MILTON FRIEDMAN EGYETEM

Rendszertervezési módszertanok

Tematika

A rendszertervezés alapjai

Információs rendszerek tervezése és a tervezés ciklusai

A rendszertervezés

Komplex folyamat.

Célja: egy adott cél elérése érdekében működő információs rendszer tervezése, létrehozása, telepítése és a működési feltételeinek a biztosítása.

Rendszertervezés és szervezet

Az információs rendszerek általában valamilyen vállalat működését segítik elő.

Az információs rendszerek tervezése szinte minden esetben szervezeti aspektusból történik, alkalmazkodva a szervezet méretéhez és információs igényeihez.

És mi van a folyamatokkal? Hiszen azokat kell követni!

Rendszertervezés és szervezet

Az optimálisan működő információs rendszer tervezéséhez elengedhetetlen, hogy megértsük a szervezet felépítését és működését és az információs folyamatok lefutását a szervezeten belül.

Életciklus modellek

Az informatikai rendszerek ún. életciklus modellek mentén jönnek létre.

Az életciklus magában foglalja mindazon tevékenységet, amely az informatikai rendszer követelményeinek meghatározásától, a rendszer létrejöttén, üzembe helyezésén és tartásán keresztül, egészen a rendszer használatának megszűnéséig tart.

A modellek strukturálják a tervezés és fejlesztés tevékenységét, útmutatást adnak a csoportmunka irányításához, meghatározzák a fejlesztésben résztvevők feladatait.

Életciklus modellek eltérései

- A megvalósítás lépései nem azonosak,
- eltérő sorrendben, esetleg párhuzamosan hajtódnak végre,
- a különböző modellek más és más lépésekre helyezik a hangsúlyt.

Életciklus modellek hasonlóságai

Az informatikai rendszerek létrehozását több lépésre bontják.

A lépések, ha nem is épülnek közvetlen egymásra, mert például a fejlesztés során nem időben egymás után, hanem párhuzamosan végzik el őket, azonban minden egyes lépésnek összhangban kell lennie az összes többivel.

Amennyiben egy lépés nem tesz eleget a követelményeknek, úgy az nem hajtható végre, hanem módosítani kell, majd újra megvizsgálni.

Verifikáció

Azt vizsgálja, hogy az informatikai rendszer egyes elemei összhangban vannak-e a többi elemmel, együtt tudnak-e működni és, hogy a specifikációban megfogalmazottaknak eleget tesz-e.

A verifikáció arra a kérdésre felel, hogy **„jól működő terméket fejlesztünk-e?”**.

Validáció

Az informatikai rendszer egészét vizsgálja, abból a szempontból, hogy rendszer megvalósítja-e a projekt kezdetekor meghatározott célokat, eleget tesz-e a rendszer a megrendelő igényeinek.

A validáció arra a kérdésre válaszol, hogy **„megfelelő terméket fejlesztettünk-e?”**.

Információs rendszertervezési modellek

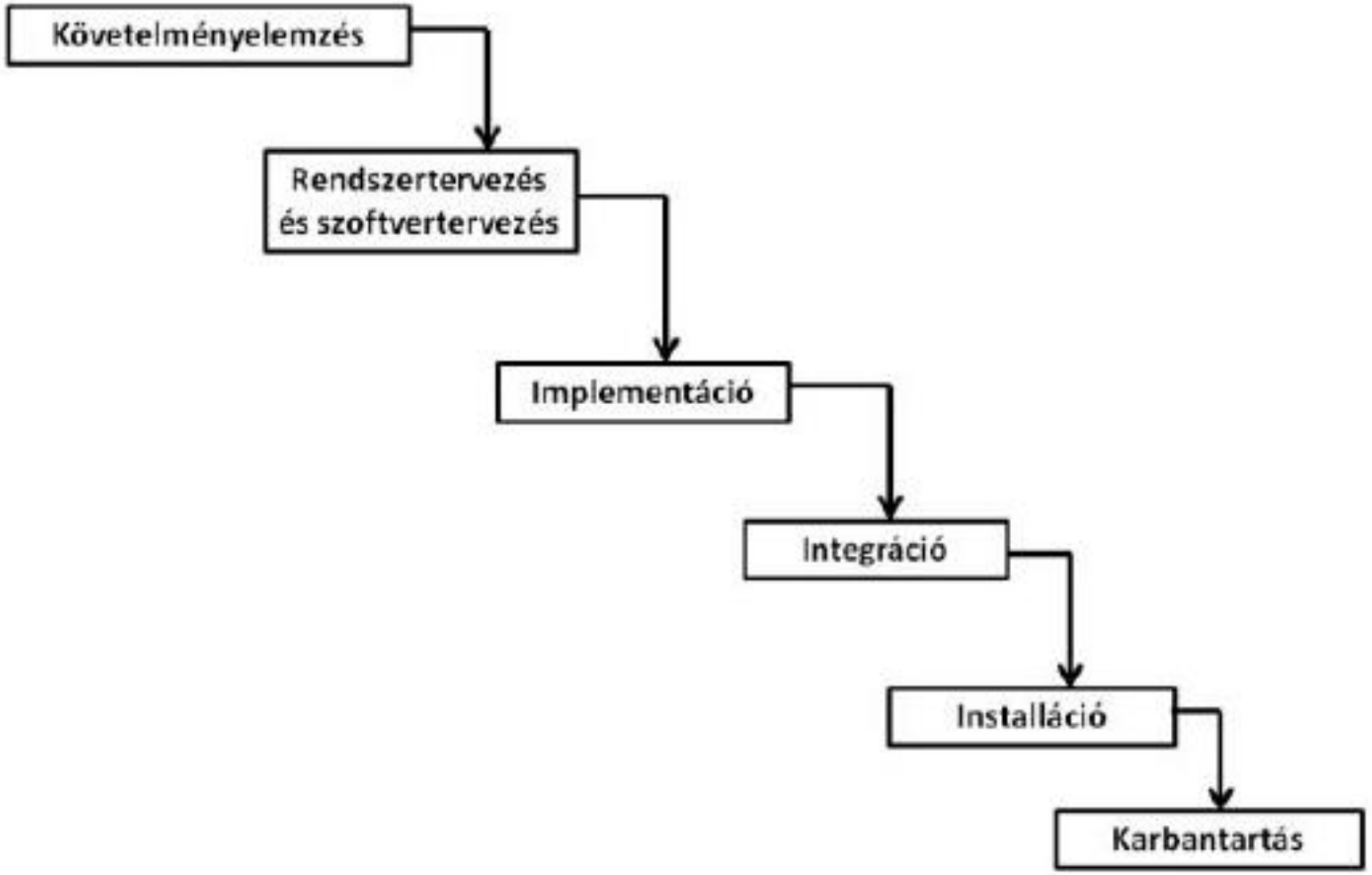
- Vízesés modell
- V modell
- Evolúciós modell
- Komponensalapú és újrafelhasználható rendszerfejlesztés
- Spirális fejlesztés
- Iteratív- inkrementális modell
- Gyors Alkalmazásfejlesztés (RAD)
- RUP (Rational Unified Process)
- eXtreme Programming
- Agilis szoftverfejlesztés

A vízesés modell

A vízesés modell az első publikált rendszerfejlesztési modell, a hetvenes években Winston W. Royce publikálta.

A vízesésmodell egy szekvenciális fejlesztési modell, amely jól elkülönülő lépésekre osztja a rendszertervezés egymás utáni lépcsőit.

A vízesés modell lépcsői



A vízesés modell lépcsői

Követelményelemzés: Milyen követelményeknek kell megfelelnie a rendszernek, milyen szolgáltatásokat kell nyújtania a felhasználó felé.

Rendszertervezés és szoftvertervezés: Ebben a szakaszban szétválasztjuk a hardver- és a szoftverkomponensek funkcióit.

