Szoftvertechnológia és -technikák

12. Előadás - Agilis fejlesztés, Scrum

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Mik a kihívások a fejlesztésben?

- Követelmények megértése
- Követelmények változása
- Pontatlan becslések / keretek tartása
- Adminisztrációs / kommunikációs overhead

- A vízesés modell nem kezeli jól ezeket a kihívásokat, ezért születtek meg az agilis módszertanok

Követelmények megértése 1

- Több résztvevő
  - **Fejlesztő**: fejlesztéshez ért, az üzleti területhez nem
  - **Megrendelő**: üzletileg érdekelt a szoftverben, de nem feltétlen ért se a szoftverhez, se az üzleti területhez
  - **Domain expert, felhasználó**: értene a doménhez, talán tudja, mi segítené a munkáját, de nem mindig kérdezik meg elégszer, illetve nem is feltétlen tudja elmondani elsőre
  - **Megrendelő ≠ felhasználó**: sokszor nem azonosak, pl. kórházigazgató és ápolónő

Követelmények megértése 2

- Rossz irányba ment fejlesztés
  - **Frusztráció** a fejlesztőnek
  - **Pazarlás** és időbeli csúszás a megrendelőnek
  - **Üzleti nehézség**/bukás
- Iteratív fejlesztés
  - Gyakran validálni: „ez az, amit szerettél volna?”
  - Nem kerüli el a félreértéseket, de kordában tarthatóak
  - Más megközelítést igényel: nem lehet rétegenként haladni, horizontálisan kell felosztani a feladatot

A követelmények változása

- Még ha meg is értjük egymást, sok minden változhat
  - Jogi környezet (ld. Uber, AirBnb)
  - Piaci verseny helyzete
  - Technológiai fejlődés
- Megrendelőnek kell tudnia, hogy versenyképes-e, amit szeretne
- Nekünk felkészülni az esetleges változásokra
  - Lehet holnap már mást kér tőlünk
- Erre is az iteratív fejlesztés a megoldás

Verifikáció és Validáció

- **Verifikáció: a specifikációval** összhangban implementálom-e a rendszert?
  - Jól építem-e meg?
  - Klasszikusan ezt teszteljük
- **Validáció: az üzleti igényeket** kielégítő rendszert építek-e?
  - **A jó rendszert építem meg?**
  - Jól lettek definiálva a követelmények?
  - Felhasználó előtti demóval vagy speciális tesztekkel végezzük

Pontatlan becslések 1

- A szoftverfejlesztés kreatív alkotómunka
  - Szobafestés vs. költészet
- Tapasztalat kell a becsléshez
- Így is lehetnek nem várt kihívások, érdemes ráhagyással tervezni
- Elnagyolt, változó, félreértett követelmények tovább nehezítik a helyzetet

Pontatlan becslések 2

- Nehezen tartható keretek!
- Következmények
  - **Több idő = több költség**
  - Ha kifutunk az időből, nem biztos, hogy továbbra is lesznek anyagi erőforrások a fejlesztés fedezésére
  - Csúszással nemcsak nőnek a közvetlen költségek, hanem a korábbi piacra lépésből realizálható haszontól is elesik

