Scrum és XP a frontvonalról
Hogyan használjuk a Scrum-ot

Írta:
Henrik Kniberg
Magyar fordítás:
Dezső Zoltán

Köszönetnyilvánítás

Ezen cikk első vázlata nagyjából egy hétvégét vett igénybe, de az aztán tényleg intenzív hétvége volt. 150%-os koncentrációs tényezővel! :)
Szeretném megköszönni feleségemnek, Sophiának és gyermekeimnek, Dave-nek és Jenny-nek, hogy
elviseltek azon a hétvégén és Sophia szüleinek, Eva-nak és Jörgennek, hogy segítettek felügyelni a gyerekekre. Köszönet illeti még kollégáimat a stockholmi Crisp-nél, illetve a scrumdevelopment yahoo levelezőlista tagjait, hogy lektorálták és ötleteikkel segítették a cikket.
Végül szeretnék köszönetet mondani Olvasóimnak. Az ő folyamatos visszajelzéseik rendkívül sokat segítettek. Kiváltképp örülök annak, hogy ez a cikk ennyi mindenkit ihletett az agilis fejlesztési módszertan kipróbálására.

Tartalomjegyzék
==========

A SZERZŐRŐL

Jeff Sutherland előszava

Egy csapatnak ismernie kell a Scrum alapjait. Hogyan készítsünk és becsüljünk Product Backlog-ot? Hogyan csináljunk belőle Sprint Backlogot? Hogyan használjuk a Burndown Chart-ot és hogyan számítsuk
ki a csapat sebességét? Henrik könyve bemutatja azokat az alapvető gyakorlatokat, amik segítségével
egy Scrum-ot próbálgató csapatból egy, a Scrum-ot teljes mértékben kihasználó csapat lehet.

A Scrum-ban való járatosság az utóbbi időben egyre fontosabbá válik olyan csapatok számára is, akik
befektetésben reménykednek. Agile trénerként én is egy olyan befektetői csoportot segítek, akik egyik követelménye, az agilis fejlesztési módszertanok beható ismerete. A csoport Senior Partner-e minden jelölttől megkérdezi, hogy ismeri-e a csapatai sebességét (velocity). Jelenleg a legtöbbjük nem vagy csak nehezen tud válaszolni. A jövőben a befektetésekhez elengedhetetlen lesz, hogy a csapatok ismerjék a fejlesztési sebességeiket.

Hogy ez miért olyan fontos? Ha a csapatok nem ismerik a sebességüket, akkor a Product Owner nem tud
előre tervezni és teljesíthető kiadási dátumokat megadni. Ha pedig nincs megbízható kiadási dátum, akkor a cég könnyen tönkre mehet és a befektetők elvesztik a pénzüket.

Ez egy olyan probléma, ami minden vállalkozást - kicsit és nagyot, régit és újat, befektetőkkel rendelkezőt és nem rendelkezőt - egyaránt érint. Nemrég, a Google Scrum használatáról tartott londoni beszédem során megkérdeztem a hallgatóságot (135 főt), hogy hányan használnak Scrum-ot.
30-an jelezték, hogy igen, ők Scrum-ot használnak. Ezután megkérdeztem, hogy hányan felelnek meg a
Nokia iteratív fejlesztési szabványának. Az iteratív fejlesztés az Agile Manifesto (agilis kiáltvány) egyik alappillére - használható szoftvert kiadni gyakran és már a kezdetektől. A Nokia több száz fejlesztői csapat több évnyi visszajelzése alapján megállapított néhány alapkövetelményt az iteratív fejlesztéshez:
* Minden iteráció legfeljebb hat hetes, rögzített időtartammal rendelkezik.
* Az iteráció végén a kód minőségteszten esett át és helyesen működik.

A 30 főből, akik azt mondták, hogy Scrum-ot használnak, csak 15 állította, hogy megfelelnek a Nokia iteratív fejlesztési követelményeinek. Tőlük ezután azt kérdeztem meg, hogy megfelelnek-e a Nokia Scrum követelményeinek:
* A Scrum csapatban van egy kirendelt, a csapat tagjai által elismert Product Owner.
* A Product Owner rendelkezik Product Backlog-gal, amiben a csapat becslései is szerepelnek.
* A csapat rendelkezik Burndown Chart-tal és ismeri a sebességét.
* A csapaton kívüli személyek nem zavarják meg a csapatot egy sprint közben.

A 30 Scrum-ot használó személy közül mindössze 3 felelt meg a Nokia Scrum tesztjének. Ezek azok a

csapatok, akikbe a beruházói köröm ismerősei a jövőben be fognak fektetni.

Henrik könyvének igazi értéke, hogy ha az általa ismertetett gyakorlatokat követi akkor rendelkezik majd Product Backlog-gal, becslést készít majd a Backlog elemeihez, rendelkezni fog Burndown Chart-
tal és ismerni fogja a sebességét. Úgy általában olajozottan fog működni a Scrum a cégében. Meg fog
felelni a Nokia követelményeinek és méltán reménykedhet majd külső befektetésben. Amennyiben cége
épp csak megalakult, kezdeti tőkét kaphat egy befektetői csoporttól. Ön lehet a jövő sikeres szoftverfejlesztője és a következő generáció sikertermékének szülőatyja.

Jeff Sutherland,
Ph.D., a Scrum társszerzője

Mike Cohn előszava
==========
A Scrum is és az Extreme Programming (XP) is azt várják el a csapatoktól, hogy egy iteráció végére előálljanak egy kézzel fogható, működő termékkel. Ezek az iterációk szándékosan rövid, rögzített hosszal rendelkeznek. A rövid idő alatt működő kód elkészítésére való összpontosítás azt jelenti, hogy egy Scrum vagy XP csapatban nincs idő elmélkedni. Nem lényeges hogy a legszebb UML
modellt készítsük el, a legtökéletesebb követelmény-elemzést írjuk meg, vagy hogy minden lehetséges
jövőbeni módosításra kész kódot írjunk. Ehelyett egy Scrum vagy XP csapatban a hangsúly azon van,
hogy a termék elkészüljön. Ezek a csapatok elfogadják, hogy hibák előfordulnak majd, de azt is felismerik, hogy ezen hibák feltárásának legjobb módja hogy az ember a elméleti eszmefuttatások helyett
felgyűri az ingujjat és nekiáll ténylegesen elkészíteni a szoftvert.

Pont ez a pragmatikus hozzáállás az, ami miatt ez a könyv többet nyújt a megszokottaknál.
Hogy Henrik is érti ezt, az már a kezdetektől teljesen világos. Nem ír regényeket arról, hogy mi is a Scrum (helyette az érdeklődő számára inkább néhány weboldalt ajánl). Inkább a közepébe vág és elmagyarázza, hogy hogyan használja a csapata a Product Backlogot. Ebből a kezdőpontból bontja ki
aztán a többi, jól működő agilis projekthez elengedhetetlen elemet. Nincsenek elmélkedések. Nincsenek hivatkozások. Nincsenek lábjegyzetek. És nincs is rájuk szükség: Henrik könyve nem egy filozófiai eszmefuttatás arról, hogy mi is az a Scrum, vagy hogy miért is próbáljon ki ezt vagy azt. Ehelyett egy jól működő Scrum csapat működésébe kapunk betekintést.

Ezért kapta a könyv találóan a "Hogyan használjuk a Scrum-ot" alcímet. Lehet, hogy Ön nem így használja a Scrum-ot, de Henrik csapata igen. Persze megfordulhat a fejében, hogy miért jó az, ha tudja, hogy egy másik csapat hogyan használja a Scrum-ot. A válasz az, hogy egy Scrum-ot használó

(és főképp egy hatékonyan használó) csapat történeteiből magunk is tanulhatunk. Nincs és nem is lesz
egy "Scrum Előírások" lista, mert a csapatok és projektek kontextusa sokkal fontosabb minden más megfontolásnál. Előírások helyett a gyakorlatok mellett arra is szükség van, hogy ismerjük azt a kontextust, amelyben sikeresen használhatóak. Ha elég történetet olvas sikeres csapatokról és azok gyakorlatairól, készen lesz bármilyen akadály elhárítására, ami a Scrum és XP használata során az útjába akad.

Henrik rengeteg jó tanáccsal szolgál majd és megadja a szükséges kontextust, hogy Ön még sikeresebben
használhassa a Scrum-ot és XP-t a saját projektjei frontvonalain.

Mike Cohn
Az "Agilis Becslés és Tervezés" és "Tervezés és Felhasználói Történetek a Gyakorlatban" című könyvek
szerzője.

Előszó – Hé, a scrum működött!
==========
A Scrum működött. Legalábbis nekünk (ami alatt a jelenlegi
stockholmi megbízót értem, akit nem fogok megnevezni). Remélem, hogy Önnek is működni fog. Talán ezt az írást is hasznos iránymutatónak találja majd.

Ez az első alkalom, hogy egy fejlesztői metodológia (bocsánat, Ken, keretrendszer) elsőre működött. Csont nélkül. Mindannyian elégedettek vagyunk vele – a fejlesztők, a teszterek,
a menedzserek. Átsegített néhány nehéz helyzeten és közben segített megőriznünk a lendületünk és koncentrációnk a nehéz piaci viszonyok és sorozatos leépítések között is.

Nem szabadna azt mondanom, hogy ez az eredmény meglepett, de mégis ezt kell mondanom. Néhány
könyv elolvasása után a Scrum jónak tűnt, de talán túl jónak is, hogy igaz legyen (ismerjük a mondást “ha valami túl jónak tűnik, hogy igaz legyen...”), így talán érhető a kezdeti kétkedésem. Azonban egy évnyi Scrum után meglehetősen elégedett lettem (csakúgy, mint a csapat többi tagja); olyannyira, hogy ezentúl hacsak nincs egy jó érv ellene, alapértelmezés szerint minden új projektben Scrum-ot fogok használni.

---

1 Bevezető
==========
Hamarosan bevezetésre kerül a Scrum a vállalatában, vagy már néhány hónapja
Scrum-ot használ. Az alapokat ismeri, a könyveket kiolvasta, az is elképzelhető, hogy Scrum Master minősítést szerzett. Gratulálok!

És mégis valahol ott bujkál egy kis zavarodottság.