Implementáció: A szoftverkomponensek létrehozása, és a komponensek funkcionális ellenőrzése.

Integráció: a különálló komponensek integrálása egységes rendszerré és a működés globális tesztelése.

Installáció: a rendszer üzembe helyezése.

Karbantartás: a rendszer működési feltételeinek biztosítása.

A vízesésmodell lényege

A következő lépcsőfokra nem léphetünk, amíg az előző fázis be nem fejeződött.

A gyakorlatban természetesen van lehetőség az előző lépcsőfokra visszatérni (ha funkcionálisan nem kielégítő az eredmény), illetve a lépcsőfokok részben átfedhetik egymást.

A lépcsőfokok végét többek között a dokumentáció elkészülte is jelzi. Ha vissza kell térni egy korábbi lépcsőfokhoz, az gyakran az adott lépcsőfok (és ebből adódóan az összes többi) tervezésének újrakezdését jelentheti.

A vízesés modell jellemzői

ELŐNYE

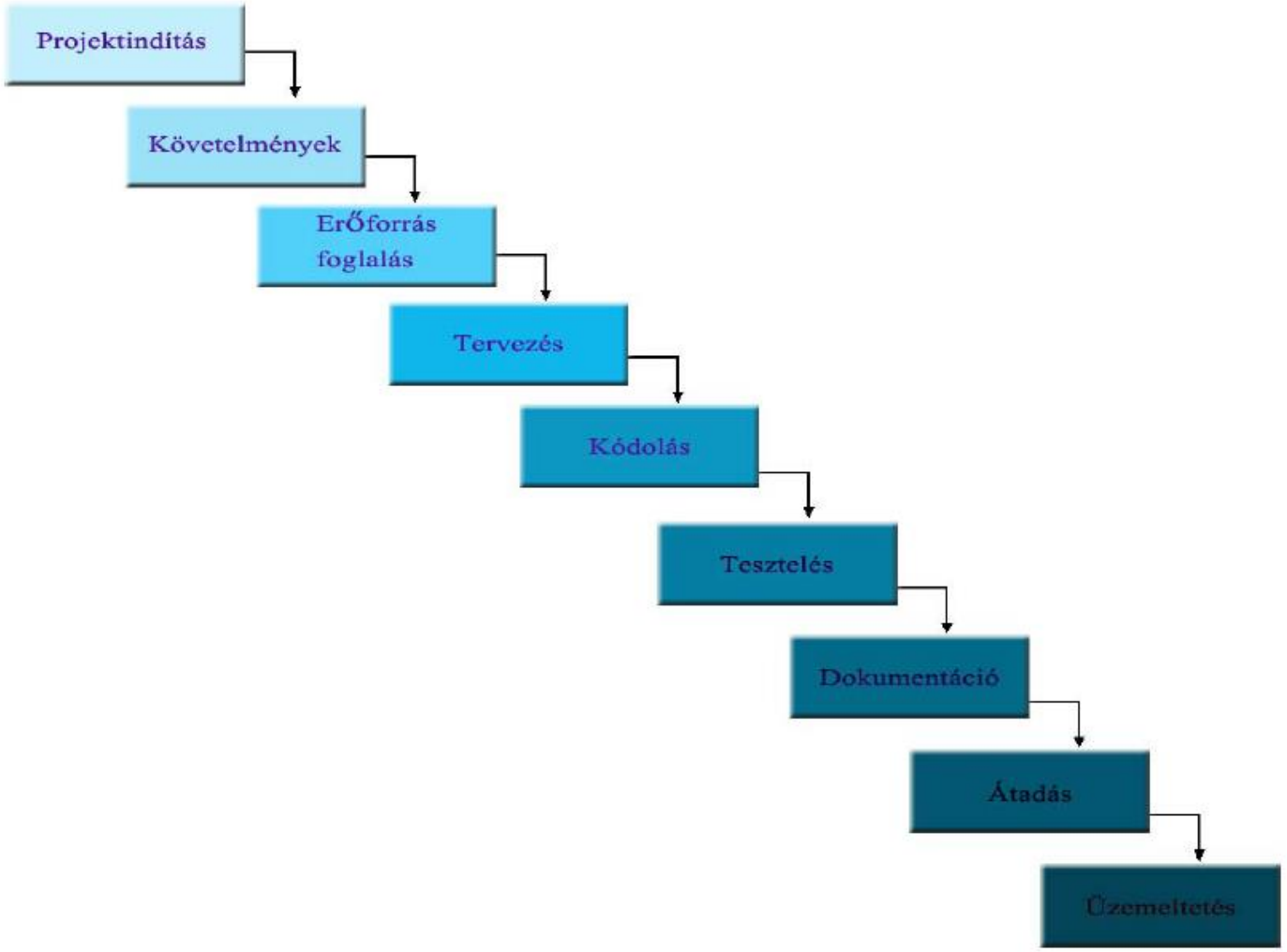
- a részletes dokumentáció következtében a rendszer átgondolt,
- redundáns elemektől mentes,
- könnyen karbantartható lesz.

HÁTRÁNYA

- a követelményeket a rendszertervezés elején pontosan meg kell határozni,
- bármilyen változás a tervezési folyamat újrakezdését jelentheti.

Vannak olyan kritikai vélemények, amelyek szerint ez a metódus (csak azután továbblépni a következő lépcsőfokra, miután az előző tökéletesen elkészült) csak triviális rendszerek esetén alkalmazható, a valóságban nem.

Javított vagy továbbfejlesztett vízesés modell



V modell

A vízesés modell továbbfejlesztett változata. Az 1980-as években a német védelmi minisztérium alakította ki. Legfőbb szempontok:

- A kockázat és költségek csökkentése
- Biztonság és a minőség biztosítása

A V modell lényege

A V egyik szárán a tervezés, a másikon a tesztelés lépései állnak, azaz

A tervezés minden egyes elméleti lépéséhez tartozik egy számítógépen végzett tesztelési fázis.