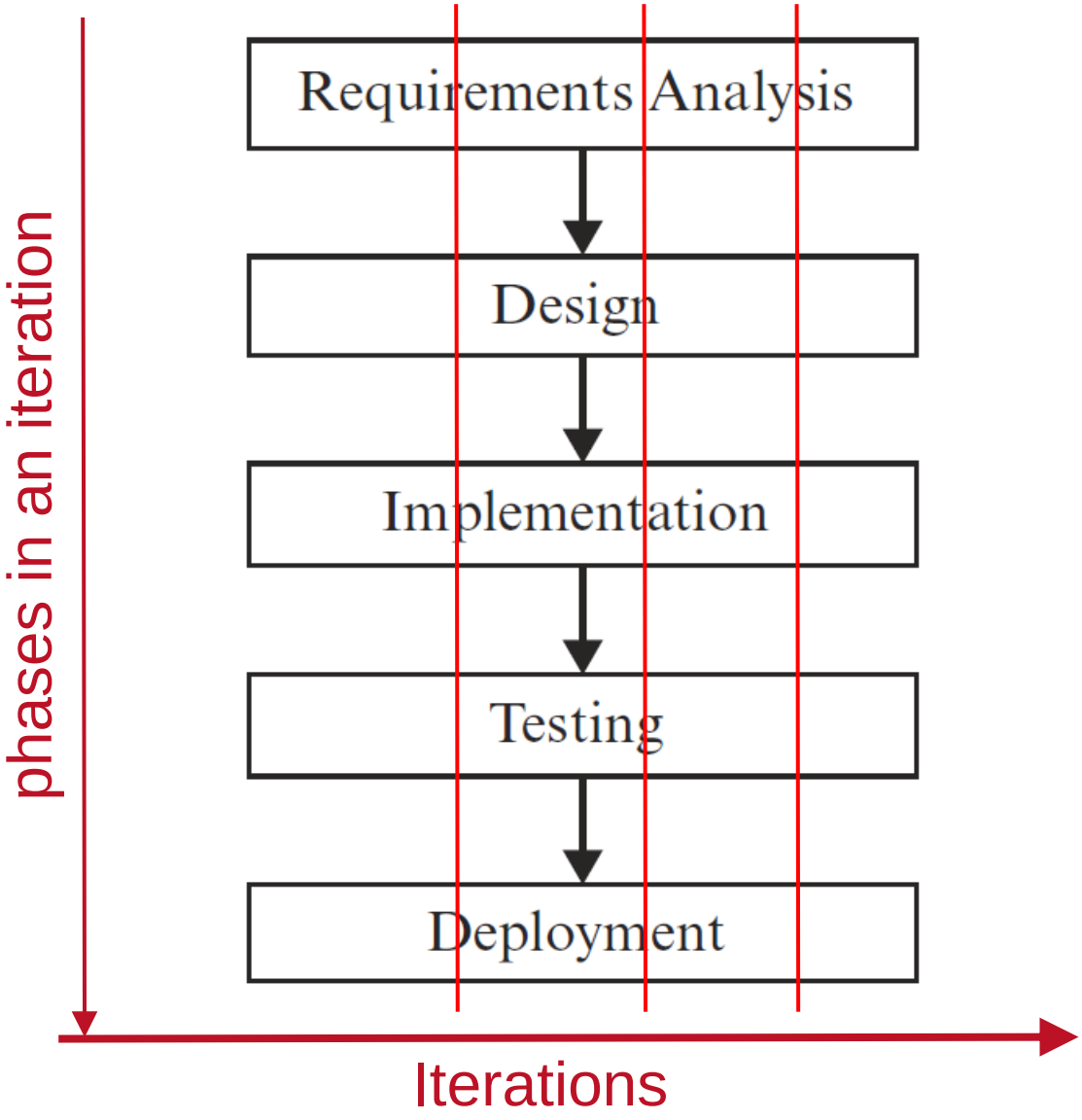
Priorizálás

- A fenti problémák priorizálással mérsékelhetők
- Mindig a legfontosabb elemeken dolgozunk
  - Ami elkészül, az legyen jó és használható…
- Ezek minél előbb elkészülnek, ha valamire végül nem jut pénz/idő, az kisebb jelentőségű dolog
- Kevésbé kerül veszélybe a piacra lépés
- Hogyan?
  - Azt teljesítsük, amit kérnek, ne próbáljuk túlteljesíteni
  - Gondolkodjunk előre, de ne túlzottan; inkább hagyjuk nyitva a bővítés lehetőségét, de ne tegyünk sok energiát túl előre tervezésre

Agilis módszertanok

- Figyelembe veszik a fenti nehézségeket
- **Csapatmunka**, mindenki tud mindenről
- A felhasználót bevonva dolgozunk
  - Látható számára, hol tartunk -> **bizalom**
- **Minimális tervezéssel** induláskor
- **Változásra készen**, azokat befogadba
  - Rendszeresen vizsgáljuk meg, hogy jó-e az irány
- Gyors fejlesztés: **gyakori kiadások**
  - Iteratívan, mindig a legnagyobb prioritásokra fókuszálunk
  - Az instabil követelmények nem tartják fel a fejlesztés
- **Pehelysúlyú, de erősen keretezett** módszertanok