Ken Schwaber szavaival élve, a Scrum nem egy metodológia, hanem egy keretrendszer, ami abban nyilvánul meg, hogy a Scrum nem mondja meg, hogy milyen helyzetben pontosan mit is tegyen. A fenébe!

A jó hír az, hogy én messzemenő részletességgel be fogom mutatni, hogy én hogyan használom a Scrum-ot. A rossz hír az, hogy ez csak azt fogja leírni, ahogy én használom a Scrum-ot, ami nem jelenti azt, hogy Önnek is így kell használnia. Ami azt illeti, én magam is elképzelhető, hogy egy más szituációban másképp használnám.

A Scrum erőssége, és egyben a gyengesége is, hogy megköveteli, hogy mindig az adott helyzethez igazítsák.

A Scrum-hoz való hozzáállásom jelenleg egy évnyi kísérletezés eredménye egy nagyjából 40 fős fejlesztői csapattal. A cég meglehetősen nehéz helyzetben volt, rengeteg túlórával, óriási minőségi problémákkal, állandó tűzoltással, kitolt határidőkkel, stb. Döntés született ugyan a Scrum bevezetéséről, de ez a gyakorlatba már nem lett teljesen átültetve: és ez lett az én feladatom. Ezekben az időkben a fejlesztői csapat nagy részének a “Scrum” csak egy fura “buzzword” volt, amit néha hall az ember, ám ami a napi gyakorlatra abszolút nincs kihatással.

Egy év alatt, bevezettük a Scrum-ot a cég teljes hierarchiájában, kipróbáltunk különböző csapat méreteket (3-12 fő), különböző sprint hosszokat (2-6 hét), különböző “kész” definíciókat, különböző backlog formátumokat (Excel, Jira, cetlik), különböző teszt stratégiákat, különböző demó módokat, különböző csapat-szinkronizációs módokat, stb... Kísérleteztünk XP módszerekkel – különböző folyamatos integrációs módokkal, pár-programozással, teszt-alapú programozással, illetve ezek a Scrum-ba történő integrációjával.

Ez egy állandó tanulási folyamat, szóval a történet persze itt nem ér véget. Meggyőződésem, hogy a cég továbbra is fejlődni fog (legalábbis ha betartják a sprint visszatekintőket (retrospective)) és új felfedezéseket fognak tenni arra vonatkozóan, hogy az ő helyzetükben mi a legjobb módszer a Scrum használatára.

Jogi nyilatkozat
==========
Ez az írás nem “Az Egy Helyes Mód” a Scrum használatára, hanem csak egy mód a sok lehetséges közül, ami egy évnyi folyamatos finomhangolás eredménye. De persze az is elképzelhető,
hogy teljesen rossz úton járunk.

Az írás teljes egészében a személyes véleményemet tartalmazza és nem feltétlenül tükrözi a Crisp vagy bármely kliensem hivatalos álláspontját. Éppen ezért szándékosan nem fogok konkrét személyekre vagy termékekre hivatkozni.

Miért írtam ezt a cikket
==========
Kezdetben, amikor a Scrum-ot próbáltam megérteni, elolvastam a

Scrum-ról szóló összes könyvet és az összes internetes cikket vagy fórumot, letettem Ken Schwaber vizsgáját, elárasztottam őt kérdésekkel és rengeteget beszéltem a munkatársakkal a témáról. Mindeközben az egyik leghasznosabb információ-forrásomnak mégis a valós történetek bizonyultak. Ezen “háborús történetek” végére az Alapelvekből és Előírásokból-okból... nos... “Jó, De **Valójában** Hogyan Használod” lesz. Abban is segítségemre voltak, hogy felismerjem (és néha el
is kerüljem) a Scrum tipikus csapdáit.

Ez tehát az én lehetőségem, hogy valamit visszaadjak a közösségnek. Ez az én történetem a frontról.

Remélem a cikk apropóján mások is felbátorodnak majd és megosztják velem, hogy hasonló helyzetekben
ők hogyan cselekedtek.

De mi is ez a Scrum?
==========
Hoppá, elnézést. Teljesen járatlan a Scrum-ban vagy az XP-ben? Ebben az esetben az alábbi néhány
linket ajánlanám figyelmébe:
* http://agilemanifesto.org/
* http://www.mountaingoatsoftware.com/scrum
* http://www.xprogramming.com/xpmag/whatisxp.htm

A türelmetlenek nyugodtan olvassanak tovább. A zsargon nagy részét menet közben meg fogom magyarázni, szóval teljesen kezdőként is érdekesnek találhatja ezt az írást.

---

2 Hogyan használjuk a Product Backlogot
==========
A Product Backlog a Scrum motorja. Minden azzal kezdődik. A Product Backlog egy prioritás alapján rendezett követelmény-, sztori-, feature- vagy akármi- lista. Olyan dolgok, amiket a kliens szeretne, a kliens nyelvezetével leírva.

Mi ezeket Sztoriknak hívjuk, vagy néha egyszerűen csak Backlog Bejegyzés-nek.

A sztorijaink a következő adatokkal rendelkeznek:
* ID – egy egyedi azonosító, jobbára csak egy automatikusan léptetett szám.
Ez azért szükséges, hogy ne veszítsük el véletlenül a sztorikat, amikor például átnevezzük őket.
* Név – rövid, kifejező Sztori név. Például “A saját tranzakció-előzmények
megjelenítése” Elég világos kell, legyen, hogy a fejlesztők és a Product Owner nagyjából
értsék miről van szó és hogy elkülönüljön a többi sztoritól. Általában 2-10 szó.
* Fontosság – a Product Owner által meghatározott fontosság számszerű értéke. Például 10, vagy 150.
  Minél nagyobb a szám, annál fontosabb a Sztori.
  o Amikor csak lehet, próbálom kerülni a "prioritás" kifejezést, mivel tipikusan 1 a legmagasabb

prioritás, ami akkor tud gondot okozni, amikor egy _még_ fontosabb feladat érkezik. Milyen prioritást adjunk annak? 0-t? -1-t?
* Kezdeti becslés - a sztori - más feladatokhoz viszonyított - elvégzendő munkamennyiségének becslése.
Egysége a sztori pont, ami nagyjából egy ideális ember-napnak felel meg.
  o Kérdezze meg a csapatot, hogy "ha a sztorihoz szükséges ideális számú (se túl kevés, se túl sok, általában 2) emberrel bezárkóznak egy étellel jól ellátott szobába, ahol minden külső zavaró tényezőtől mentesen dolgozhatnak, hány nap múlva tudnak előállni a kész, bemutatható, tesztelt, kiadható implementációval?". Ha a válasz az, hogy "hárman bezárva egy szobába 4 nap", akkor a kezdeti becslés 12.
  o Nem az a lényeg, hogy az abszolút becslések helyesek legyenek (vagyis hogy egy 2 pontos Sztori
    valóban 2 napot vegyen igénybe), hanem hogy a relatív becslések helyesek legyenek (azaz hogy egy 2 pontos sztori nagyjából fele annyi időt vegyen igénybe, mint egy 4 pontos)
* Demó módja - egy áttekintő leírás arról, hogy ezt a sztorit hogyan kell bemutatni a sprint demó során. Ez igazából egy leegyszerűsített teszt előírás. "Ezeket a lépéseket végrehajtva, ez kell, legyen az eredmény".
  o Ha TDD-t (teszt-vezérelt fejlesztés) használ, ez a leírás szolgálhat az elfogadási teszt, egyfajta pszeudokódban megadott, alapjául.
* Megjegyzések - minden más információ, pontosítás, más forrásokra való hivatkozás. Legyen minél rövidebb

<figure>

Kipróbáltunk más adatokat is, de végül a fenti hat mező volt, amit ténylegesen használtunk.

Általában egy megosztott Excel táblázatban tároljuk a backlog bejegyzéseket. Hivatalosan a Product Owner felelős a dokumentumért, de persze nem zárjuk ki a többi felhasználót sem, mivel sok esetben
egy fejlesztő pontosíthat egy Sztorit, vagy esetleg megváltoztathat egy becslést.

Hasonló okokból nem tároljuk ezt a dokumentumot a verziókövető rendszerben, ehelyett egy hálózati meghajtóra tesszük. Ez bizonyult a legegyszerűbb módszernek arra, hogy zárolások vagy konfliktusok
nélkül több felhasználónak egyidőben hozzáférést biztosítsunk.

Minden más dokumentumot azonban verziókövető rendszerben tárolunk.

További sztori mezők
==========
Néha további mezőket is használunk a backlogban, többnyire, hogy megkönnyítsük a Product Owner dolgát
amikor a Sztorik fontosságát határozza meg.
* Kategória – a Sztori hozzávetőleges fajtája, például "optimalizáció" vagy "backend". Ezt használva a Product Owner könnyen kiválaszthatja például az összes optimalizációs feladatot és mondjuk azok prioritását alacsonyra állíthatja.
* Komponens - általában jelölőnégyzetként jelenik meg az Excel dokumentumban, például

"adatbázis,
szerver, kliens". A Product Owner ezek segítségével könnyen azonosíthatja azokat komponenseket, amelyek
az implementációban szerepet játszanak. Ez akkor lehet hasznos, amikor több Scrum csapat van (például egy a "backend"-hez, egy a klienshez), mivel leegyszerűsítheti a sztorik csapatokhoz rendelését.
* Megbízó - a Product Owner dokumentálhatja, hogy melyik külső- vagy belső megbízó kérte a sztori implementálását, hogy később könnyen visszajelezhessen ennek a személynek.
* Hibakövető ID - amennyiben külön hibakövető rendszert is használ (mi például Jira-t), nagy segítséget jelent, ha közvetlenül összekapcsolhatja a sztorit a hozzá tartozó hibajelentésekkel.

Hogyan vezetjük a Product Backlogot az üzleti szinten

Ha a Product Owner technikai beállítottságú, előfordulhat, hogy olyan sztorik kerülnek a Backlogba, mint
például "Indexek hozzáadása az Events táblához". De igazából miért van erre szüksége? A valódi ok valószínűleg valami olyasmi, hogy "keresések felgyorsítása az adminisztrációs felületen".

Előfordulhat az is, hogy a keresések nem is az indexek hiánya miatt voltak lassúak, hanem valami teljesen más ok miatt. Általában a fejlesztő csapat sokkal jobb pozícióban van, hogy a probléma forrását meghatározza és elhárítsa, ezért biztosabb, ha a Product Owner inkább az üzleti oldalra koncentrál.