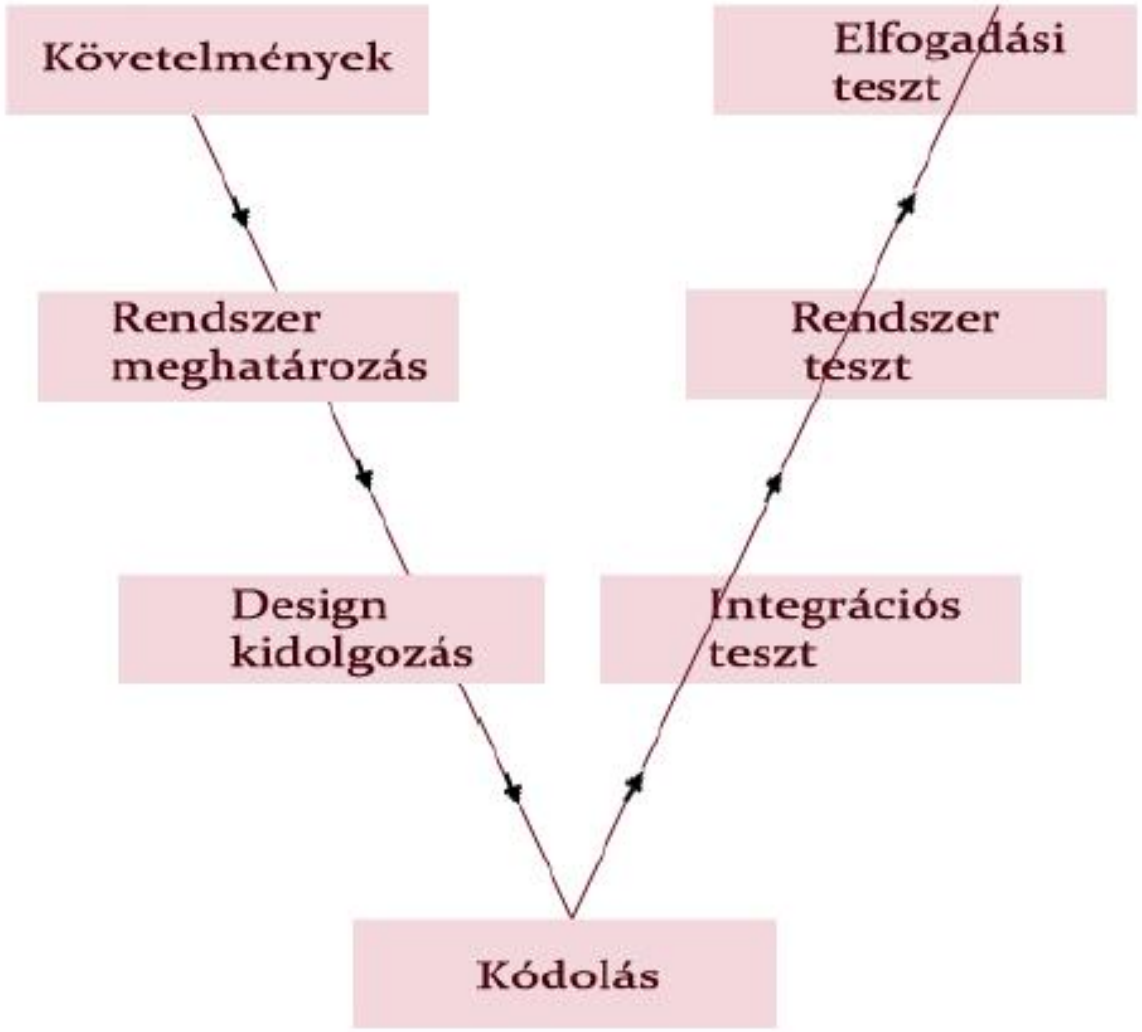
Az életciklus modellben egyensúlyba kerül a verifikáció és a validáció.

A V modell jellemzői

A V modell előnye, hogy a megrendelő/ felhasználók jobban részt tudnak venni a munkában, mivel a fejlesztéssel egy időben zajló tesztelésben részt tudnak venni, és tapasztalataikat figyelembe vehetik a fejlesztők.

Biztonságkritikus rendszerek fejlesztéséhez kiváló.

Design és implementáció



Tesztelés és verifikáció

MILTON FRIEDMAN EGYETEM

Evolúciós modell

Lényege, hogy a fejlesztők létrehoznak egy kezdeti információs rendszert, majd azt a felhasználókkal véleményeztetik, majd sok-sok verzión keresztül addig finomítják, amíg a minden igényt kielégítő rendszert el nem érik.

Az evolúciós fejlesztés két típusa

FELTÁRÓ FEJLESZTÉS

A felhasználóval történő egyeztetés után az adott alap-követelményeknek megfelelő működőképes rendszert adnak át a felhasználóknak.

A rendszer továbbfejlesztése során az alaprendszer működését látva a felhasználók újabb és újabb követelményeket határoznak meg a rendszer fejlesztői csoport számára, közösen kialakítva a rendszer végleges felépítését.

ELDOBHATÓ PROTOTÍPUS KÉSZÍTÉSE

A fejlesztés alapja a felhasználó elvárásainak minél pontosabb megismerése, amelyekre alapozva pontosan definiálhatók azok a tulajdonságok, amelyeket a rendszernek tudnia kell.

Azokat a követelményeket, amelyek nem érthetőek teljesen pontosan, egy-egy prototípusban valósítják meg, és a felhasználó véleménye alapján finomodik a követelmények meghatározása és a rendszer funkcionalitása.

A módszer hátránya, hogy nehezen átlátható a fejlődési folyamat és a változatok nagy száma miatt a pontos dokumentáció szinte lehetetlen, ami megnehezíti az üzemeltetési és karbantartási feladatokat.

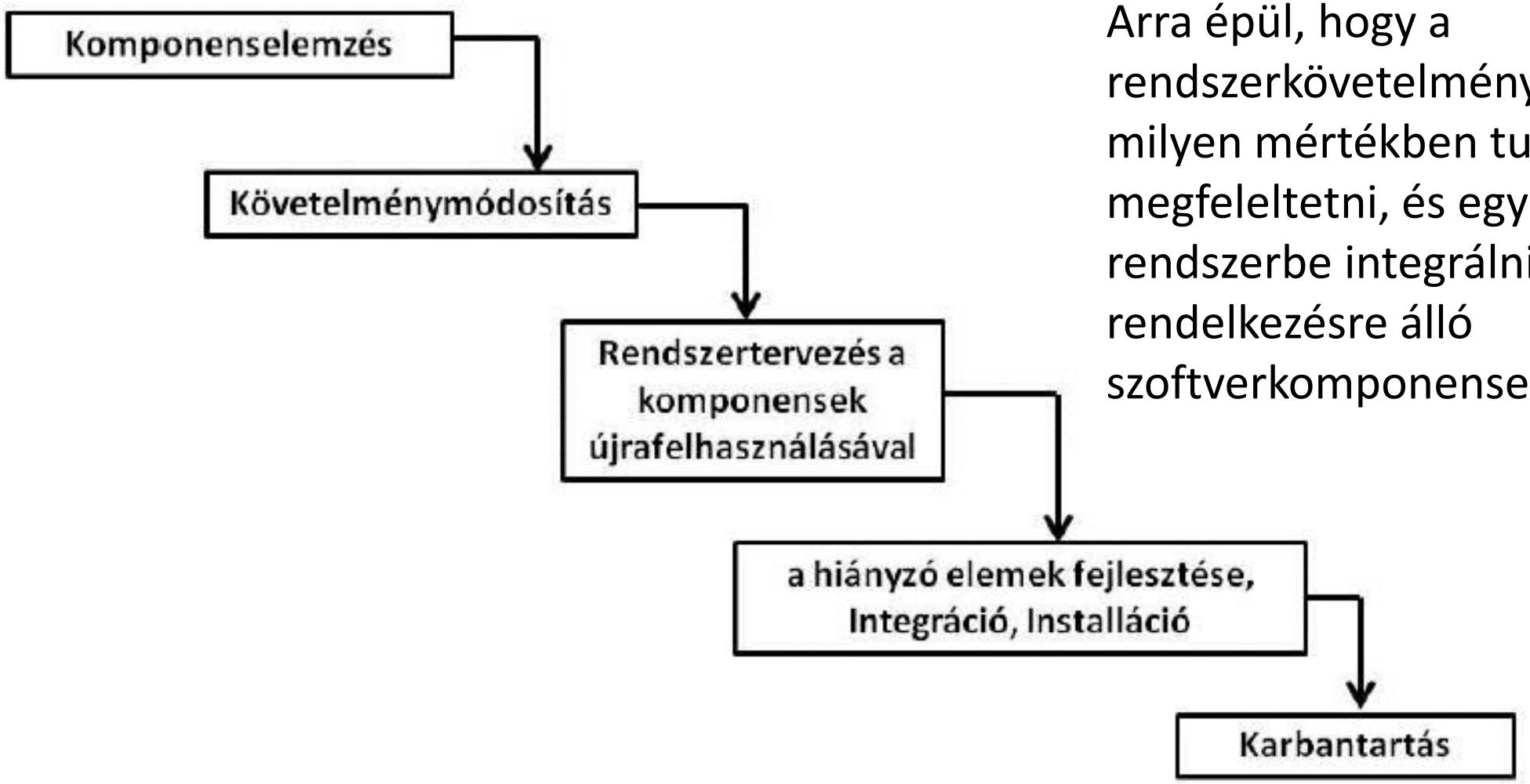
Komponensalapú és újrafelhasználható rendszerfejlesztés

Alapja, hogy szoftverek jelentős részében ismétlődnek egyes szoftverkomponensek.

