.geeksforgeeks.org/software-engineering-agile-software-development/>

<https://asana.com/resources/what-is-kanban>

<https://progalap.elte.hu/downloads/seged/eTananyag/lecke28_lap1.html>