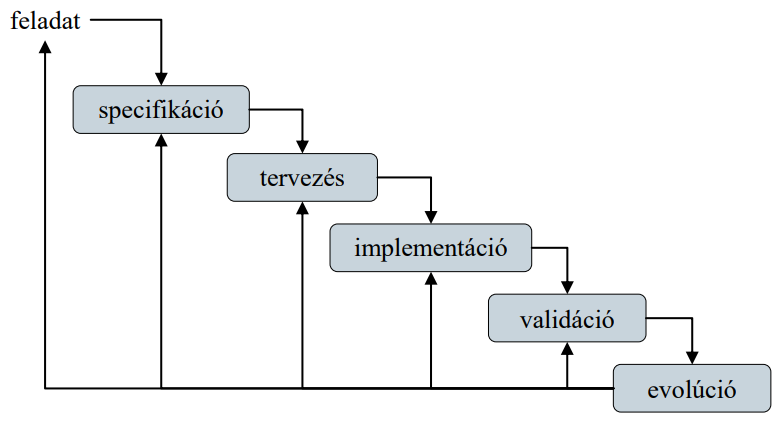
# Szoftverfejlesztési modellek

A szoftverfejlesztési modell határozza meg az életciklus egyes fázisai közötti kapcsolatot, időbeliséget.

A legegyszerűbb fejlesztési modell a **vízesés** (waterfall) modell, amelyben az egyes fázisok lineárisan követik egymást. Ez a modell:

* előre megtervezi a projekt időtartamát, ráfordításait
* elvárja minden fázis megfelelő dokumentálását, amely tartalmazza annak eredményeit
* előnyei: jól strukturált, dokumentált folyamatot biztosít, az ügyfél tudja mire számíthat: mennyibe fog kerülni, mikor készül el, mit fog tudni a program.
* hátrányai: nem teszi lehetővé a követelmények megváltoztatását, nem készül fel az esetleges nehézségekre (nincs kockázatkezelés), nehéz visszalépni, későn derülnek ki a hibák
* akkor használjuk, amikor pontosan tudjuk, milyen eredményt szeretnénk, és a követelmények nem változnak.



Az **inkrementális** (incremental) modell több, egyre többet tudó változatban állítja elő a szoftvert:

* minden változat újabb funkciókkal bővíti a szoftvert, a fázisok rövidek, gyors visszajelzésekkel a felhasználói oldalról
* előnyei: gyorsan alkalmazkodik a változó követelményekhez, a felhasználó jobban követheti a fejlesztési folyamatot
* hátrányai: kevésbé menedzselhető, tervezhető, áttekinthető; nehezebben validálható

## Agilis szoftverfejlesztés

Az agilis szoftverfejlesztés (agile software development) célja a gyors alkalmazásfejlesztés megvalósítása inkrementális alapon:

* a szoftver folyamatos fejlesztés és kiadás alatt áll (continuous delivery), a változtatások minden lépésben beépíthetőek (welcome changes)
* a működő szoftver az előrehaladás mérőeszköze,
* a fejlesztést általában önszerveződő, kis csapatok végzik, megosztott felelősséggel, folytonos interakcióval, gyors visszajelzésekkel

Az agilis fejlesztés alapelveit az Agilis Kiáltványban fogalmazták meg: <http://agilemanifesto.org/iso/hu/manifesto.html>

Az agilis fejlesztés:

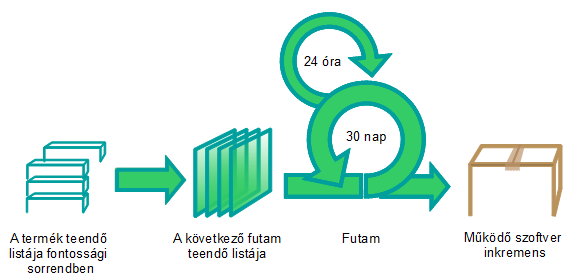
* előnyei: jól alkalmazkodik a változtatásokhoz, az ügyfeleket is bevonja a fejlesztésbe, hatékonyabbá teszi a fejlesztési folyamatot, hamarabb megtalálják a hibákat, mindig van működő termék
* hátrányai: (különösen nagyobb szoftverek esetén) a megvalósításhoz tapasztalt fejlesztők kellenek, a végső termék lényegesen különbözhet az eredetileg tervezettől
* akkor használjuk, amikor az ügyfelek módosíthatnak az igényeiken, és nem tudjuk pontosan, milyen végeredményt szeretnénk