Az újrafelhasználás során egy már elkészült rendszer egyes komponenseit az új feladatnak megfelelően átalakítják, és így kerülnek az új rendszerbe. Ez jelentősen felgyorsíthatja a rendszerfejlesztés menetét.

A fejlesztés gyorsasága és a követelményeknek való megfelelés szoros kapcsolatban van egymással, optimális esetben viszonylag gyors fejlesztési folyamat is eredményezhet funkcionálisan jól működő rendszert.

Komponens alapú fejlesztés



Arra épül, hogy a rendszerkövetelményeknek milyen mértékben tudjuk megfeleltetni, és egységes rendszerbe integrálni a rendelkezésre álló szoftverkomponenseket.

Lépései

Komponenselemzés: A rendszerkövetelmények alapján a rendelkezésre álló szoftverkomponensek közül megkeressük azokat, amelyek teljesíthetik az elvárásokat.

Követelménymódosítás: A szoftverkomponensek funkcionális leírását felhasználva megfeleltetjük az elvárásokat a rendelkezésre álló komponenseknek. Ha az elvárásnak nem felel meg teljes egészében a rendelkezésre álló komponens, meg kell vizsgálni, hogy a követelmény módosítható-e olyan mértékben, hogy a rendszer integritása megmaradjon, és megfeleljen a komponens funkciójának. Ha ez nem lehetséges, akkor a komponenst kell módosítani, vagy új szoftverkomponenst kell kifejleszteni.

Rendszertervezés a komponensek újrafelhasználásával: ebben a szakasz-ban összeállítjuk a rendszer elemeit, a korábban már definiált újrafelhasználható komponenseket beépítjük a rendszerbe, a hiányzó elemek funkcióját pedig a fejlesztés számára dokumentáljuk.

Spirális fejlesztés

A spirális fejlesztési modellt Barry Boehm amerikai matematikus publikálta a nyolcvanas évek végén.

A modell alapgondolata két dologban is eltér az eddigiektől:

1. A szoftverfejlesztési folyamatot nem egyirányú tevékenység folyamatnak tekinti, hanem egy spirálként reprezentálja. A spirál részei a szoftverfejlesztési folyamat egy-egy ismétlődő fázisát reprezentálják.

Spirális fejlesztés

2. A modell kiemelten foglalkozik a fejlesztés kockázati tényezőinek felmérésével, a kockázatból származó problémákra való felkészüléssel.

Spirális fejlesztés

A spirál minden egyes ciklusát négy szektorra oszthatjuk fel:

1. **Célok meghatározása**: az adott fázis által kitűzött célok meghatározása.

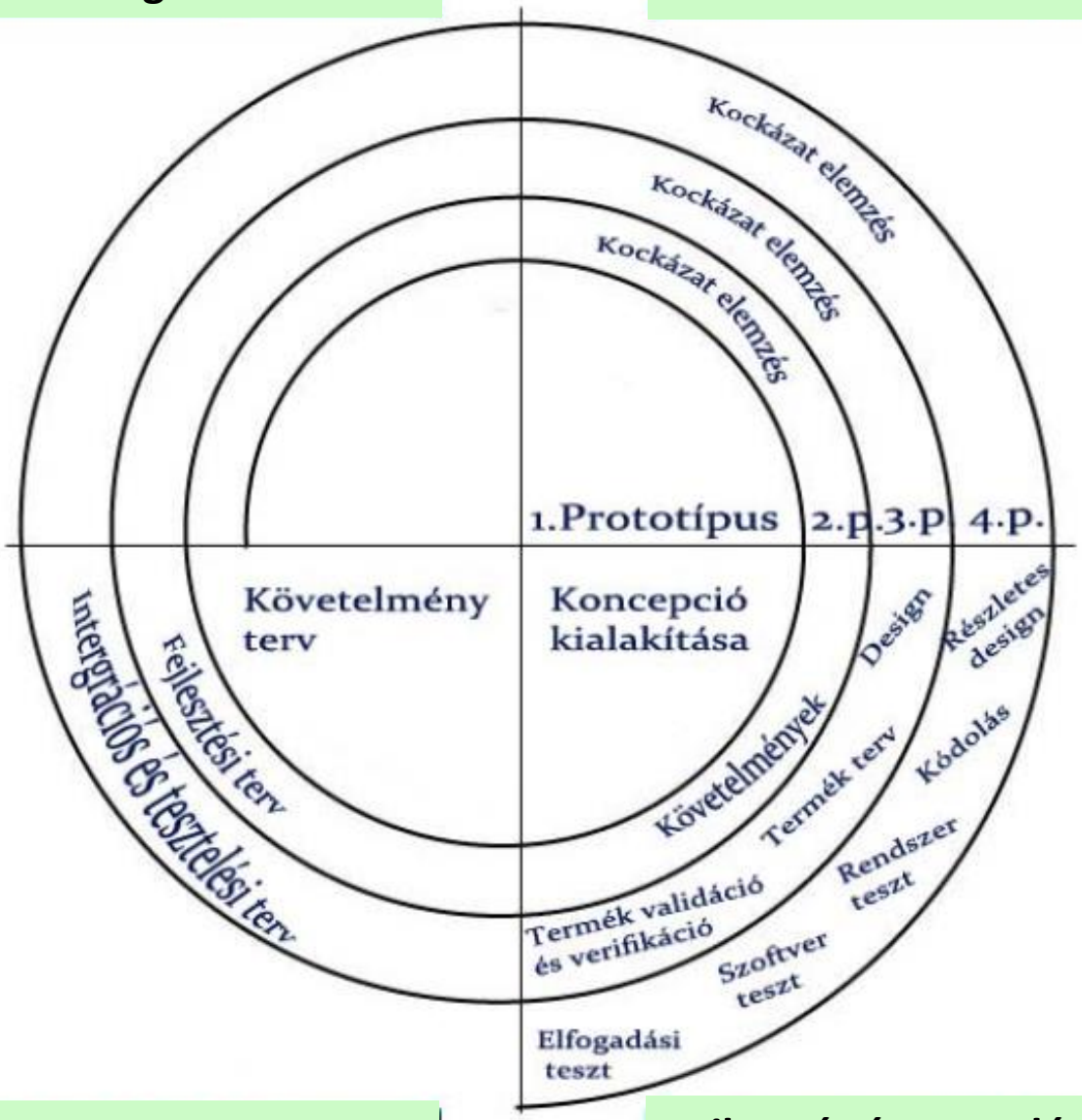
2. **Kockázati tényezők figyelembe vétele**: minden egyes felismert kockázati tényező esetén lépéseket kell tenni a kockázat csökkentése érdekében.

3. **Fejlesztés és tesztelés**: a kockázat kiértékelése után egy fejlesztési modellt kell választani a problémának megfelelően. Adott esetben minden egyes spirálban más-más fejlesztési modell alkalmazható, a kockázatanalízis eredményeként választható a fejlesztési modell.

4. **Tervezés**: a tervezési folyamat során ekkor kell eldönteni, hogy folytatódjon-e a fejlesztési folyamat egy következő ciklussal, vagy sem. Ha a folytatás mellett döntünk, akkor kezdődik a következő kör a célok meghatározásával.

Célok meghatározása

Kockázat elemzés



Tervezés

Fejlesztés és tesztelés

MILTON FRIEDMAN EGYETEM

Iteratív- inkrementális modell

A spirális modellhez hasonlóan ennek a modellnek is az iteráció az alapja.