Amikor ilyen technikai beállítottságú sztorikkal találkozom, a Product Owner-t addig nyúzom "de miért" kérdésekkel amíg meg nem találjuk a valódi indokot, és a sztorit át nem fogalmazzuk ("keresés felgyorsítása az adminisztrációs felületen"). Az eredeti - technikai jellegű - megfogalmazást pedig egy megjegyzésként tartjuk meg ("Indexek hozzáadása az events táblához talán megoldja a problémát").

3 Hogyan készülünk fel a sprint planning meetingre
==========
OK, egyre közeleg a sprint planning napja. A következő leckét mi is újra és újra meg kell, tanuljuk:

Jótanács: A Sprint Planning Meeting előtt mindenképpen tegyük rendbe a Product Backlogot.

Hogy mindez mit jelent? Hogy minden sztori legyen teljesen definiálva? Hogy minden becslés legyen pontos? Hogy minden prioritás legyen lefixálva? Nem, nem és nem! Mindössze a következőket jelenti:
* A Product Backlog létezik! (képzelje csak el)
* Egy Product Backlog és egy Product Owner van (természetesen termékenként értendő).
* Minden lényegesebb elemhez hozzá van rendelve egy egyedi fontossági érték.
  o Ami azt illeti, nem probléma, ha a nem fontos elemekhez ugyanazt az értéket rendeli, mivel azok valószínűleg úgysem fognak előkerülni a Sprint Planning Meeting során.
  o Minden sztorihoz, amiről a Product Owner úgy véli, valós eséllyel rendelkezik, hogy bekerüljön a következő sprintbe, egyedi érték van rendelve.
  o A fontossági érték mindössze arra szolgál, hogy az elemeket fontossági sorrendbe állítsuk, azaz,

ha az A Sztori fontossága 20 és B Sztori fontossága 100, ez csak annyit jelent, hogy B fontosabb, mint A, nem azt, hogy B ötször olyan fontos, mint A. Ha B fontossága 21 lenne, az is pontosan ugyanezt jelentené.
o Hasznos lehet a fontossági értékek között hézagokat hagyni, ha esetleg egy C feladat jelenne meg, ami fontosabb, mint A, de kevésbé fontos, mint B. Persze használhatnánk 20,5-t is C-re, de az ilyen értékek elég hamar kicsúsznak az ember irányítása alól, úgyhogy jobb hézagokat hagyni.
* A Product Owner érti az összes sztorit
(általában ő írja a sztorikat, de néha mások is hozzáadnak kéréseket, amiket a Product Owner-nek kell fontossági sorrendbe állítania). Az nem szükséges, hogy pontosan tudja, hogy hogyan lesz implementálva, de legalább azt értenie kell, hogy miért van a sztori a listán.

Megjegyzés: Más személyek is adhatnak sztorikat a Product Backlog-hoz, de csak és kizárólag a Product Owner rendelhet fontossági értékeket a sztorikhoz. Hasonlóan, csak és kizárólag a csapat adhat meg időbecsléseket.

További módszerek, amiket kipróbáltunk vagy fontolóba vettünk:
* Jira-t használni a Product Backlog tárolásához. Ezt a legtöbb Product Owner nehézkesnek tartotta, mivel túl sok kattintást igényelt. Az Excel lapot könnyen tudják szerkeszteni, könnyű színkódokat megadni, cellákat átrendezni, új oszlopokat hozzáadni, megjegyzéseket írni, importálni és exportálni, stb...
* Agilis fejlesztőeszközöket, mint például VersionOne, ScrumWorks, XPlanner, használni. Még nem jutottunk el oda, hogy ezeket behatóbban tanulmányozzuk, de előbb-utóbb ki fogjuk őket próbálni.

4 Hogyan tervezzük a sprinteket
==========
A Sprint Planning a Scrum egy fontos - ha nem a legfontosabb - eleme (legalábbis véleményem szerint) Egy rosszul végrehajtott Sprint Planning Meeting kihatással lehet a teljes Sprint-re.

A Sprint Planning Meeting célja, hogy a csapat számára elég információval szolgáljon ahhoz, hogy zavartalanul dolgozhassanak egy pár hétig illetve hogy elég biztonságot nyújtson a Product Owner-nek,
hogy ezt engedje is.

Na jó, ez elég általános volt. Konkrétan a következő dolgokról születik döntés:
* A sprint célja
* A csapat tagjainak listája (és a rendelkezésre állásuk szintje, ha nem 100%)
* Egy Sprint Backlog (= a sprint alatt elvégzendő sztorik listája)
* A sprint demó dátuma
* A Napi Scrum helye és időpontja

Miért kell a Product Ownernek jelen lennie

Néhány esetben a Product Owner nem szeretne hosszú órákat vesztegetni a csapattal a Sprint Planning
Meetingen. "Srácok, én már leírtam, mit akarok. Nincs időm részt venni a meetingen". Ez egy elég súlyos probléma.

Az egész csapat és a Product Owner azért kell, hogy jelen legyen a meetingen, mert minden sztoriban
három egymástól erősen függő változó rejtőzik.

<figure>

A kiterjedést és a fontosságot a Product Owner határozza meg. A becsült időt a csapat. A Sprint Planning során a szemtől-szembeni megbeszéléssel folyamatosan ezeket a változókat hangoljuk.

A Product Owber általában a sprint céljának összefoglalásával és a legfontosabb sztorik listájával nyitja a meetinget. Ezután a csapat fontossági sorrendben végigmegy sztorik listáján és megbecsüli a szükséges időket. A becslések közben azonban kérdések merülhetnek fel a sztori kiterjedésével kapcsolatban - "ez a 'felhasználó törlése' magában foglalja azt, hogy megszakítjuk a felhasználó összes várakozó tranzakcióját?" A válaszok akár meg is lephetik a csapatot és persze jelentősen befolyásolhatják a becsléseket.

Egyes esetekben a sztorira adott becslés eltérhet a Product Owner elképzeléseitől. Ennek hatására megváltoztathatja a sztori fontosságát, vagy esetleg a sztori kiterjedését, ami aztán újabb becslést igényel a csapattól.

Ez a közvetlen együttműködés alapvető fontosságú a Scrum-ban és minden agilis fejlesztési módszertanban.

Mi történjen, ha a Product Owner továbbra sem szeretné az idejét a Sprint Planning Meetingen tölteni? Általában a következő taktikák egyikét szoktam alkalmazni ebben a sorrendben:
* Elmagyarázom a Product Owner-nek, hogy miért elengedhetetlen a jelenléte annak reményében, hogy
  megváltoztatja a véleményét.
* Megkérem a csapatot, hogy egy önkéntest jelöljön ki a Product Owner közvetítőjeként. A Product Owner-nek a következőt mondom: "Mivel nem tudsz jelen lenni a meetingen, Jeff fog téged képviselni és fel lesz hatalmazva hogy megváltoztassa a prioritásokat és a sztorik kiterjedését. Amennyire csak lehet próbálj megegyezni vele még a meeting előtt. Ha nem értesz egyet azzal, hogy
  Jeff képviseljen téged, akkor jelölj ki valaki mást, aki a meeting teljes ideje alatt jelen tud lenni."
* Megpróbálom meggyőzni a vezetést, hogy rendeljenek ki egy új Product Owner-t.
* Elhalasztom a sprintet amíg a Product Owner nem talál elég időt hogy jelen legyen és ezalatt nem kötelezem el magam semmilyen kiadás mellett. A csapatot hagyom azokon a feladatokon dolgozni, amiket
  épp a legfontosabbnak gondolnak.

Miért nem engedünk a minőségből

A fenti háromszögben szándékosan nem említettem egy negyedik változót: a minőséget.

Kétféle minőséget szoktam megkülönböztetni: belső- és külső minőséget.

* Külső minőségnek hívom a felhasználók által érzékelhető minőséget. A lassú vagy nem intuitív felhasználói interfész a rossz külső minőség egy példája.
* Belső minőségnek tekintem az olyan tényezőket, amelyek ugyan nem láthatóak a felhasználó számára,
de jelentősen befolyásolják a rendszer karbantarthatóságát. Ide tartoznak az olyan dolgok, mint az architektúra konzisztenciája, teszt lefedettség, a kód olvashatósága vagy a refaktoring.

Általánosságban egy kiemelkedő belső minőséggel rendelkező rendszer önmagában még nem jelent jó külső
minőséget, azonban egy rossz belső minőséggel rendelkező rendszer nagyon ritkán rendelkezik jó külső
minőséggel, mivel rendkívül nehéz rothadó alapozásra jó dolgot építeni.

A külső minőség a kiterjedés (scope) része. Üzleti szempontból néha teljesen elfogadható ha a felhasználói felület lassú vagy nehéz használni, amennyiben egy későbbi kiadásban ezeket javítjuk. A Product Owner dolga annak eldöntése, hogy ez tényleg megéri-e.

A belső minőség azonban sosem vita tárgya. A csapat feladata minden körülmények között megőrizni a
rendszer jó minőségét és ebből nem engedünk soha, semmi szín alatt.

(Na jó, majdnem soha)

De hogyan is különböztetjük meg a belső és külső minőséget érintő kérdéseket?

Tegyük fel, hogy a Product Owner azt mondja: "Rendben, értem én, hogy 6 sztori pontra becsültétek, de biztos vagyok benne, hogy ha akarjátok, fele idő alatt össze tudtok hozni egy gyors fixet."

Aha! A belső minőséget változónak próbálja tekinteni. Ezt onnan tudom, hogy le akarja csökkenteni a becslést anélkül, hogy azért fizetne a csökkentett kiterjedéssel. A "gyors fix" szó láttán gondolatban meg kell szólalnia egy vészcsengőnek...

És hogy miért is nem hagyjuk ezt?

Az a tapasztalatom, hogy a belső minőség feláldozása szinte mindig nagyon nagyon rossz ötlet. Ami időt nyerünk vele, azt kamatostól elveszítjük mind rövid-, mind pedig hosszú távon. Ha egy kódbázis minősége romlásnak indul, akkor nagyon nehéz azt utólag helyrehozni.