Az agilis fejlesztés elveit több módszertan ültette át a gyakorlatba:

* pl. Extreme programming, **Scrum**, Kanban, …
* a módszertanok nem szeparáltak, a gyakorlatban sokszor vegyítve alkalmazzák őket

## Scrum

A Scrum egy agilis szoftverfejlesztési módszer. Jellegzetessége, hogy fogalmait az amerikai futballból meríti. Ilyen fogalom maga a Scrum is, amely dulakodást jelent. A módszertan jelentős szerepet tulajdonít a csoporton belüli összetartásnak. A csoporton belül sok a találkozó, a kommunikáció, lehetőség van a gondok megbeszélésre is. Az ajánlás szerint jó, ha a csapat egy helyen dolgozik és szóban kommunikál.



A Scrum által előírt fejlesztési folyamat röviden így foglalható össze:

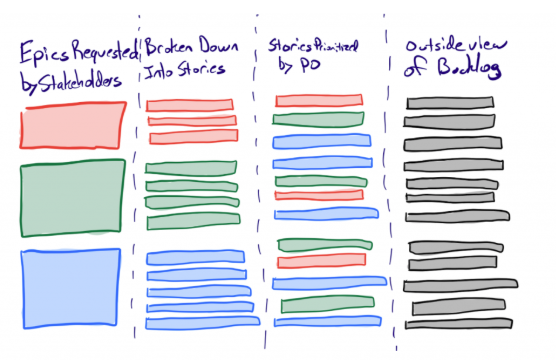
* A Product Owner létrehoz egy Product Backlog-ot, amelyre a teendők listája van. A teeendőket prioritással kell ellátni és megmondani, mi az üzleti értékük. Ez a Product Owner feladata.
* A Sprint Planning Meetingen a csapat tagjai megbeszélik a Product Owner-rel, és a csapat teljes mértékben megérti, hogy a vevő mit szeretne, majd eldöntik hogy mely teeendők megvalósítását vállalják el (lehetőleg a legnagyobb prioritásúakat). Ehhez a teeendőket kisebb feladatokra bontják, hogy megbecsülhessék mennyi ideig tart megvalósítani azokat.
* Ezután jön a sprint (futam), amely 2-4 hétig tart. A sprint időtartamát az elején fixálja a csapat, ettől eltérni nem lehet. Ha nem sikerül befejezni az adott időtartam alatt, akkor sikertelen a sprint, ami büntetést, általában prémium megvonást von maga után.
* A sprinten belül a csapat és a Scrum Master naponta megbeszélik a történteket a Daily Meetingen. Itt mindenki elmondja, hogy mit csinált, mi lesz a következő feladata, és milyen akadályokba (impediment) ütközött. Akadály csak munkahelyi probléma lehet (például, hogy lejárt az egyik szoftver licence, szükség lenne egy plusz gépre a gyorsabb haladáshoz, vagy több memóriára az egyik gépbe), vagy akár az is lehet, hogy 2 tag megsértődött egymásra. Ilyenkor kell a Scrum Masternek elhárítani az akadályokat, hogy a munka minél gördülékenyebb legyen. A Daily Meeting maximum 15 perc hosszú lehet, és egy előre megbeszélt időpontban, a csapattagok és a Scrum Master jelenlétében történik (mások is ott lehetnek, de nem szólhatnak bele). Érdekesség, hogy nem szabad leülni, mindenki áll, ezzel is jelezve, hogy ez egy rövid találkozó.
* A sprint végén következik a Sprint Review, ahol a csapat bemutatja a sprint alatt elkészült sztorikat. Ezeket vagy elfogadják, vagy nem.
* Majd a Sprint Retrospective találkozó következik, ahol a Sprint során felmerült problémákat tárgyalja át a csapat. A megoldásra konkrét javaslatokat kell tenni.
* Ezek után újra a Sprint Planning Meeting következik, amelyen a következő sprintet tervezik meg.
* A fejlesztett termék az előtt piacra kerülhet, hogy minden teendőt megvalósítottak volna.

A csapatban minden szerepkör képviselője megtalálható, így van benne fejlesztő és tesztelő is. Téves azt gondolni, hogy a sprint elején a tesztelő is programot ír, hiszen, amíg nincs program, nincs mit tesztelni. Ezzel szemben a tesztelő a sprint elején teszttervet készít, majd kidolgozza a teszteseteket, végül, amikor már van mit tesztelni, ellenőrzi azokat.