A megvalósítandó célokat prioritásuk sorrendjében valósítja meg, és az iterációk végén az adott rendszer a követelmények egyre nagyobb hányadának felel meg.

A fejlesztés során ún. inkremenseket hoznak létre. Az inkremens egy 20 000 sornál kevesebb sort tartalmazó egység, amely legalább egy rendszerfunkciót megvalósít.

Iteratív- inkrementális modell

A rendszer nem bonyolódik, hanem szélességében bővül.

A legfontosabb funkciókat már a fejlesztés elején megvalósítjuk, az első inkremens képében, amit később a többi inkremenssel bővítünk.

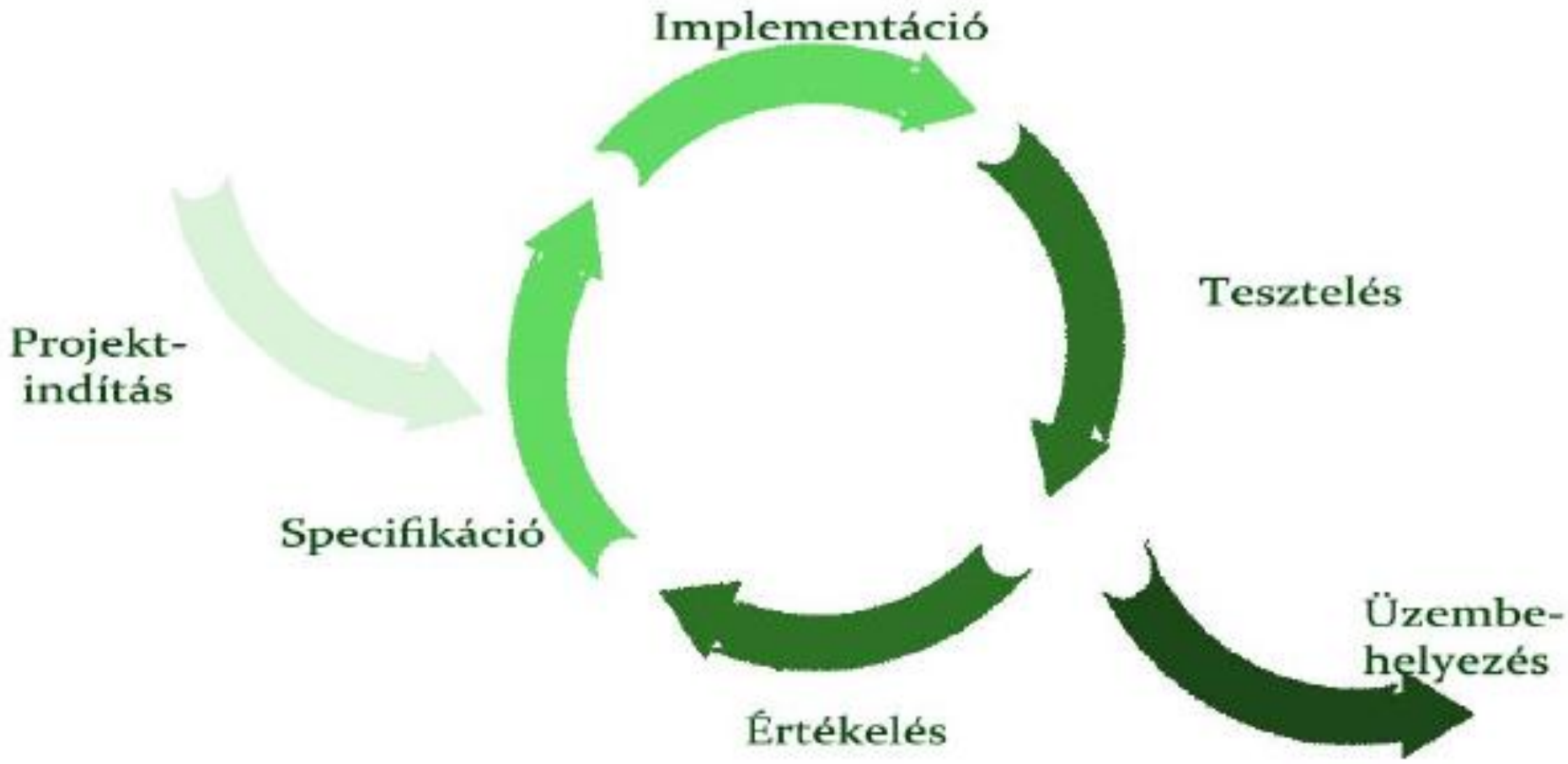
Iteratív- inkrementális modell

A megrendelő/felhasználó az első inkremens létrejötte után használhatja a rendszert, a legfontosabb szolgáltatásokkal.

A kockázat alacsony, hiszen már nagyon korán elkészül egy működőképes rendszer.

A modell jól használható, ha a követelmények nem teljesen tisztázottak, hanem a projekt megvalósulásával párhuzamosan kristályosodnak ki.

Iteratív- inkrementális modell



Gyors Alkalmazásfejlesztés (RAD)

A módszer elemei:

- ciklikus fejlesztés,
- működő prototípusok létrehozása, és a
- szoftverfejlesztést támogató számítógépes programok (pl. integrált fejlesztői környezetek) használata.

Gyors Alkalmazásfejlesztés (RAD)

Rendszerint kompromisszumokkal jár a használhatóság, a tulajdonságok és a program futási sebessége terén, amit ellensúlyoz a jelentősen lecsökkent fejlesztési idő, a felhasználói igények magas fokú kielégítése, és a grafikus felhasználói felület.

RUP (Rational Unified Process)

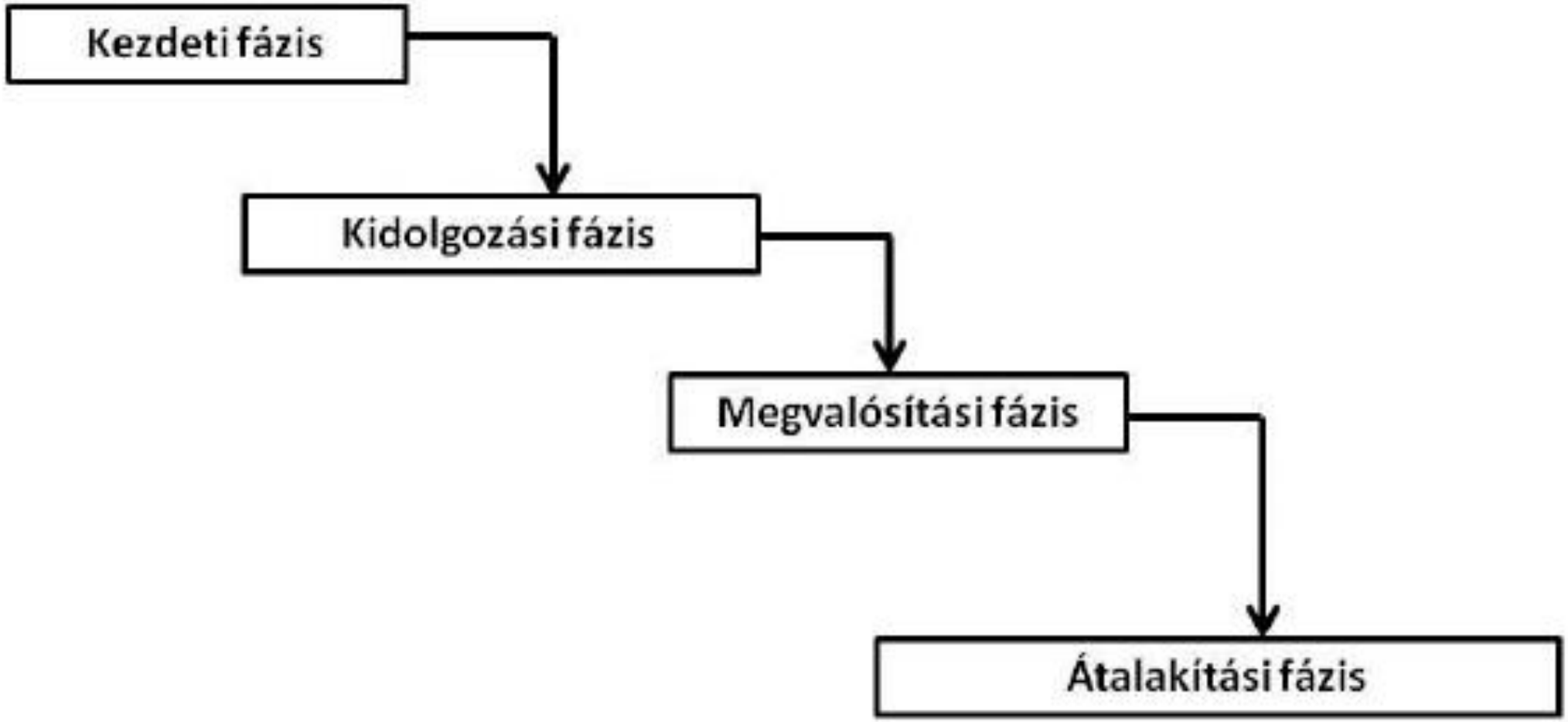
A RUP a rendszerfejlesztés folyamatát alapvetően három dimenzióval írja le:

1. dinamikus perspektívával, amely a modell fázisait mutatja;

2. statikus perspektívával, amely a végrehajtandó folyamattevékenységeket mutatja;

3. gyakorlati perspektívával, amely jól hasznosítható gyakorlatokat javasol a folyamat alatt.

RUP rendszerfejlesztési fázisok



RUP rendszerfejlesztési fázisok

Kezdeti fázis

A kezdeti fázis (inception) során a rendszerrel szembeni elvárásokat és a rendszer funkcionalitását olyan mélységig kidolgozzák, hogy az alapján a fejlesztési folyamat részletesen tervezhető lesz.

Kidolgozási fázis

A kidolgozási fázisban (elaboration) részletekbe menően meghatározzák, hogy a felhasználók hogyan fogják használni a rendszert, és a rendszer hogyan fog felépülni, valamint elkészül az ún. stabil alaparchitektúra terve (architecture baseline). Az alaparchitektúra segítségével a teljes fejlesztés folyamata ütemezhető és a költségei is tisztázhatók.

RUP rendszerfejlesztési fázisok

Megvalósítási fázis

A megvalósítási fázis (construction) során a teljes rendszert kifejlesztik. A folyamat a rendszertervre, a programozásra és a tesztelésre fókuszál. A rendszer különböző részei párhuzamosan fejleszthetők, majd ez alatt a fázis alatt integrálhatók. A fázis végére elkészül a működőképes szoftverrendszer, és a hozzá kapcsolódó dokumentáció.

Átalakítási fázis

Az átadási fázis (transition) során a rendszer működésének tesztelése zajlik, melynek tapasztalatait részletesen dokumentálják. A tesztelés eredményeképpen döntenek a rendszer átalakításáról vagy késszé nyilvánításáról.

Agilis szoftverfejlesztés

Az Agile (agilis) szoftverfejlesztési módszer egy sikertelen tanácskozás eredménye, amelyet 2001 februárjában a vezető szoftverfejlesztő cégek szakemberei tartottak az amerikai Utah államban.

A tanácskozás során teljesen új módszertant nem tudtak kidolgozni, de létrehozták az Agile Manifesto kiáltványt, amely összefoglalása azoknak az elgondolásoknak, amelyeket a résztvevők fontosnak tartottak leszögezni.

Az Agilis Kiáltvány

A szoftverfejlesztés jobb módjait fedezzük fel azáltal, hogy csináljuk, és segítünk másoknak is csinálni. Ennek során az alábbi hangsúly eltolódásokat találtuk:

- Egyének és interakcióik, szemben az eljárásokkal és eszközökkel.
- Működő szoftver, szemben a teljes körű dokumentációval.
- Együttműködés a megrendelővel, szemben a szerződésről való alkudozással.
- Változásokra való reagálás, szemben a terv követésével.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

Agilis módszertan

Az agilis módszertanok jellemző tulajdonságai a kiáltvány alapelveit tükrözik:

A fejlesztési folyamatok modulárisak;

Rövid, fix ciklusú (1-6 hetes) iterációk teszik lehetővé a gyors ellenőrzést és korrekciót;

Szükségtelen tevékenységek megszüntetése (kevesebb dokumentáció, előzetes átfogó dizájn kialakításának mellőzése);

Alkalmazkodás az új kockázatokhoz;

Ember-orientált, kollaboratív és kommunikatív munkastílus.

Az agilis módszertan születése

2001-ben 17 szoftverfejlesztő megfogalmazott egy kiáltványt az agilis szoftverfejlesztésért, mely szerint többre tartják:

Az egyéneket és a személyes kommunikációt a módszertanokkal és eszközökkel szemben;

A működő szoftvert az átfogó dokumentációval szemben;

A megrendelővel történő együttműködést a szerződéses egyeztetéssel szemben;

A változás iránti készséget a tervek szolgai követésével szemben.

Extrém programozás (XP)

A módszertan neve abból a koncepcióból származik, hogy a hagyományos fejlesztési módszertanokban bevált jó gyakorlatok szélsőséges használata még jobb eredmények elérését teszi lehetővé.

Egy szemléletes példája az agilis módszertanoknak

Noha az alapelvei és gyakorlatainak egy része könnyen befogadható, más pontjai jelentősen leszűkítik az alkalmazhatóságát a magasan képzett csapatokra.

Az XP szerveződése

Öt érték köré szerveződik, melyek egymással kombinálva teszik könnyebbé a siker elérését.

kommunikáció,

egyszerűség,

visszajelzés,

bátorság,

amelyek az *egymás iránti tiszteletre* épülnek

Kommunikáció, egyszerűség, bátorság

A megfelelő **kommunikációt** több folyamat is segíti, mint a páros programozás, az egységtesztek készítése, vagy épp a közös becslések.

Az **egyszerűség** szem előtt tartása arra bátorítja a résztvevőket, hogy inkább készüljön egy minimalista funkció alacsony költséggel, és csak ha szükséges, akkor kerüljön bővítésre.

- Ez a szemlélet azzal a veszéllyel jár, hogy egy több lépésben bővített funkció összességében nagyobb erőfeszítésbe kerülhet, viszont előnyt jelenthet akkor, ha már az első verzió is elégségesnek bizonyul, vagy időközben az egész szükségtelenné válik az igények változása miatt.

A **bátorság** ahhoz kell, hogy a programozó csapat legyen nyitott arra, hogy adott esetben kidobja a már elkészült kódokat, vagy alapvetően tervezze át a szoftverdizájnt, mikor felfedezi annak hibáit.

XP - Visszajelzés

Visszajelzés
- Programozók felé
- Ügyfél felé

A programozók felé részben az egységtesztek biztosítanak **visszajelzést**, melyek által azonnal képet kapnak adott funkció jelenlegi állapotáról.

Az ügyfél pedig a becslések során kap visszajelzést a fejlesztőktől arra vonatkozóan, hogy az általa kitalált sztorik (a funkciók leírásai) milyen minőségűek. Az elkészült funkcionalitás minél korábbi éles üzembe állításával a résztvevők mindannyian megfigyelhetik a munkájuk és döntéseik eredményét valós körülmények között, és a későbbiek folyamán felhasználhatják ezeket a tapasztalatokat.