Ehelyett a vitát inkább a kiterjedés irányába próbálom terelni. "Mivel fontos hogy ezt a sztorit minél előbb kiadhassuk, lecsökkenthetjük-e a méretét, hogy gyorsabban implementálhassuk? Például a hibakezelést leegyszerűsítjük és később egy másik sztoriban dolgozunk a bonyolultabb hibakezelésen?
Vagy módosíthatjuk a többi feladat fontosságát, hogy inkább erre koncentrálhassunk?"

Ha a Product Owner megtanulja, hogy a belső minőséget illetően nincs helye kompromisszumoknak, elég
hamar rájön, hogyan érje el a céljait a többi paraméter hangolásával.

Véget nem érő Sprint Planning Meeting-ek...

A legnehezebb dolog a Spring Planning Meetingekben, hogy:
1) Mindenki azt hiszi, hogy nem vesznek sok időt igénybe
2) ... de igen.

A Scrum-ban minden időkorlátos. Szeretem ezt az egyszerű, konzisztens szabályt, amit mindig próbálunk is
betartani.

Mit tegyünk, ha egy ilyen időkorlátos Sprint Planning Meeting az időkorlát vége felé közeledik és még nincs Sprint cél vagy nincs készen a Sprint Backlog? Egyszerűen hagyjuk félbe? Vagy toljuk ki egy
órával? Vagy esetleg hagyjuk félbe, de folytassuk másnap?

Ez a kérdés lépten-nyomon felmerül, főleg új csapatok esetében. Őszintén szólva nem tudom, mi a legjobb módszer, de mi általában brutálisan félbeszakítjuk még ha ez a sprint rovására megy is. Egész pontosan azt mondom a csapatnak és a Product Owner-ner, hogy "a meetingre kijelölt idő 10 perc
múlva lejár és még nincs egy kézzel fogható sprint tervünk. Fussunk neki a sprintnek ezzel, vagy hívjunk össze egy új 4-órás meetinget holnap reggel 8-ra?". Nem nehéz kitalálni, hogy általában mi a válasz... :)

Próbáltam kitolni a meetinget, de ez szinte soha sem vezetett sehová, mert az emberek már fáradtak.
Ha nem voltunk képesek összehozni egy normális sprint tervet 2-8 óra alatt (vagy amilyen hosszúra szabtuk időkeretet) akkor valószínűleg egy további óra sem fog sokat számítani. A következő - amúgy teljesen elfogadható - opció, hogy összehívunk egy új meetinget a következő napra. Csakhogy
általában az emberek türelmetlenek és inkább nekikezdenének már a sprintnek ahelyett, hogy még egy
napot töltsenek sprint tervezéssel.

Úgyhogy félbevágom és igen, a sprint issza meg a levét. A jó oldala a dolognak viszont az, hogy a csapat megtanult egy fontos leckét és a következő sprint planning meeting sokkal gördülékenyebben fog menni. Ráadásul az emberek kevésbé fognak ellenkezni, ha olyan időkeretet szabnék ki, amit egyébként túl hosszúnak tartanának.

Fontos, hogy megtanulja betartani az időkereteket és hogy azokat realisztikusan válassza meg. Ez vonatkozik mind a meetingekre, mind pedig a sprint hosszára.

Sprint Planning Meeting napirend

Egy előzetes menetterv készítése jelentősen lecsökkenti annak az esélyét, hogy a meeting kifut a megszabott időkeretből.

Íme a tipikus menetrendünk.
Sprint Planning Meeting: 13:00- 17:00 (10 perc szünet minden órában)

* 13:00 - 13:30. A Product Owner ismerteti a sprint célját és összefoglalja a Product Backlogot. Kijelölésre kerül a demó helye és ideje.
* 13:30 - 15:00. A csapat elkészíti a becsléseket és felbontja a nagyobb feladatokat. A Product Owner frissíti a sztorik fontosságát. A feladatok pontosításra kerülnek. Minden fontos sztorihoz kitöltésre kerül a "Demó módja" mező.
* 15:00 - 16:00. A csapat kiválasztja a következő sprintbe kerülő elemeket. Ellenőrzik hogy a sztorik valóban belefférnek-e a sprintbe.
* 16:00 - 17:00. Kijelölésre kerül a Napi Scrum helye és ideje (amennyiben eltér az előző sprinttől). A sztorik lebontása feladatokra.

Ehhez a menetrendhez nem ragaszkodunk vasszigorral. A Scrum Master a körülményeknek megfelelően
meghosszabbíthatja vagy lerövidítheti az egyes részeket.

A sprint hosszának megválasztása

A Sprint Planning Meeting egyik eleme a sprint demó idejének megválasztása, ami valójában a sprint időtartamának megválasztásával egyenértékű.

Mit tekinthetünk jó sprint hossznak?

Jó, ha a sprint rövid, mivel ez segíti a vállalat "agilitását", azaz, hogy gyakran képes irányt váltani.
Rövid sprint = gyors visszajelzés = több kiadás = gyorsabb visszajelzés a felhasználóktól = kevesebb idő pazarlása vakvágányokra = gyorsabb adaptáció, stb...

De ugyanakkor a hosszabb sprintek is jók, mivel időt hagynak a csapatnak hogy felgyorsuljanak és több teret ad nekik a problémák megoldására, hogy végül a sprint célját teljesítsék; kevesebb időt veszítenek továbbá a Sprint Planning Meetingeken vagy demókon.

A Product Owner-ek általában rövid-, míg a fejlesztők hosszabb sprinteket szeretnek, így a sprint hossza valójában kompromisszum kérdése. Elég sokat kísérleteztünk ezzel és végül a három hetes sprintet találtuk a legmegfelelőbbnek. A legtöbb (de nem az összes) csapat 3 hetes sprinteket használ, ami elég rövid, hogy biztosítsa a vállalat flexibilitását, de elég hosszú, hogy a csapat megfelelően koncentrálhasson és megoldhassa a sprint során esetlegesen felmerülő problémákat.

Azt javaslom, hogy kezdetben kísérletezzen különböző sprint hosszakkal. Nincs szükség mélyreható vizsgálatra, csak válasszon egy jónak tűnő hosszot és próbálja ki egy pár sprint erejéig.

Ha viszont megtalálta a legkedvezőbb sprint hosszat, akkor már ne nagyon változtasson rajta. Mi egy pár hónap kísérletezgetés után arra jutottunk, hogy a három hét jól működik nálunk, ezért három hetes sprinteket használunk és kész.

Néha a sprint túl hosszúnak fog tűnni és néha túl rövidnek. De az azonos hosszhoz ragaszkodás egyfajta céges szívveréssé válik, amibe mindenki beleszokik. Nincsenek viták a kiadások időpontját illetően, mivel mindenki tudja, hogy a kiadás három hetente történik. És kész.

A sprint céljának megválasztása

Szinte mindig ugyanaz történik: a sprint planning meeting során felteszem a kérdést, hogy "tehát mi is a sprint célja?" és mindenki csak bambán néz rá, miközben a Product Owner ráncolja a homlokát.

Valami megmagyarázhatatlan okból kifolyólag túl nehéz eldönteni a sprint célját, de arra kellett rájönnöm, hogy megéri. Egy mondvacsinált cél is jobb, mint egy cél nélküli sprint. Az is megfelelő cél, hogy "több pénzt csinálni" vagy "a legfontosabb három sztorit végigcsinálni" vagy "lenyűgözni a főnököt" vagy "a bétatesztereknek kiadható állapotba hozni a rendszert" vagy "adminisztráció felület létrehozása" vagy bármi. A lényeg, hogy üzleti és ne technológiai cél legyen, hogy a csapaton kívüli emberek is megértsék.

A sprint célja arra az alapvető kérdésre ad választ, hogy "miért dolgozunk ezen a sprinten, miért nem megyünk szabadságra helyette?". Ami azt illeti, az egyik módja, hogy a Product Ownerből kiszedjük a sprint célját, hogy konkrétan feltesszük neki ezt a kérdést.

A cél persze egy olyan dolog kell, legyen, amit még nem értünk el. A "lenyűgözni a főnököt" megfelel célnak, de értelmetlen, ha a főnök a jelenlegi rendszerrel is teljesen elégedett, ugyanis akkor mindenki nyugodtan hazamehet és a cél úgy is teljesítve lesz.

Lehet, hogy a sprint célja bugyután hangzik a meetingen, de gyakran a sprint közepén jön elő a haszna, amikor az tagok nem tudják, mit is csináljanak. Ha, ahogy mi is, egyszerre több Scrum csapattal több terméken is dolgozik, hasznos összefoglalni a sprint célokat egy wiki oldalon vagy hasonló helyen ahol az egész vállalat (nem csak a felső vezetés) hozzáférhet és megtekintheti, hogy pontosan mit is csinál a cég - és miért.

A sprint alatt elvégzendő sztorik kiválogatása

A sprint planning meeting alatt az egyik fő tevékenység a sprint sztorijainak kiválasztása, azaz pontosabban hogy a Product Backlog mely sztorijait másoljuk át a sprint backlogba.

<figure>

A fenti ábra minden téglalapja egy-egy sztorit jelez fontosság szerint rendezve. A legfontosabb sztori a lista tetején található. A téglalap nagysága a sztori nagyságát jelképezi (azaz az időmennyiség becslését sztori pontokban). A kék zárójel nagysága jelzi a csapat becsült sebességét, vagyis, hogy az adott sprintben hány sztoripontnyi feladatot tudnak befejezni.

A jobb oldalon található sprint backlog a product backlogban található sztorik egy pillanatfelvétele. Ez tartalmazza azokat a sztorikat, amelyeket a csapat be kíván fejezni az adott sprintben.

Az, hogy hány sztorit vesznek be a sprintbe a csapat és nem a Product Owner vagy valaki más feladata.
Ez két kérdést vet fel:
1. Hogyan dönti el a csapat, hogy melyik sztorikat vegye be a sprintbe?
2. Hogyan befolyásolhatja a Product Owner a döntésüket?

Először a második kérdésre válaszolnék.

Hogyan befolyásolhatja a Product Owner, hogy mely sztorik kerülnek be a sprintbe?

Tegyük fel, hogy a következő szituáció áll elő a sprint planning meeting során:

<figure>

A Product Owner elégedetlen, mert a D sztori nem került be a sprintbe. Milyen lehetőségei vannak?