Agilis folyamat modell



Az Agile Manifesto

- Az agilis mozgalom fő dokumentuma
- 2001-ben írta le több neves szoftverguru
- **Célravezetőbb** alternatívák a szoftverfejlesztés támogatására
- A fontosabbak:
  - Kent Beck, Alistair Cockburn, Martin Fowler, Robert C. Martin, Ken Schwaber és Jeff Sutherland
- Értékeket és 12 irányelvet rögzítettek.
- https://agilemanifesto.org/

Agilis irányelvek

1. Egyének és interakció inkább,
   mint folyamatok és eszközök
2. Működő szoftver inkább,
   mint jól dokumentált szoftver
3. Felhasználó bevonása inkább,
   mint szerződések tárgyalása
4. Reagálás a változásokra inkább,
   mint egy terv követése

Az agilis fejlesztés 12 pontja 1

1. A legfontosabb dolog a megrendelő kielégítése úgy, hogy a kezdetektől folyamatosan **értékes szoftvert** szolgáltatunk.
2. **Üdvözöljük a változtatási kérelmeket**, még a fejlesztés késői szakaszában is. Az agilis folyamatok a megrendelő versenyelőnye érdekében hasznosítják a változtatásokat.
3. **Működő szoftver** gyakori kiadása (pár hét – pár hónap) minél rövidebb időszakokkal.
4. Az üzleti megrendelőnek és a fejlesztőknek **napi szinten együtt kell működniük**.

Az agilis fejlesztés 12 pontja 2

5. A projekteket **jól motivált egyének**re kell építeni. Adjuk meg nekik a környezetet és a támogatást, és bízzunk bennük, hogy a rájuk bízott feladatot elvégzik.
6. A leghatékonyabb és leghatásosabb módja az információáramlásnak a csapaton belül és azon kívül a **személyes beszélgetés**.
7. **A haladás mértéke a működő szoftver.**
8. Az agilis folyamatok a **fenntartható fejlesztést** támogatják. A megrendelők, a fejlesztők és a végfelhasználók egy állandó sebességet tudnak fenntartani gyakorlatilag végtelen ideig.

Az agilis fejlesztés 12 pontja 3

9. A **technikai tökéletességre** és a **jó dizájnra** való törekvés növeli az agilitást.
10. Az **egyszerűség** – az el nem végzett munka maximalizálása – alapkövetelmény.
11. A legjobb architektúrát, követelményt és dizájnt **önszerveződő csoportok** tudják kitermelni.
12. Bizonyos időközönként **a csapat** megvitatja, hogyan lehetne hatékonyabb és ennek megfelelően **hangolja a viselkedését.**

Értékek - Kommunikáció

- „What matters most in team software development is communication”
- Minden probléma visszavezethető a kommunikációra
  - Programozó – Programozó
  - Programozó – Ügyfél
  - Manager – Programozó
- Egy cél: minden fejlesztő ugyanúgy lássa a rendszert, ahogy a majdani felhasználók is látni fogják.
- Gyakorlatok
  - Unit testing,
  - Pair Programming,
  - Planning Game, Sit Together, Demo, Retrospective

Értékek - Egyszerűség

- „What is simplest thing that could work?”
- Az agilis módszer arra fogad, hogy...
- YAGNI, KISS
- Az egyszerűség segíti a kommunikációt
- Gyakolatok
  - Refactoring
  - Simple Design, Incremental Design
  - Sit Together

Értékek – Visszacsatolás

- „Az optimizmus szakmai ártalom a programozóknál, és a visszajelzés rá a gyógyír.”
- „Elsőre jól?”
  - Nem tudjuk hogyan
  - Ha tudjuk, holnap már lehet a jó rossz
  - Sok és gyors visszacsatolással tudunk közel kerülni a jó megoldáshoz.
- Gyakorlatok
  - Retrospective, Pair programming, Ten Minute Build, Iteration Demo