Néhány gyakorlat az extrém programozás eszköztárából

A megvalósítandó feladatokat bontsuk az ügyfél számára értéket teremtő, önmagukban is értelmezhető funkciókra.

Kisebb csomagok kiadásával érdemes felépíteni a rendszert, amit minél hamarabb üzembe kell állítani, majd az éles üzem során tovább lehet fejleszteni. Az ügyfél által a legfontosabbnak ítélt részletek kerülnek először megvalósításra.

Minél egyszerűbb dizájnra kell törekedni, a feleslegesen elbonyolított részleteket azonnal el kell távolítani a programból.

Az extrém programozás egy tesztek által vezérelt fejlesztési módszer.

- Az automata tesztek megírásának szükségessége segít megelőzni a hatókör akaratlan túllépését, meghatározza a fejlesztési folyamat ritmusát, továbbá bizalmat épít a csapattagok közt (mivel kizárólag működő kód kerül a rendszerbe).

Csak a kódot és a teszteket tekinti értéknek, egyéb dokumentáció elkészítését nem.

A folyamatos refaktorálás (a forráskód kisebb-nagyobb mértékű megváltoztatása valamilyen aspektus szerinti optimalizálás céljából) biztosítja, hogy a kódban ne maradjon duplikáció, egyszerűsödjön, vagy rugalmasabb, hatékonyabb legyen.

A csapat minden tagja végezhet módosításokat mások által írt kódrészeken is.

Az XP használhatósága

A módszertan használhatóságát befolyásolja az üzleti környezet, melyben működnie kell.

Nehezen, vagy egyáltalán nem lehet alkalmazni, ha

- a megrendelők valamilyen okból ragaszkodnak a részletes dokumentációhoz,
- ha nem lehetséges a fejlesztői és az éles környezetek gyakori integrációja,
- ha a fejlesztőcsapat nem tud fizikailag azonos helyen (vagy nagyon jó kommunikációs lehetőséggel) dolgozni.

SCRUM

A Scrum kifejezés eredetileg a rögbihez kapcsolódik

- a játék újraindításának egy módját jelenti.

A termékfejlesztéssel kapcsolatban 1986-ban jelent meg először a szakaszokkal élesen elválasztott módszerek alternatívájaként.

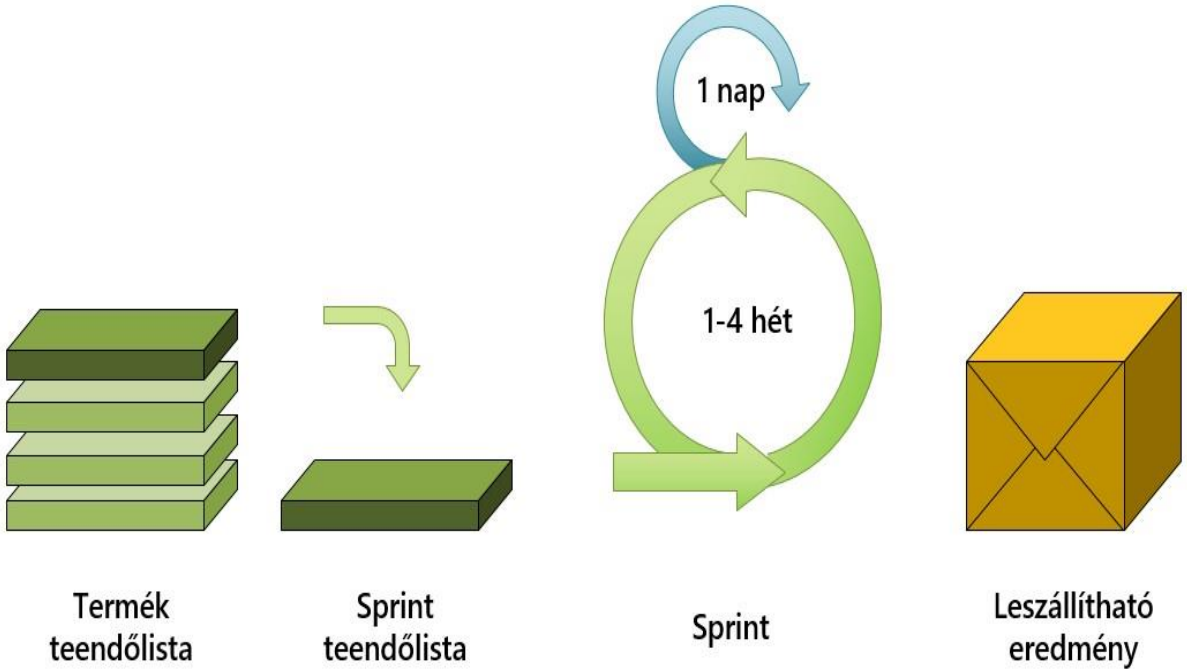
Első formalizált leírása 1995-ből származik, ezután egyre nagyobb népszerűségre tett szert.

A Scrum az agilis módszertanok közül a leggyakrabban használt

Nem kizárólag a szoftverfejlesztési projektek során alkalmazható, mégis használói között többségben vannak az IT területen dolgozók (Scrum Alliance).

A Scrum megközelítés szerint a kereszt-funkcionális csapat a projekt kezdetétől a végéig együtt dolgozik, és közös munkájuk során bontakozik ki a fejlesztés folyamata, aminek egyes szakaszai (követelmények felmérése, tervezés, megvalósítás, tesztelés stb.) átfedésben vannak egymással

SCRUM működése



A Scrum csapat

A csapaton belül három szerep különböztethető meg, a:

Terméktulajdonos

- A Terméktulajdonos egyszemélyben felelős a projektért, a Fejlesztők munkájáért és a megvalósításra váró feladatok összegyűjtéséért, leírásáért, menedzseléséért

Scrum Mester

- A Scrum Mester feladata biztosítani, hogy a csapat a Scrum szabályainak, gyakorlatainak és értékeinek figyelembevétel működjön. A napi munka során segít lebonyolítani a különböző megbeszéléseket és elhárítja a felmerülő akadályokat a Fejlesztőcsapat munkája elől.

Fejlesztő

- magában foglal minden olyan személyt, aki hozzájárul a termék létrehozásához, függetlenül attól, hogy milyen tevékenységet végez.

A Scrum eszközei, gyakorlatai

Termék teendőlista

Sprint

Sprint teendőlista

Scrum Sprint

Sprint: egy 1-4 hét (a projekt során lehetőleg végig fix) időtartamú munkafázis, mely során egy előre meghatározott cél érdekében megvalósításra kerülnek a termék teendőlistájának egyes tételei.

A **sprinttervezés** során kerül meghatározásra, hogy milyen funkcionalitás leszállítása legyen a sprint célja, mely feladatok kerülnek be a sprintbe és hogyan fognak ezek elkészülni. A fejlesztési folyamat hagyományos elemei (követelmények elemzése, tervezés, kódolás, szállítás) a sprinten belül történnek a kiválasztott funkciókra vonatkozóan. A csapat a korábbi sprintek tapasztalatai alapján határozza meg, hány pontot tud megvalósítani egy sprint alatt.

A **napi scrum megbeszélés** a sprint során rendszeresen megtartott rövid (10-15 perces) beszélgetés, mely során a csapat meggyőződik az előrehaladás mértékéről és azonosítja a felmerülő akadályokat.

Az **áttekintés** a sprint végén megtartott megbeszélés, melyen a csapat mellett részt vesz a megrendelő is, hogy együtt átnézzék a sprint során elkészített funkciókat, illetve szükség esetén módosítsák a termék teendőlistát.

Az áttekintés után a Fejlesztőcsapat egy **visszatekintő** megbeszélést is tarthat, ahol dedikáltan a fejlesztés folyamatának, gyakorlatainak és eszközeinek elemzése történik abból a célból, hogy javítsák a tevékenységük hatékonyságát.