Az első opció, hogy újraértékeli a fontosságokat. Ha D-nek adja a legnagyobb prioritást, akkor a csapatnak kötelessége ezt bevenni a sprintbe (ebben az esetben a C sztorit kivéve a sprintből).

<figure>

A második lehetőség, hogy megváltoztatja a kiterjedést - az A sztorit addig egyszerűsíti, amíg a csapat szerint bele nem fér a sprintbe.

<figure>

A harmadik opció a sztorik kettévágása. A Product Owner dönthet úgy, hogy az A sztori néhány része
nem olyan fontos, tehát szétválaszthatja az A-t A1-re és A2-re különböző fontossággal.

<figure>

Amint láthatja, bár a Product Owner nem szabályozhatja a becsült sebességet, azért több mód is rendelkezésére áll, hogy befolyásolja a sprintbe kerülő sztorik listáját.

Hogyan dönti el a csapat, hogy mely sztorik kerüljenek a sprintbe?

Két taktikát szoktunk alkalmazni:
1. megérzés alapú becslés
2. sebességszámítás alapú becslés

Megérzés alapú becslés
* Scrum master: "Mit gondoltok, be tudjuk fejezni A-t ebben a sprintben?" (a product backlog legfontosabb elemére mutat)
* Lisa: "Simán! Három hetünk van rá és ez egy elég triviális feladat."
* Scrum master: "Oké, mi van, ha a B-t is bevesszük?" (a második legfontosabb elemre mutat)

* Tom és Lisa: "Naná!"
* Scrum master: "Oké, A, B és C együtt?"
* Sam (a Product Owner-hez): "C-ben benne van a teljes hibakezelés?"
* Product Owner: "Nincs, azt egyelőre kihagyhatjuk és csak az alap hibakezelést adjuk hozzá."
* Sam: "Akkor a C is belefér."
* Scrum master: "Oké, mi van, ha a D-t is hozzáadjuk?"
* Lisa: "Hmm..."
* Tom: "Szerintem belefér."
* Scrum master: "Mennyire biztos? 0%? 50%?"
* Lisa és Tom: "Közel 100%"
* Scrum master: "Oké, akkor a D-t bevesszük. Mi van az E-vel?"
* Sam: "Elképzelhető, hogy belefér."
* Scrum master: "0%? 50%?"
* Sam: "Mondjuk 50% körül."
* Lisa: "Hát én nem tudom..."
* Scrum master: "Oké, E-t kihagyjuk és A, B, C és D mellett kötelezzük el magunk. Ha belefér, akkor persze E-t is befejezzük, de mivel ez nem biztos, a sprintbe nem kerül be. Mit gondoltok?"
* Mindeni: "Rendben!"

A megérzés alapú becslés inkább kis csapatokban és rövid sprintek esetén szokott jól működni.

Sebességszámítás alapú becslés

Ez a taktika két lépést igényel:
1. Megállapodás egy becsült sebességben
2. A sebességkeretbe beleférő sztorik kiszámítása

A sebesség az "elvégzett munka" mértékegysége, ahol minden elemet a kezdeti becslésével súlyozunk.

A lenti ábra egy példa a sprint kezdetén becsült sebességre és a sprint végén számított valós sebességre. A téglalapok egy-egy sztorit jelölnek, a bennük található szám pedig a kezdeti becslés.

<figure>

Vegyük észre, hogy a valós sebesség is a kezdeti becslésen alapul, azaz a sprint közben történt esetleges változásokat figyelmen kívül hagyjuk.

Már hallom is, ahogy tiltakozik: "De ez miért jó? A sebesség mértéke így egy csomó olyan tényezőn múlik, mint hülye programozók, rossz kezdeti becslések, folytonosan növekvő feladatok, nem betervezett akadályok és így tovább..."

Egyetértek abban, hogy ez egy közelítő érték, de attól még hasznos tud lenni és mindenképpen több, mint a semmi. Remek kiindulópontot biztosít. "Minden tényezőtől függetlenül, íme a becsült és a tényleges időtartam közötti különbség."

Mi történik, ha egy sztorit nem sikerült teljesen befejezni a sprint alatt? Miért ne adjunk fél

pontokat a tényleges sebesség mérésekor? Ez jól tükrözi a Scrum (és minden agilis fejlesztési módszertan) szellemiségét, vagyis hogy a legfőbb dolog, hogy befejezett, kiadható dolgokat készítsünk.
A félkész termék értéke nulla (a valóságban akár negatív értéke is lehet). Ha többre kíváncsi ezzel kapcsolatban, olvassa el Donald Reinertsen "A dizájn-gyár irányítása" című könyvét.

És hogy milyen fekete mágiát használunk a sebesség becslésére?

Az egyik legegyszerűbb mód a sebesség becslésre a múltbeli értékek használata. Megnézzük, hogy hogyan adódott az elmúlt pár sprint sebessége és feltételezzük, hogy nagyjából most is ugyanannyi lesz.

Ezt a módszert "tegnapi időjárás"-nak is nevezik és csak olyan csapatok esetén működik, akik már maguk mögött tudnak néhány sprintet és a következő sprintet is hasonló körülmények között csinálják,
ami persze nem minden esetben igaz.

Ennek egy fokkal bonyolultabb változatában az erőforrások alapján számítunk. Tegyük fel, hogy egy 3
hetes sprintet tervezünk (ami 15 munkanap) egy négyfős csapatban. Lisa kivett két nap szabadságot, Dave csak az ideje felében tud segíteni és ráadásul az egyik napot szabadságon tölti.
Végeredményben:

<figure>

...vagyis erre a sprintre 50 ember-napot kapunk.

Tehát a becsült sebességünk 50? Nem, mivel a becslésünk mértékegysége a sztori pont, ami esetünkben
nagyjából egy "ideális ember-nap"-nak felel meg. Egy ideális ember-nap egy tökéletesen hatékony és zavartalan munkanap, ami a gyakorlatban nagyon ritkán fordul elő, ráadásul számolnunk kell előre nem
tervezett feladatokkal, vagy esetleg lebetegedő emberekkel is.

Éppen ezért a becsült sebességünk biztosan kisebb lesz 50-nél. No, de mennyivel kisebb? Erre használjuk a "koncentrációs tényezőt".

<figure>

A koncentrációs tényező arra ad közelítést, hogy a csapat mennyire tud összpontosítani a feladatokra.
Ha ez a tényező alacsony, az jelentheti, hogy a csapat sok akadállyal számol vagy például nem bízik a saját becsléseiben.

A koncentrációs tényező számításakor legreálisabb, ha az előző sprintet (vagy ami még jobb, a legutóbbi
néhány sprint átlagát) vesszük alapul.

<figure>

A valós sebesség az előző sprintben befejezett sztorik kezdeti becslésének összege.

Tegyük fel, hogy egy három fős csapat (Tom, Lisa és Sam) az előző három hetes (45 ember-napos) sprintben 18 sztori pontot fejezett be. Próbáljuk megbecsülni a következő sprint sebességét. Legyen ráadásul a helyzet még bonyolultabb és csatlakozzon a csapathoz egy új ember, Dave. Mindent
egybevetve számoljuk egy 50 ember-napos sprinttel.

<figure>

Tehát az előzetes becslésünk 20 sztori pont, azaz a csapat maximum 20 pont erejéig adhat sztorikat a sprinthez.

<figure>

Ebben a helyzetben a csapat választhatja a legfelső 4 sztorit 1 sztori pont értékben vagy a felső 5 sztorit 24 értékben. Tételezzük fel, hogy ebben az esetben a 4 sztorit választják, mivel ez van közelebb a 20 ponthoz (kétség esetén inkább válasszon kevesebb sztorit).

Mivel a 4 sztori összesen 19 sztori pontot jelent, a végső sebesség erre a sprintre 19-nek adódik.

A tegnapi időjárás hasznos tud lenni, de csak ha nem gondolkodás nélkül használja. Ha az előző sprint kivételesen rossz volt, mert mondjuk a csapat felét egy hétre leverte a láz, akkor nyugodtan feltételezheti, hogy ezúttal nem lesz ilyen pechje és használhat magasabb koncentrációs tényezőt. Ha
a csapat nemrég állított üzembe egy villámgyors continuous build rendszert akkor nyugodt szívvel növelheti a koncentrációs tényezőt. Ha egy új tag érkezett a csapathoz, akkor nyugodtan csökkentse a
koncentrációs tényezőt, mivel valószínűleg időt fog igénybe venni a betanítása.

A minél pontosabb becslés érdekébe amikor csak lehet, használja a legutóbbi pár sprint átlagát.

De mi van, ha a csapat teljesen új és még nincs semmilyen múltbeli adata? Ez esetben használhatja más csapatok hasonló helyzetben kapott koncentrációs tényezőjét. Ha nem ismer egyetlen más hasonló csapatot sem, akkor nyugodtan csapjon a hasára, hiszen ez a tényező csak az első sprintre vonatkozik, és utána már valós adatokra fog tudni hivatkozni.

Számomra az "alapértelmezett" koncentrációs tényező új csapatok esetében általában 70%, mivel a legtöbb csapatom valahol e körül mozgott.

Melyik becslési technikát használjuk?

A fentiekben ismertettem néhány technikát, a megérzésen alapuló becsléstől kezdve a tegnapi időjáráson át a rendelkezésre álló munkanapokon és a koncentrációs tényezőn alapuló számításig.

Hogy mi melyik technikát használjuk?

Általában a fentiek valamilyen elegyét használjuk, ami nem vesz túl sok időt igénybe.

Kezdetnek megnézzük az előző sprint tényleges sebességét és koncentrációs tényezőjét, majd összevetjük ezt a következő sprintre számított rendelkezésre állással és megbecsüljük a koncentrációs tényezőt. Ha a kettő között jelentős eltérések adódnak, akkor azt megbeszéljük és a végén megegyezünk egy értékben.

Ha már van egy előzetes sztori listánk erre a sprintre, akkor megnézem, hogy mennyire érzem reálisnak, illetve a csapatot is megkérem, hogy egy pillanatra felejtsék el a pontokat és mondják el, hogy mennyire érzik reálisnak ezt a listát a sprintre. Ha túl soknak tűnik, akkor kihagyunk egy sztorit, ha túl kevésnek, hozzáadunk egyet.