Értékek – Bátorság

- Tenni valamit a félelem ellenére
  - Betartani a YAGNI-t
  - Refaktorálni
  - Revertálni
- „Hibázni érték”
- Főleg a többi értéket támogatja
  - Kimondani jót és rosszat
  - Eldobni a rossz megoldásokat
  - Igazi, valódi válaszokat keresni

Értékek – Respect

- Ha nem törődünk egymással, akkor…
- Ha nem törődünk a projekttel, akkor…
- „I am important and so are you”

Scrum

- Egy konkrét agilis keretrendszer
- Előír szerepköröket és eseményeket, de további technikák is belevehetők, amelyek nem mondanak ellent az alapvetéseinek
- Időben keretezett eseményekkel dolgozik, ami alatt be kell fejezni őket
- https://scrumguides.org/index.html
  - > 2020-ban frissítették utoljára
  - > A Scrum Guide 2020 13 oldal

A Scrum csapat

- **Product Owner (PO)**: megrendelőt képviseli
  - Átmenet üzleti és szoftveres világ közt
  - Érti a követelményeket és közvetíti a fejlesztőknek
  - Csapat része
- **Scrum Master (SM)**: a Scrum elvek betartását segíti
  - Nem klasszikus PM
  - Nem a csapat vezetője
  - A Scrumban képzett
- **Development Team**: a fejlesztők szigorú szerepkörök nélkül

Hajó analógia

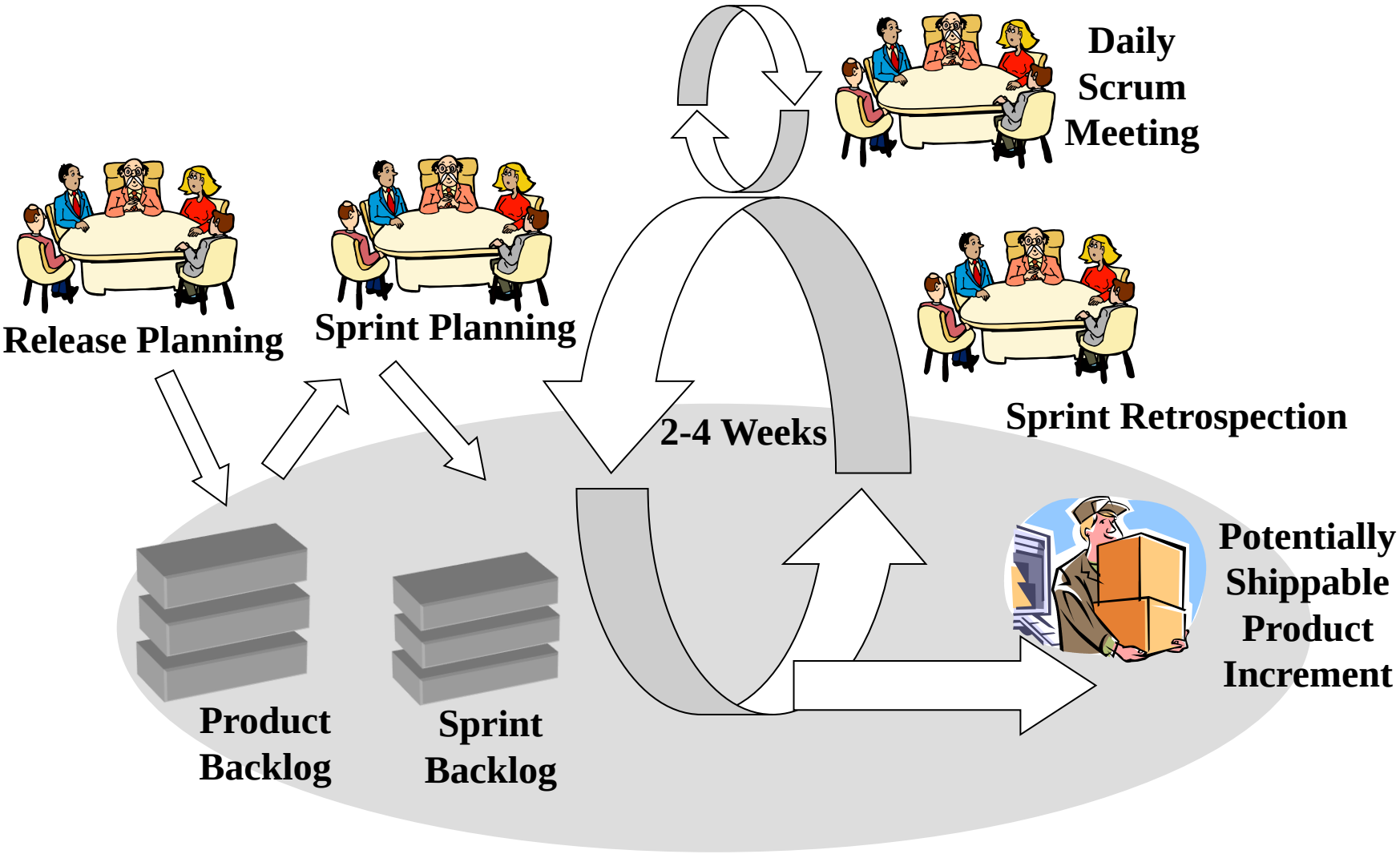
- Egy csapat, közös felelősség, három szerepkör