SCRUM Sprint teendőlista

A termék teendőlistának azon elemei, melyek az adott sprint elején kiválasztásra kerültek.

Az egyes feladatok tartalma, leírása a sprint során kizárólag a Fejlesztőcsapat által bővíthető vagy módosítható, ahogy egyre átfogóbb képet kapnak a konkrét teendőkről.

A fentieken kívül a Scrum nem definiál konkrét fejlesztési vagy projektmenedzsment technikákat, azonban alkalmazható más módszertanokkal (akár a PRINCE2-vel és az extrém programozás gyakorlataival) kombinálva is.

- Némi módosítással akár olyan projektek során is használható, ahol lényegesen több ember munkáját kell koordinálni. Ezekben az esetekben több csapatot lehet alkotni, és a csapatok közötti együttműködés egy úgynevezett „Scrum of Scrums” megbeszélés során történhet, ahová minden csapat egy-egy főt delegál.

SCRUM - Megjegyzések

Mivel a Scrum valójában az egész projekt irányítását célozza, ez megnehezíti az alkalmazását olyan szervezetekben, melyek nem kellően elkötelezettek a módszer mellett, vagy ahol a projektcsapattal szemben nincs meg a teljes bizalom a vállalatirányítási oldalról.

Kisebb csapatok esetén az is problémát jelenthet, hogy a Scrum Mester irányító szerepe a hagyományos projektmenedzsmenthez képest kevésbé produktív.

- Ez ugyan lehetővé teszi, hogy egyszerre több csapatban dolgozzon, egyúttal viszont ennek megfelelő környezetet igényel, hiszen szerepe az érdekellentétek miatt nem vonható össze sem a Terméktulajdonos, sem a Fejlesztők szerepeivel. Erre hatékony megoldás lehet egy külső Scrum Mester alkalmazása, esetleg a módszertan szervezeti kultúrába való beépítése és egy belső, független munkatárs kinevezése.

Egy vállalaton kívüli Terméktulajdonos, aki csak közvetíti az üzleti érdekeket, nem feltétlenül tudja teljes elhivatottsággal képviselni a megbízót, illetve nincs kellő rálátása az egyes feladatokra, így az előzetes elvárásokhoz képest több támogatásra lehet szüksége.

- Egy olyan szervezetben, ahol a képességek és a tudáshalmaz házon belül is rendelkezésre állnak, célszerűbb lehet, ha közvetlenül érdekelt, belső munkatárs látja el a Terméktulajdonos szerepét még akkor is, ha a módszertannal kapcsolatban nincs tapasztalata.
- A Scrum egyszerű, könnyen elsajátítható, ezzel szemben a vállalati folyamatok és igények mélyreható ismerete hosszú évek alatt alakul ki, ez utóbbi pedig a módszer ismereténél sokkal értékesebb eszköz a projekt számára.

A SCRUM és XP modell összehasonlítása

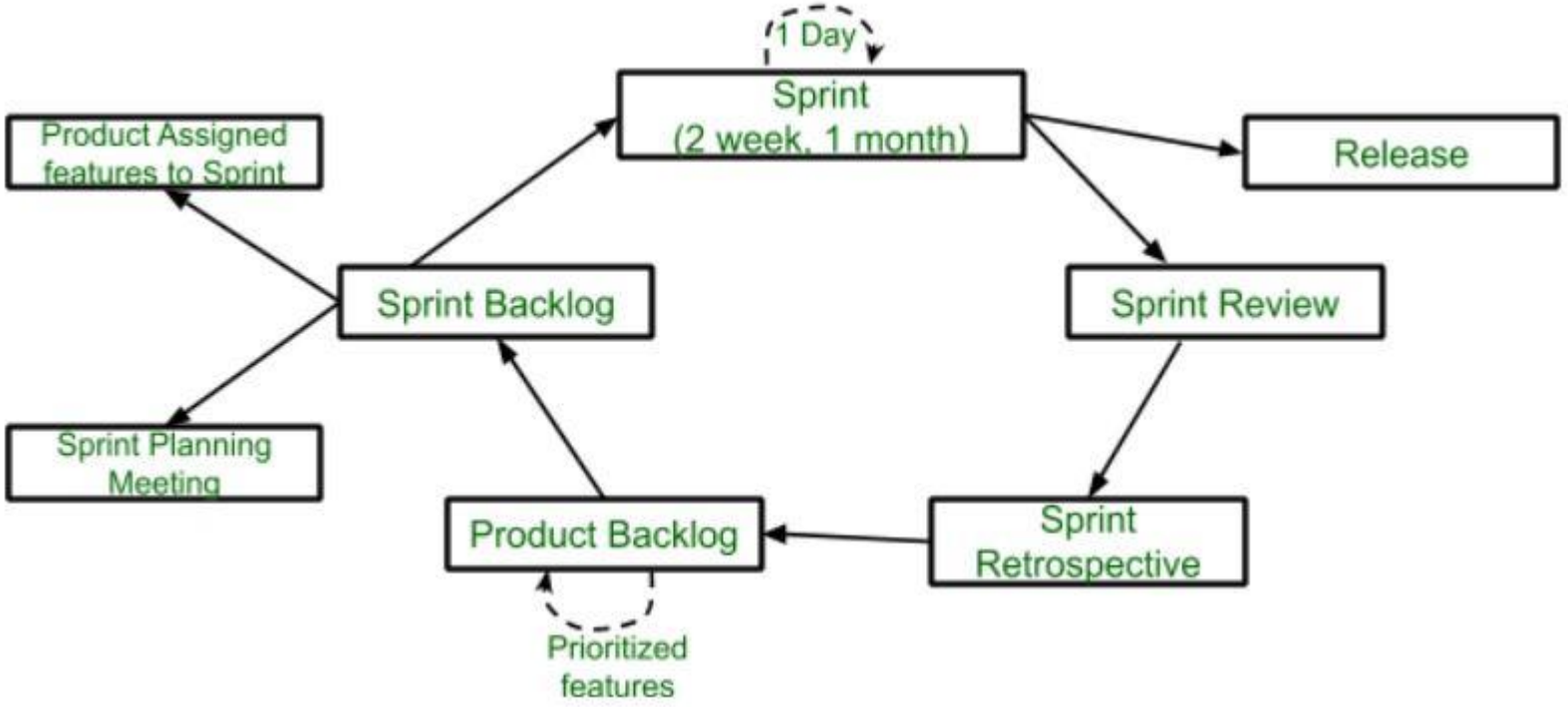
Scrum Framework:
Scrum is the type of Agile framework. It is a framework within which people can address complex adaptive problem while productivity and creativity of delivering product is at highest possible values. Scrum uses Iterative process.

Extreme Programming (XP):
Extreme Programming is one of the most important models of Agile framework. This model emphasizes team-work and customer satisfaction as well. The five basic component of Extreme Programming are:

1. Communication
2. Simplicity
3. Feedback
4. Respect
5. Courage

Scrum	Extreme Programming (XP)
In Scrum framework, team work in iterations called Sprint which are 1-2 month long.	In Extreme Programming(XP), teamwork for 1-2 weeks only.
Scrum model do not allow changes in their timeline or their guidelines.	Extreme Programming allow changes in their set timelines.
Scrum emphasizes self-organization.	Extreme Programming emphasizes strong engineering practices
In Scrum framework, team determines the sequence in which the product will be developed.	In Extreme Programming, team have to follow a strict priority order or pre-determined priority order.
Scrum framework is not fully described. If you want to adopt it then you need to fill the framework with your own frameworks method like XP, DSDM or Kanban.	Extreme Programming(XP) can be directly applied to a team. Extreme Programming is also known for its **Ready-to-apply** features.

SCRUM életciklusa



XP Életciklusa