A Scrum, mint minden agilis módszertan, arra épít, hogy a fejlesztés közben a megrendelő igényei változhatnak. A változásokhoz úgy alkalmazkodik, hogy a Product Backlog folyamatosan változhat. Az erre épülő dokumentumok folyamatosan finomodnak, tehát könnyen változtathatók. A csapatok gyorsan megvalósítják a szükséges változásokat.

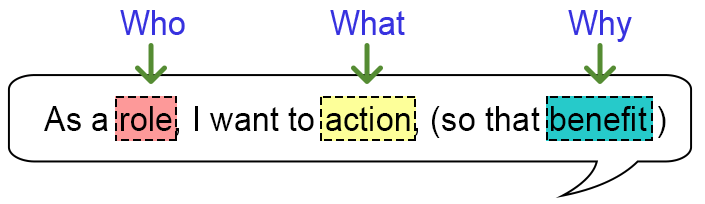
## Backlog készítése

A teeendőket általában felhasználói sztorikként (lásd lejjebb) a projekt elején egy workshopon készítik el. Ezen a megrendelők, a Product Owner és a csapat néhány tagja vesznek részt. A Product Owner ezeket rendezi és fontossági sorrendet állít be. A folyamatot az alábbi ábra mutatja:

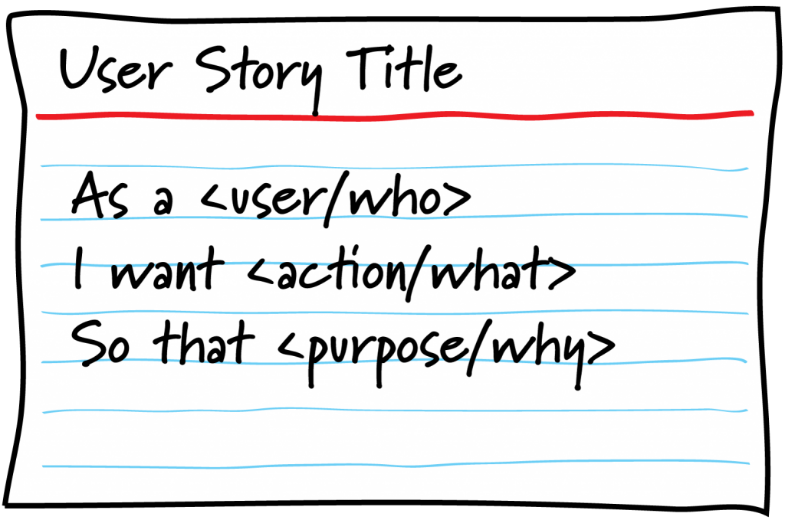


Az első oszlopban nagyobb feladatok vannak (epics), amelyeket kisebb felhasználói sztorikra bontanak a második oszlopban. Ezeket azután prioritás szerinti sorrendbe rendezik (harmadik oszlop), majd bekerülnek a termék backlogjába. Az egyes sprintek elején a backlogból válogatják ki az adott sprintben megvalósítandó sztorikat.

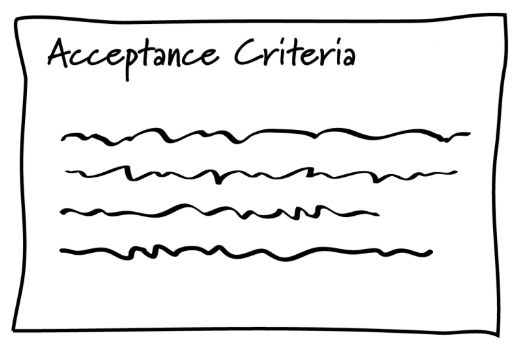
## User stories



A sztorikat eredetileg kártyákra írták:



Ezeket a feladatokat a közös megértés érdekében át kell beszélni a csapatban, majd a hátuljára ráírni az elfogadás feltételeit (Acceptance Criteria):



A feltételek eldöntendőek (igaz/hamis), egyértelműek és teljesek. Általában jelen időben írják, és a feladat előtt hamisak, utána pedig igazzá válnak. Ezeket lehet a teszteléssel elllenőrizni.

Például egy szállásfoglaló oldalon egy user story lehet a következő:

Felhasználóként vissza akarom vonni a foglalást, hogy másik szállást foglalhassak le.