Végső soron a cél mindössze annyi, hogy kiválasszuk a sprintbe kerülő sztorikat. A koncentrációs tényezők, erőforrás rendelkezésre-állások és becsült sebességek mindössze eszközök arra, hogy ezt a
célt elérjük.

Miért használunk cetliket

A sprint planning meeting nagy részében a product backlog sztorijaira adunk becsléseket, rendezzük őket
fontosság alapján, tisztázzuk vagy épp kisebb feladatokra osztjuk őket.

Ez a gyakorlatban hogy is néz ki?

Régebben kivetítettük a product backlogot tartalmazó Excel-t és valaki (tipikusan vagy a Product Owner vagy a Scrum Master) a billentyűzet mellett ülve végigment az egyes sztorikon elindítva a beszélgetést. Amikor például a fontosság vagy egyéb hasonló részlet változott, ez a valaki frissítette a sztorit közvetlenül az Excel táblában.

Remekül hangzik? Hát nem az, hanem szívás. És ami a legrosszabb, hogy a csapat általában észre sem
veszi, hogy mekkora szívás amíg a meeting a vége felé nem közeledik és még mindig nem sikerült végigmenni a sztorik listáján.

Sokkal hatékonyabb megoldásnak találtuk a falra (vagy egy nagyobb asztalra) ragasztott cetliket.

<figure>

Ez messze jobb a kivetítőnél a következő okok miatt:
* Az emberek állnak és körbesétálhatnak => tovább maradnak éberek
* Sokkal személyesebb élményt nyújt (mintha csak egyvalakinél lenne a billentyűzet)
* Egyszerre több sztorin lehet dolgozni
* Triviális dolog fontosság alapján átrendezni a cetliket

* A meeting után a cetlik könnyen átvihetőek a csapatszobába és közvetlenül használhatóak a fali feladat-térképen. (Lásd: "Hogyan használjuk a Sprint Backlogot")

Írhatja kézzel is (ahogy mi is) de használhat például egy szkriptet, ami legenerálja ezeket a cetliket közvetlenül a Product Backlog-ból.

<figure>

Ui: a blogomról letölthető az általunk használt szkript: http://blog.crisp.se/henrikkniberg

Fontos! A Sprint Planning Meeting után a Scrum Master feladata a cetlik alapján kézzel frissíteni a product backlogot tartalmazó Excel táblát. Igen, ez némi többletmunkát jelent, de ez eltörpül a Sprint Planning Meetingben nyert hatékonyság mellett.

Egy gyors megjegyzés a "Fontosság" mezőről. A cetli megírásakor ez az Excel táblában található fontossággal azonos, ami segít egyszerűen fontossági sorrendbe rendezni a cetlijeinket (nálunk általában balról jobbra csökken a fontosság). Ha a cetlik már felkerültek a falra, akkor már nincs szükség ezekre a számértékekre, helyette egyszerűen a fizikai elhelyezkedés jelzi a fontosságot. Ha a Product Owner felcserél két cetlit, ne vesztegessen időt a fontosság mező frissítésével, elég, ha a meeting után frissíti az Excel táblát.

Az időbecslések általában egyszerűsödnek (és pontosabbá válnak) ha a sztorit kisebb feladatokra bontjuk. Mi általában az "aktivitás" szót használjuk, mert svéd nyelven a "feladat" szó teljesen más jelentéssel bír. Ez a felbontás is egyszerű a cetlik használatával, sőt több sztorit is könnyen szét lehet bontani párhuzamosan.

Ezt úgy szoktuk csinálni, hogy a sztori alá újabb cetliket ragasztunk, ahol minden cetli egy-egy feladat a sztorin belül.

<figure>

<figure>

Nem szoktuk felvezetni az Excel-alapú product backlogba magukat a feladatokat már csak két okból sem:
* A feladatok elég gyakran változnak a sprint során és túl sok munka lenne állandóan szinkronizálni a backlogot.
* A Product Owner amúgy sem kell, hogy részt vegyen ezen a szinten

Ahogy a sztori cetliket, úgy a feladat cetliket is közvetlenül fel lehet használni a sprint backlogban (Lásd "Hogyan használjuk a Sprint Backlogot").

A "kész" definíciója

Nagyon fontos, hogy a Product Owner és a csapat megegyezzen a "kész" pontos definíciójában. Készen
van-e a sztori, amikor a kódot feltöltjük a verziókövető rendszerbe? Vagy esetleg csak akkor,

amikor telepítettük a teszt rendszerre és az integrációs teszter csapat leellenőrizte? Mi a legtöbb esetben úgy definiáljuk a "készt", hogy "éles telepítésre készen", de néha csak annyit tudunk mondani, hogy "telepítve a teszt rendszerre és tesztre készen áll".

Kezdetben részletes listákat készítettünk, de újabban az a gyakorlat, hogy "egy sztori akkor van kész, amikor a csapat azt mondja, hogy kész van". Ezután már a teszterek dolga biztosítani, hogy a csapat valóban megértette a Product Owner szándékát és hogy a sztori megfelel az előre megegyezett
követelményeknek.

Arra jöttünk rá, hogy nem lehet minden sztorit ugyanúgy kezelni. A "Felhasználó-keresés" teljesen más, mint például a "Üzemeltetői kézikönyv", mivel az utóbbi esetében a "kész" definíciója lehet, hogy egyszerűen csak az, hogy "az üzemeltetők elfogadták". A legtöbb esetben a józan paraszti ész használata jobb eredményre vezet, mint egy részletes lista.

Ha gyakran merül fel probléma a "kész" definíciója kapcsán, érdemes lehet a sztorikhoz külön hozzáadni egy "kész definíciója" mezőt"

Időbecslés Planning Poker-rel

A becslés csapattevékenység, azaz a csapat minden tagja részt vesz minden sztori becslésében. Hogy miért?
* A sprint tervezés folyamán általában még nem tudjuk, hogy az egyes részeket ki fogja majd implementálni.
* A sztorik általában több különböző szakértelmet igényelnek (felhasználói felület dizájn, programozás, tesztelés, stb...)
* A becsléshez elengedhetetlen, hogy tudjuk, a sztori miről szól. Azzal, hogy mindenkit megkérdezünk, voltaképp azt biztosítjuk, hogy mindenkinek lesz valamennyi fogalma az adott sztoriról, ami cserében megnöveli annak a valószínűségét, hogy a csapattagok a sprint során segítenek majd egymásnak. Ráadásul a fontos kérdések és problémák így már a kezdetektől előkerülnek.
* Amikor tényleg mindenkit megkérdezünk, előfordul, hogy két csapattag drasztikusan eltérő becslést ad ugyanarra a sztorira. Az így felmerülő kérdéseket jobb a lehető leghamarabb tisztázni.

Amikor egy csapatot arra kér, hogy becsüljön meg egy feladatot, általában a feladatot legjobban ismerő tag fog először előállni egy becsléssel, ám ezzel befolyásolja a többiek döntését.

Egy kiváló stratégia ennek elkerülésére a Planning Poker (ha jól emlékszem Mike Cohn találmánya)

<figure>

Minden tag kap egy, a fentiekhez hasonló, 13 kártyából álló paklit. Amikor egy sztori becslésre kerül, a tagok kiválasztják és lefordítva az asztalra teszik azt a lapot, amely szerintük a becsült időtartamhoz (sztori pontban) a legközelebb áll. Amikor mindenki készen áll, a kártyákat egyszerre megfordítják. Így senki sem hagyatkozhat mások becslésére, azaz mindenkinek önállóan kell döntenie.

Ha drasztikus eltérés van két érték között, akkor a csapat megbeszéli a különbségeket és megpróbál egy lapra kerülni a sztorit illetően. Ha kell, a feladatot részekre is bonthatják és újra becsülhetnek. Ezt a folyamatot addig ismétlik, amíg konszenzusra nem jutnak (vagyis a becslések nagyjából azonosak).

Fontos emlékeztetni a csapatot, hogy a teljes munkát kell megbecsülniük és nem csak a "saját" részüket, vagyis egy teszter nem csak a tesztekre szükséges időt méri fel.

Vegyük észre, hogy a számsor nem lineáris, azaz például nincs semmi 40 és 100 között. Miért van ez?

Ezzel annak illúzióját próbáljuk meg eloszlatni, hogy a nagy időtartamokat pontosan meg tudjuk becsülni.
Ha egy sztorit 20 pontra értékeltünk, akkor nem igazán fontos azon vitázni, hogy ez pontosan 20 vagy
18 esetleg 21. Elég annyit tudnunk, hogy ez egy nagy feladat, amit nehéz előre felmérni, és amit mi nagyjából 20 pontra gondolunk.

Ha ennél pontosabb becslést szeretne, válassza szét a sztorit kisebb sztorikra és becsülje inkább azokat.

Nem ér csalni úgy, hogy egy 5-t és egy 2-t összerakva 7-t becsül. Vagy az 5-t vagy a 8-t kell választani, nincs 7 értékű becslés.

Néhány különleges lap:
* 0 = "ezt a sztorit már korábban befejeztük" vagy "ez teljesen triviális, legfeljebb pár percnyi munka".
* ? = "halvány lila gőzöm sincs, de tényleg."
* kávé = "túl fáradt vagyok gondolkodni, tartsunk pár perc szünetet."

Sztorik pontosítása

A lehető legrosszabb dolog amikor a csapat nagy büszkén bemutatja a működő új dolgot a sprint demóban, majd a Product Owner szomorúan közli, hogy "igen, ez szép és jó, csak épp nem ezt kértem".

Hogyan lehetünk biztosak, hogy a csapat és a Product Owner elképzelés a sztoriról ugyanaz, vagy hogy a csapattagok elképzelése megegyezik? A rossz hír, hogy sosem lehetünk teljesen biztosak, azonban van néhány stratégia legalább a nyilvánvaló félreértések felderítésére. Legegyszerűbben azzal
segíthet, ha a sztorihoz tartozó összes mezőt kitölti (egész pontosan az összes fontos és ezáltal a sprintbe valószínűleg bekerülő sztorihoz tartozó mezőket).