- PO: kapitány, kijelöli az irányt
- SM: motivál, koordinál, segít kikerülni az akadályokat
- DT: evezősök

Önmenedzselő csapat

- A DT csapat önmenedzselő
- Ők tudják a legjobban, hogyan tudnak hatékonyan dolgozni
- Ki, hogyan, min dolgozik
- Minden taszkért közösen felelnek
- SM, PO része a csapatnak

Scrum folyamat



A Scrum események

- Spring Planning (<= 8 óra)
- Sprint (<= 1 hó)
- Daily Scrum (<= 15 perc)
- Sprint Review (<= 4 óra)
- Sprint Retrospective (<= 3 óra)
- +1: Grooming

Projektindítás - vízió

- Első sprint előtti tervezés
- **Product Vision**: nagyon magas szintű vízió arról, hogy mi fog készülni
  - „I would have written you a shorter letter, but I didn't have the time." (Blaise Pascal)
  - **Üzleti célok: milyen értéket képvisel a megrendelőnek**
  - (Gyakorlati célok: fejlesztőként mit kell megcsinálnunk)
- NOT list: egyelőre nem kell vagy bizonytalan
  - Ez is a fókuszálást, priorizálást szolgálja

Projektindítás - DoD

- **Definition of Done**: fektessük le előre, hogy mikor tekintünk valamit elkészültnek?
  - Implementáltam
  - Lefordul
  - Tesztek futnak
  - Gitre pusholtam
  - Többiek átnézték
  - Többi komponenssel jól integrálódik
  - Master branchbe merge-elve lett
  - PO elfogadta
  - Stb.

Product Backlog

- Definiálni kell az elkészítendő **user story**kat
- Hagyományos specifikáció helyett ezeket használjuk
- **Prioritás** szerint csökkenő sorrendben, ez a backlog
- Minden sprint elején a tetejéről veszünk le annyit, amennyit a sprint alatt elkészítünk ➔ **Sprint Backlog**

User story jellemzői

- A követelmények megfogalmazásának eszköze, egy kicsi, tovább nem bontható feature
  - Független
  - Egyeztethető, módosítható
  - Értékes a felhasználónak vagy megrendelőnek
  - Kicsi
  - Becsülhető
  - Tesztelhető

User story tartalma

- A követelmények megfogalmazásának eszköze, egy kicsi, tovább nem bontható feature
- Adott formátuma van
  - **Cím**
  - **Leírás**
  - **Elfogadási kritériumok**
  - NOT lista
  - Nemtriviális tesztesetek
  - Alfeladatok
  - Story pontok
  - Becsült idő, elköltött idő