A megbeszélésen a következő kérdések merültek fel:

1. Kap-e a felhasználó visszatérítést, és mennyit? Hogyan kapja meg?
2. Mennyivel előre kell jelezni a visszavonást?
3. Kell-e jelezni a visszavonás elfogadását a felhasználónak? Hogyan?
4. Kell-e jelezni a visszavonás elfogadását a szállodának? Hogyan?

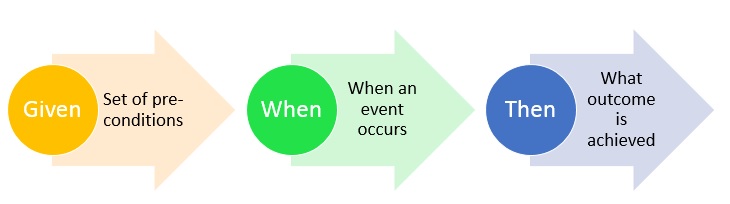
Az elfogadás feltételei lehetnek a következők:

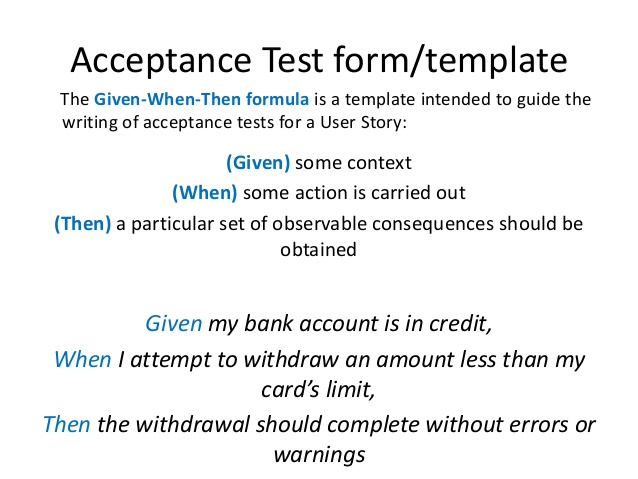
1. Egy prémium tag még az utazás napján is térítésmentesen visszavonhatja a foglalást.
2. Egy nem prémium tagot az ár 10%-ával terheli a program aznapi visszamondás esetén, korábban neki is díjmentes.
3. A felhasználó e-mail értesítést kap a visszamondásról.
4. A szálloda e-mail értesítést kap a visszamondásról.

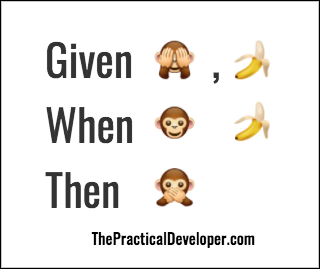
Egy ilyen összetett user story-t több részre is lehet bontani. (Például külön a prémium tagoknak, és külön a többieknek.)

Az elfogadási feltételeknek van formálisabb leírása is. Például:

**Amennyiben** prémium tag a felhasználó, **amikor** aznap visszamondja az utazást, **akkor** nem kell térítést fizetnie  
**GIVEN** prémium tag a felhasználó, **WHEN** aznap visszamondja az utazást, **THEN** nem kell térítést fizetnie.







***Feladat***: Írj néhány user story-t egy online programba történő bejelentkezéshez! Az elfogadási feltételeket is add meg!

## Minimum Viable Product

Általában több ötlet merül fel az alkalmazással kapcsolatban, mint amit meg tudunk valósítani. Ezért többnyire olyan alkalmazás készítésére törekszünk, amely minimális, de a célok már megvalósíthatók vele. Ezt nevezzük Minimum Viable Product-nak (MVP).

A tervezés során vegyük figyelembe Pareto 80/20 törvényét: A felhasználók 80%-a a program funkcióinak 20%-át használja.

## Feladatokra bontás

A user sztorik a felhasználóknak szólnak, azt írják le a felhasználók szemszögéből, hogy mit kell tudnia a programnak.

A kódolás során azonban az egyes sztorikat általában több feladatra kell bontani.

Egy feladat a fejlesztésnek egy olyan része, amelyet egy fejlesztőnek kell elvégeznie.

Minden feladatnak van:

* címe, amellyel tudunk rá hivatkozni,
* leírása, amely a részleteket tartalmazza,
* és egy becslése az elvégzéséhez szükséges időre.

Példa (randikezelő app 😊):