1. Példa:
Mind a csapat, mind a Product Owner elégedett a sprint tervével és már a meeting befejezésén gondolkodnak, amikor a Scrum Master megszólal: "várjunk csak, itt ez a 'felhasználó hozzáadása' sztori, amihez még nincs becslés." Néhány oda-vissza múlva a csapat megegyezik 20 pontban,

amire a
Product Owner dühösen felpattan: "Micsoda???" Pár percnyi heves vita után kiderül, hogy a csapat félreértette a feladatot és arra gondolt, hogy a 'felhasználó hozzáadása' egy 'szép webes felület, felhasználó hozzáadására, törlésére, keresésére', miközben a Product Owner mindössze arra gondolt,
hogy 'felhasználók hozzáadása manuálisan, valami egyszerű SQL használatával". Újabb becslés után
végül 5 pontban állapodnak meg.

2. Példa:
Mind a csapat, mind a Product Owner elégedett a sprint tervével és már a meeting befejezésén gondolkoznak, amikor a Scrum Master megszólal: "várjunk csak, itt ez a 'felhasználó hozzáadása' sztori, ezt hogyan fogjuk demonstrálni?". Pár perc tanácstalanság után valaki feláll és azt mondja "no igen, először is belépünk a weboldalra és...", amire a Product Owner közbevág: "Mi az, hogy belépünk a weboldalra? Nem, nem, nem és nem, ez a sztori egyáltalán nem része a weboldalnak, csak egy
egyszerű SQL szkript a rendszergazdáknak".

A "hogyan demonstráljuk" mező legyen minél tömörebb, egyébként sosem fog végezni a Sprint Planning
Meetinggel. Mindössze egy áttekintő és könnyen érthető teszt előírás a legtipikusabb esetekhez: "ezt és ezt csináld, majd ez történik és így ellenőrizhető a dolog". Pályafutásom során többször segített ez az egyszerű leírás felfedni kisebb-nagyobb félreértéseket. És minél előbb sikerül őket felfedezni, annál jobb!

Sztorik felbontása kisebb sztorikra

A sztorik ne legyenek se túl kicsik, se túl nagyok (a becsült érték alapján). Ha egy csomó fél pontos sztorija van, akkor valószínűleg a mikromenedzsment áldozata. Ugyanakkor egy 40 pontos sztori
nagy valószínűséggel csak félig fog elkészülni, ami nem túl hasznos a vállalatnak. Ráadásul ha a becsült sebesség 70 és a két legfontosabb sztori 40 40 pontos, akkor elég nehéz sprintet tervezni. Vagy kevesebbet vesz fel a listára (esetünkben csak egy sztorit) vagy túl sokat (a két legfontosabb sztorit).

Az a tapasztalatom, hogy egy nagy sztorit szinte mindig fel lehet bontani kisebb sztorikra, csak arra kell ügyelni, hogy a kisebb sztorik is kézzelfogható üzleti értéket képviseljenek.

Általában arra törekszünk, hogy a sztorijaink 2-8 ember-nap körül legyenek. Mivel a sebességünk valahol 40 és 60 között mozog, ez nagyjából 10 sztorit jelent egy sprintben. Néha ez az érték csak 5 és néha akár 15-ig is felmegy, de ezek tökéletesen kezelhetőek a cetlis módszerrel.

Sztorik felbontása feladatokra

Itt álljunk meg egy pillanatra és tisztázzuk, hogy mi a különbség a sztorik és a feladatok között.

Igazából a különbség viszonylag egyszerű: a sztorik olyan dolgok, amik fontosak a Product Owner

számára, míg a feladatok vagy nem önmagukban kiadhatóak, vagy olyan dolgok, amik nem számítanak a
Product Ownernek.

Példa egy sztori kisebb sztorikra darabolására:

<figure>

Példa egy sztori feladatokra osztására:

<figure>

Néhány figyelemre méltó észrevétel:
* Új Scrum csapatok általában húzódzkodnak a sztorik feladatokra bontásától, mivel úgy érzik, hogy ez túlságosan is hasonlít a vízesés-modellre.
* Olyan sztorik esetében, amit mindenki tökéletesen ért, utólag is ugyanolyan egyszerű a feladatokra bontás.
* A feladatokra bontás eredményeként gyakran kerülnek elő új részletek és az időtartamra adott becslések gyakran növekednek, cserében viszont a sprint terve sokkal életszerűbbé válik.
* A felbontásnak köszönhetően a Napi Scrum sokkal hatékonyabbá válik (lásd "Hogyan használjuk a Napi
Scrum-ot")
* Még ha a felbontás pontatlan és később módosításokra szorul is, a fenti előnyök akkor is jelen vannak.

Éppen ezért úgy próbáljuk ütemezni a sprint tervezést, hogy a feladatokra bontás is beleférjen, azonban nem nagy tragédia, ha mégis kifutunk az időből (Lásd: "Hol húzzuk meg a határt")

Megjegyzés: Mivel TDD-t (Teszt Alapú Fejlesztés) használunk, a gyakorlatban az első feladat mindig a "meghiúsuló teszt írása" és az utolsó mindig "refaktorálás" (= a kód olvashatóságának javítása és az esetlegesen duplikált kód eltávolítása).

A Napi Scrum helyének és idejének kijelölése

A sprint planning meeting egy gyakran elfelejtett része a "Napi Scrum helyének és idejének kijelölése". E nélkül viszont a sprint elég rosszul kezdődik, mivel az első napi scrum gyakorlatilag az a hely, ahol a tagok eldöntik, hogy min is dolgozzanak.

Én a reggeli meetingek híve vagyok, de be kell valljam, hogy tulajdonképpen sosem próbáltuk délre vagy délutánra ütemezni a napi scrumot.

A délután tartott scrum meeting hátránya, hogy emlékeznie kell arra, hogy mit is mondott tegnap, min fog ma dolgozni.

A reggeli scrum meeting hátránya, hogy emlékeznie kell, hogy mit is csinált tegnap.

Véleményem szerint az előbbi hátrány a jelentősebb, mivel sokkal fontosabb az, hogy mit fog csinálni,
mint az, hogy eddig mit csinált.

Mindig azt a legkorábbi időpontot próbáljuk választani, amin már senki sem morog. Ez általában 9:00, 9:30 vagy 10:00, de a legfőbb, hogy olyan időpontot válasszon, amibe a csapat minden tagja beleegyezik.

Hol húzzuk meg a határt

Kicsúszni látszunk az időből, mit hagyhatunk ki a sprint planning meetingből?

Én általában az alábbi fontossági sorrendet használom:

1: A sprint célja és a demó időpontja. Ez az abszolút minimum a sprint elkezdéséhez. Ha a cél világos és meg lett választva a sprint vége, a csapat elkezdhet dolgozni közvetlenül a Product Backlogból, ami persze szívás és komolyan érdemes elgondolkodni egy új meeting összehívásán, de ha
tényleg muszáj elkezdeni a sprintet, akkor ez is megteszi. Ugyanakkor őszintén szólva én ennyi információval még sosem kezdtem sprintet.
2: A sprintbe felvett sztorik listája
3: Az összes sztori becsült időtartama
4: A "Demó módja" mező kitöltve az összes sprintbe foglalt sztorihoz
5: Sebességszámítás. A csapattagok listája és rendelkezésre állásuk mértéke (ami nélkül nem lehet a sebességet kiszámítani)
6: A napi scrum helyének és idejének kijelölése. Ez legfeljebb pár percet vesz igénybe, de ha kifut az időből, a Scrum Master utólag is dönthet és e-mailben értesítheti a csapattagokat.
7: A sztorik felbontása feladatokra. Ezt lehet naponta is csinálni a Napi Scrum keretein belül, de az valamelyest megtöri a sprint lendületét.

Technikai sztorik

A technikai jellegű sztorik (vagy nem-funkcionális elemek) meglehetősen összetett problémát jelentenek.

Ezek olyan dolgok, amiket bár meg kell csinálni, közvetlenül nem adhatóak ki és nem igazán tartoznak
egyik másik sztorihoz sem. Ráadásul nem képviselnek közvetlenül értéket a Product Owner számára.

Például:
* Continuous build rendszer telepítése
  o azért kell, mert rengeteg időt megspórol a fejlesztők számára és csökkenti annak esélyét, hogy az iteráció végén integrációs problémák lépnek fel.
* Áttekintő rendszerterv készítése
  o azért szükséges, mert a fejlesztők nem tudják állandóan fejben tartani a rendszer felépítését és ez inkonzisztens kódot eredményez. Szükséges egy áttekintő dokumentum, amire a tagok

rendszeresen
hivatkozhatnak.
* Adatbázis-elérési réteg refaktorálása
  o azért szükséges, mert az adatbázis-elérési réteg eléggé rossz állapotban van már, ami lassítja a fejlesztést a sok hiba és félreértés miatt. A kód megtisztításával rengeteg időt lehet megtakarítani és azzal egyidőben a rendszer stabilitása is javul.
* JIRA (hibakövető) frissítése
  o azért kell, mert a jelenlegi verzió túl lassú és tele van hibákkal, a frissítés sok időt takarít meg.

Ezek vajon normális sztorik, vagy esetleg olyan feladatok, amik nem tartoznak egy konkrét sztorihoz sem? Ki határozza meg a fontosságukat? Egyáltalán a Product Owner szükséges az ilyen dolgokhoz?

Mi több különböző módszert is kipróbáltunk. Használtunk sztorikat rájuk, ami nem igazán működött, mert a Product Ownernek nem volt viszonyítási alapja amikor a Product Backlogot rendezte fontossági
sorrendbe. Ami azt illeti, a technikai jellegű sztorik gyakran kaptak alacsony prioritást olyan dolgokra hivatkozva, mint "értem én, hogy a continuous build rendszer fontos, de azért próbáljuk a valós bevételt jelentő sztorikat megcsinálni először, aztán majd gondolkodhatunk ezeken a technikai izéken, értem?"

Az esetek egy részében persze igaza van, ám nagyon gyakran nincs. Arra jutottunk, hogy a Product Owner nem mindig a megfelelő személy a fontosság eldöntésére.