PO írja meg

Fejlesztőkre bízva

User story leírása

As a [ROLE], **I want** [GOAL] **so that** [REASON]

As an *administrator*, **I want** to be able to create *user accounts*, **so that** I can grant users access to the system

- Három fő dolog mindig benne van
  - **Role**: ki számára értékes
  - **Goal**: funkcionálisan mit tudjon a szoftver, gyakorlati cél, amit elérünk
  - **Reason**: miért képvisel ez értéket, hogyan kapcsolódik az üzleti célhoz

Elfogadási kritériumok

- Nem technológiai, hanem felhasználóközpontú
- De objektíven mérhető kritérium, pl.
  - A formon szerepel egy név mező
  - Ebben csak betűk szerepelhetnek, különben hibaüzenet jelenik meg
  - Van alatta egy mentés gomb
  - A gombon való kattintás elmenti az adatot az adatbázisban

NOT Lista

- Ami **jelenleg nem a scope része** a user storyban
  - Lehet későbbi user story része
  - Vagy még nem eldöntött, hogy szükséges-e
- Pl.:
  - A bemenetet még nem validáljuk
  - A jelszóemlékeztető funkciónak még nem kell működnie

Nemtriviális tesztesetek

- Emlékeztető, hogy milyen speciális eseteket kellene vizsgálni
- Például
  - Határidő nem lehet múltbeli dátum
  - Nem publikus képeket teszünk közzé egy publikus bejegyzésben

Alfeladatok

- A megrendelőnek lényegtelen
- Az önmenedzselő agilis csapat használja
  - Eldöntik, hogy lehet lépésekre bontani a feladatot,…
  - … és felosztják a lépéseket maguk közt
- Teljesen a csapatra van bízva
  - Lehet alfeladatokat kiosztani, de teljes storykat is
- Segíti a becslést

User Story példák

- The accounting software must run on a dedicated server
- As an administrator, I want the accounting software not to put extra load of critical systems
- Accounting data must be visualized fast
- As an end-user, I want the data of clients to be accessible in 1 second

A sprintek menete

- Ha a kezdeti tervezés kész, sprinteket kell terveznünk
- A sprintek menete
  1. Sprint **planning** (<= 8 óra)
  2. Tényleges **fejlesztés** napi munkával
  3. Spring **review** (<= 4 óra)
  4. **Retrospektív** (<= 3 óra)
- A sprint során a csapat nem kommunikál a megrendelővel, a Scrum Master tartja a kapcsolatot

Sprint tervezés

- Párórás meeting a sprint elején
- A sprint tipikusan 2-4 hét, erre tervezünk, ideális napokkal és hetekkel (napi 8 óra, heti 5 nap)

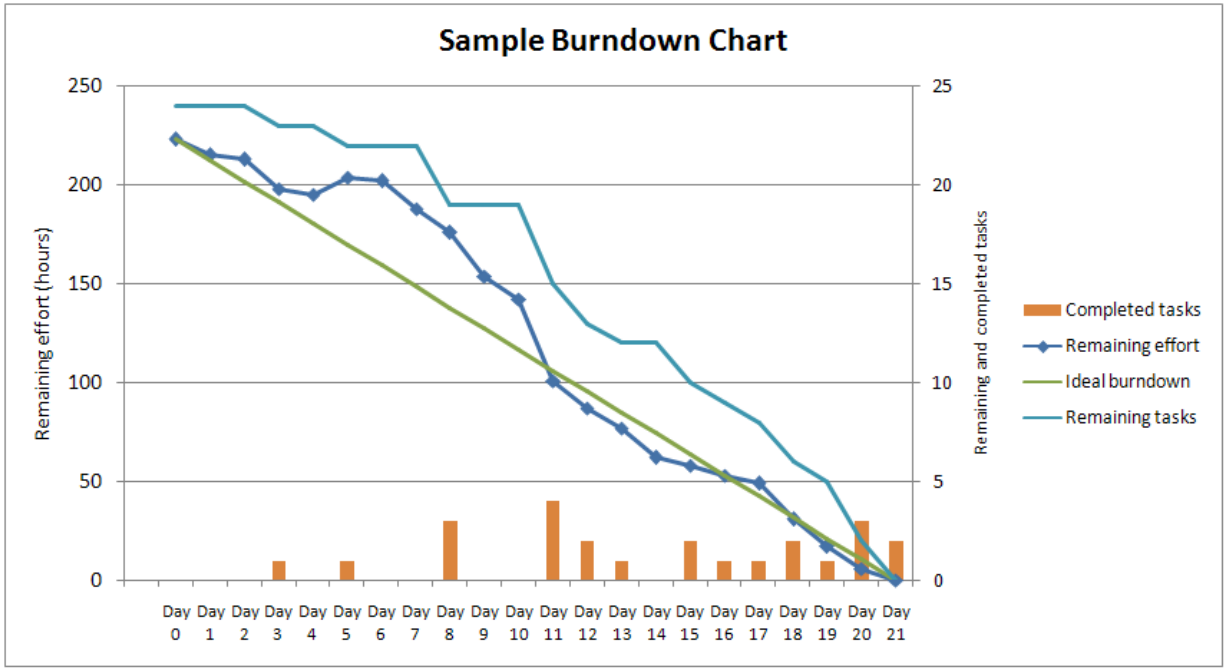
1. A user storykon megyünk prioritási sorban
2. Beazonosítjuk az alfeladatokat
3. Hozzárendelünk egy story pontot
   (Adunk egy időbecslést)
4. Amikor elég story-t „beáraztunk”, vállalást teszünk

Story Points

- A user storykat relatív nehézségi sorrend szerint rendezzük, és a közepes ér 5 pontot
- A közepes nehézségre adunk becslést
  - A többit ehhez relatívan becsüljük, pl. kétszer olyan nehéz, dupla idő
  - Korábbi tapasztalat segít a becsléseinket egyre pontosabbá tenni
  - Átlagos story pont / sprint teljesítmény kalkulálható
    - Velocity
  - Ez segít a hosszabb távú becslésben

Scrum artifacts

- **Product Backlog**: Termék teendőlistája prioritás szerint rendezve
- **Sprint Backlog**: Sprint teendőlistája
- **Burn down chart**: napi eredmény-kimutatás



Use Case vs. User Story

- Hasonlóság
  - Szerep/aktor, cél, elfogadási kritériumok
- Különbségek
  - Use case általában részletesebb
  - User story inkább beszélgetés indító
- “A story card is a promise for a conversation.”
- User story ~ feature – eredmény központú
  - „a marker for what is to be built, fine-grained enough to fit into modern iteration/sprint periods” (A. C.)
- Use case: hogyan reagál/mit csinál a rendszer
  - a contextual view of what is to be built

Daily Scrum

- Napi szinten is tervezünk a nap elején
- Mindenki képben van a teljes státusszal, a felmerülő problémákat rövid távú tervezéssel lehet kezelni
- Max. 15 perc, **ténylegesen felállva!**
  - Nem fotelben elterülve
  - Nem laptopba búvva
- Mindenki megválaszolja például ezt:
  - Mit végeztem el tegnap?
  - Mit fogok ma csinálni?
  - Van-e valami, ami akadályozza a munkámat?

A sprint terv módosulása

- Scope levágható
- Az időbeli hosszt sosem változtatjuk!
- Nincs félkész user story
  - Kettébontás
  - Rollback és a következő sprintre marad
- Sürgős user story bejöhet, PO dönti el
- Sprint csak akkor szakítható meg, ha a célja elavulttá vált

“Kész, kész” (done, done)

- Megtervezett
- Lekódolt
- Integrált
- Tesztelt
- A build lefut
- A termék installálódik
- Migrálódik
- A végfelhasználó ellenőrizte és elfogadta
- Megfixált

Demo

- Annak áttekintése, hogy mely munkák készültek el és melyek nem.
- Az elkészült munka bemutatása a terméktulajdonos és a fejlesztésben érdekeltek részére (demo).

Retrospektív

- Cél a munkafolyamataink folyamatos javítása
  - > Mit csináltunk jól, mit kell megtartani?
  - > Mit kéne másképp próbálnunk?
  - > Nem bűnbakok kereséséről szól!
- Kimenet: **végrehajtható akciók**
  - > Legalább egyet fel kell venni a következő sprint backlogjába
  - > Teszteljünk többet
    - – Például: csak getter, setter és konstruktor maradhat teszteletlen

Mit csinál pontosan a SM?

- „Szolgálva vezető”
- Facilitátor
- Coach
- Menedzser
- Mentor
- Oktató
- Problémamegoldó
- Változást segítő

https://www.scrum.org/resources/blog/8-stances-scrum-master

Kanban tábla

- Nem a Scrum része, külön technika, de szokták vele használni
- Fizikai tábla oszlopokkal, a sprint alatt a user story-kat és esetleg más taszkokat ezen vezetjük
- Az állapotokat a csapat dönti el, tipikusan a Definition of Done lépései alapján

Kanban

Agilis módszertanok kihívásai

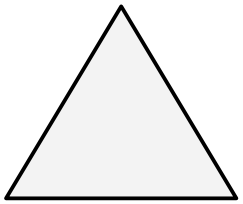
- Nehéz lehet érdeklődésben tartani a megrendelőt, aki benne van a folyamatban
- A csapat tagok nehezen alkalmazkodnak az intenzív folyamathoz, ami az agilis karakterisztikája
- Több stakeholder esetén kihívás a priorizálás
- Az egyszerűség fenntartása extra munka
- A szerződés nehézkes, ahogy minden iteratív fejlesztésnél

Agilis és terv alapú összevetése 1

- Agilis fejlesztés előtérbe helyezi az **egyén**eket, **interakció**t a folyamatokkal és eszközökkel szemben, **kevéssé formális**
- Agilis módszerek jobban működnek **kis-közepes** projektekre, ahol a **megrendelő bevonódik, nincs sok külső szabályozás, időkritikus helyzetek**
  - Skálázása kihívás
- Terv alapú megközelítés: nagy, **komplex projektek**
- **A pénzügyi megállapodás struktúrája meghatározó**
- Mind a kettőre szükség van

Agilis és terv alapú összevetése 2

- Mindegyik módszertannak megvan a maga erőssége, ki van valamire „hegyezve”!
  - Egyiket sem „könnyű” jól csinálni



- Vízesés:
  - Előre rögzített keretek tartása, tervezés, beavatkozás ☺
  - Nem biztos, hogy az készül el, amire szükség van ☹
- Agilis
  - Biztos, hogy olyan készül el, amire szükség van ☺
  - Előre nem tudjuk biztosan, hogy mikorra, mennyiért, és azt sem, hogy pontosan mi lesz az ☹

Mi határozza meg a módszertant?

- Az ügyféllel való megállapodás!
- A **megállapodás meghatározza, hogy mi a cél**, ehhez a célhoz érdemes választani a módszertant!
  - A **módszertan eszközöket ad** a kezünkbe a célok elérésére → nem tudjuk használni más célok elérésére (csavarhúzóval a szöget)
  - *Más módszertanok más eszközöket adnak más célok elérésére*

Az elvárások hatásai

- Ügyfél nem tudja előre pontosan megmondani, mit szeretne

--> Agilis -> „Change is welcome”

<- Kisebb egységek, visszajelzés alapján módosítjuk

<- Bármikor, könnyen lehessen módosítani!

<- A módosítás költsége kicsi legyen!

<- „Élő”, igaz dokumentáció

<- Gyorsan, magabiztosan módosítható kód

<- Jó unit tesztek -> Dependency injection, … …

Módszertan – architektúra kapcsolata

- A projekt módszertan támaszt elvárásokat: például csak akkor tudunk agilisan fejleszteni, ha tudunk megfelelő unit teszteket írni
  - Ez további architekturális és egyéb kritériumokat ad
  - Ez csak egy kritérium …
- **Egy módszertan alkalmazhatóságának vannak feltételei – legyünk ezekkel tisztában!**
- Hozhatunk bármilyen döntést: **tudatosan**!

Köszönöm a figyelmet!