Ehelyett az alábbiakat tesszük:
1) Amennyire csak lehet kerüljük a technikai jellegű sztorikat és megpróbáljuk őket valamiféle mérhető üzleti értéket biztosító sztorivá változtatni, hogy a Product Owner könnyebben mérlegelhesse
a fontosságát.
2) Ha ez nem lehetséges, akkor megpróbáljuk egy másik sztori keretein belül elvégezni a munkát. Például az "adatbázis-elérési réteg refaktorálása" elvégezhető a "felhasználó adatainak szerkesztése" sztori keretein belül, mivel az érinti az adatbázis-elérési réteget.
3) Ha ez sem kivitelezhető, akkor technikai jellegű sztoriként vesszük fel, de egy különálló listára, amit a Product Owner megtekinthet, ám nem szerkeszthet. A "koncentrációs tényező" és "becsült sebesség" értékeket használva megpróbálunk megegyezni a Product Ownerrel, hogy a sprint egy
részét a technikai sztorikra fordíthassuk.

Példa (az alábbihoz kísértetiesen hasonló párbeszéd zajlott le az egyik sprint planning meetingünkön)
* Csapat: "Van pár belső technikai jellegű feladat, amit jó lenne megcsinálni. Szeretnénk erre használni az időnk 10%-t, vagyis csökkentsük a koncentrációs tényezőt 75%-ról 65%-ra. Ez elfogadható?"
* Product Owner: "A frászt fogadható el, nincs most erre időnk!"
* Csapat: "Nézd, az előző sprintben a becsült sebességünk 80 volt, de a végére a valós sebességünk csak 30-ra jött ki."

* PO: "Pontosan, ezért nincs időnk ilyen technikai dolgokra, muszáj végezni az új feladatokkal."
* Csapat: "Igazából azért teljesítettünk ilyen rosszul, mert túl sok időt töltöttünk azzal, hogy a release tesztre kész állapotba kerüljön."
* PO: "Igen, és?"
* Csapat: "Szóval valószínűleg a sebességünk ezután is ilyen rossz marad, hacsak nem teszünk valamit."
* PO: "Igen, és?"
* Csapat: "Szóval azt javasoljuk, hogy a következő sprintből szánjunk 10%-ot egy continuous build szerver és hasonló eszközök telepítésére, amivel egyszerűbbé válik az integráció. Ez legalább 20%-os sebesség növekedést fog jelenteni az összes utána következő sprintben."
* PO: "Tényleg? Miért nem csináltuk ezt meg az előző sprintben?"
* Csapat: "Öööö, mert te nem akartál hallani se róla."
* PO: "Öööö, oké, így elmondva jó ötletnek tűnik."

Persze egy másik megoldás, a Product Owner-t teljesen kihagyni az egészből és nem engedni a javasolt koncentrációs tényezőből, de célszerű legalább megpróbálni konszenzusra jutni vele.

Amennyiben a Product Owner hozzáértő és értelmes ember (és ebben mi nagyon szerencsések voltunk), én
azt javaslom, hogy amennyire csak lehet beszéljen vele és hagyja, hogy ő hozza meg a fontossági sorrendet
illető döntéseket. Az áttekinthetőség végül is a Scrum egyik alapértéke.

Hibakövető rendszer és a Product Backlog

Az Excel egy remek formátum a Product Backloghoz, de mindenképp szükség van hibakövető rendszerre
is, amire viszont az Excel nem a legjobb választás. Mi például JIRA-t használunk.

De hogyan hozzuk a JIRÁ-ban található feladatokat a sprint planning meetingre? Az azért mégsem járható út, hogy egyszerűen elfelejtjük őket és csak a sztorikra koncentrálunk.

Itt is többféle stratégiával próbálkoztunk:
1) A Product Owner kinyomtatja a legfontosabb JIRA bejegyzéseket és ezeket tesszük a többi sztori mellé a falra (amivel egyben a többi sztorihoz viszonyított fontosságuk is meghatározásra kerül)
2) A Product Owner olyan sztorikat készít, amelyek hivatkoznak a JIRA bejegyzésekre, például "A legfőbb hibák kijavítása: Jira-124, Jira-126 és Jira-180".
3) A hibajavítást nem tekintjük a sprint részének, azaz a csapat elég alacsonyan tartja a koncentrációs tényezőt (például 50%), hogy maradjon idő a hibák kijavítására. Ezzel gyakorlatilag implicite feltételezzük, hogy a csapat az ideje egy részét minden sprintben hibák javításával tölti.
4) A Product Backlogot is JIRÁ-ban tároljuk (azaz kidobjuk az Excelt). A hibákat is normális sztoriként jegyezzük.

Nem igazán sikerült még eldönteni, hogy melyik stratégia a legjobb, mivel ez csapatról csapatra és sprintről sprintre változik. Én az első stratégia felé hajlok, mivel az elég egyszerű.

A Sprint Planning Meeting vége

Sosem gondoltam volna, hogy ilyen hosszúra nyúlik majd ez a fejezet. Ez valahol biztos azt tükrözi, hogy talán a Scrum legfontosabb részének tartom a Sprint Planning Meetinget. Ha elég időt szán ennek tökéletesítésére, a maradék sokkal egyszerűbb lesz.

A Sprint Planning Meeting akkor tekinthető sikeresnek, ha mindenki (az összes csapattag és a Product
Owner is) mosolyogva távozik és másnap is mosollyal ébred és kezdi meg az első Napi Scrum-ot.

De persze van egy csomó dolog, ami még összeomolhat a sprint során, de legalább a sprint terve tökéletes volt. :)

5 Hogyan kommunikáljuk a sprinteket
==========
Ha nem informáljuk az egész vállalatot, az könnyen vezethet elégedetlenséghez, vagy - ami még rosszabb - hibás feltételezésekhez.

Erre való a "Sprint adatlap".

<figure>

Néha a sztorik demonstrálásának módját is jelezzük.

A Scrum Master feladata ezt az oldalt elkészíteni, a wikire feltölteni és a vállalat minden alkalmazottjának elküldeni a linket e-mailben.

<figure>

Ezen túl a wikiben van még egy "állapotjelző" oldalunk is, amelyről linkek vezetnek az összes folyamatban lévő sprint oldalára.

<figure>

A Scrum Master feladata továbbá a sprint adatlapot kinyomtatni és a csapat szobájának ajtajára tűzni, hogy bárki, aki arra jár láthassa, hogy a csapat éppen min dolgozik. Mivel ez tartalmazza a napi scrum helyét és idejét, tudja, hogy hova kell mennie, ha többre kíváncsi.

Amikor a sprint a végéhez közeledik, a Scrum Master kötelessége emlékeztetni mindenkit a demóra.

<figure>

Mindezek mellett igazán senki sem találhat kifogásokat, hogy miért nem tudta, hogy mi történik.

6 Hogyan vezetjük a Sprint Backlogot
==========
Gratulálok, hogy már eddig eljutott.

Most, hogy vége a Sprint Planning Meetingnek és már mindenki tud az új sprintről, ideje, hogy a Scrum Master elkészítse a sprint backlogot. Ezt a Sprint Planning Meeting után, de még az első Napi Scrum előtt kell megcsinálni.

A Sprint Backlog formátuma

Több formátumot is kipróbáltunk, többek között Jirá-t, Excel-t és falra ragasztható feladat-térképet is. Kezdetben főként Excel-t használtunk, amihez rengeteg template-t lehet letölteni sprint backlogtól kezdve automatikusan generált Burndown Chart-ig. Beszélhetnék arról, hogy milyen finomításokon esett át ez az Excel-alapú sprint backlog, de mégsem fogok, sőt még példákat sem fogok felhozni.

Ehelyett inkább arról a formátumról szeretnék beszélni, amit végül a leghatékonyabb megoldásnak találtunk: a fali feladat-térképről.

Először is vegyen egy használaton kívüli nagyméretű falat (olyan is megteszi, amin csak amúgy használatlan elemek vannak, mint például céglogó, idejétmúlt ábrák vagy randa képek). Tisztítsa le a falat (ha muszáj, kérjen előtte engedélyt), majd ragasszon fel egy nagy, legalább 2-szer 2 méteres (nagy csapatok esetén legalább 3x2 méteres) papírt. Ezután alakítsa a felületet az alábbihoz hasonló módon:

<figure>

Használhat persze letörölhető táblát is, de az nekem túlzásnak tűnik, szóval ha lehet, a táblákat inkább tartogassa a design tervezésre és használjon sima papírt a feladat-térképhez.

Megjegyzés: ha a feladatokra post-it cetliket használ, ne felejtse el rögzíteni őket ragasztószalaggal is, nehogy a földön végezzék.

Hogyan működik a feladat-térkép

<figure>

Persze használhat további oszlopokat is, mint például "Integrációs tesztre vár", vagy "Törölve", de gondolja át kétszer is, mielőtt hozzáadja őket, mert bonyolíthatják a helyzetet.

Tapasztalatom szerint az egyszerűség rendkívül értékes szempont, ezért tényleg csak akkor növelem a
bonyolultságot ilyen dolgokkal, amikor egyébként a költségük nem túl jelentős.

1. Példa - az első Napi Scrum után

Tegyük fel, hogy az első Napi Scrum után a feladat-térkép így néz ki:

<figure>

Amint az látható, három feladatot "vettek fel", azaz a csapat ma ezeken fog dolgozni.

Nagyobb csapatok esetén előfordulhat, hogy egy feladat "felvett" állapotban ragad, mert senki sem emlékszik, hogy ki is dolgozott rajta. Ha ez gyakran fordul elő, akkor előírható, hogy a felvett feladatokra a felvevő ráírja a nevét.

2. Példa - pár nappal később

Egy pár nap múlva a tábla mondjuk valahogy így néz ki:

<figure>

Itt látható, hogy már befejeztük a "betét" sztorit (azaz benn van a verziókövető rendszerben, átment a teszteken és refaktorálva lett, stb...). A migrációs eszköz részben kész, az adminisztrációs felület bejelentkezés funkcióját elkezdtük és a felhasználó adminisztráció funkciója még nem lett elkezdve.

Három előre nem tervezett feladat is látható a jobb alsó részen, amire érdemes emlékezni a sprint visszatekintőben.

Az alábbiakban egy tényleges sprint backlog látható a sprint vége felé. Elég kuszává tud válni a sprint előrehaladtával, de ez nem nagy probléma, mivel ez a dokumentum rövid életű és a következő sprint elején tiszta lappal indítjuk a sprint backlogot.

<figure>

Hogyan működik a Burndown Chart

Nagyítsuk csak ki a burndown chartot:

<figure>

Ez a grafikon a következőket mutatja:
* A sprint első napján (augusztus 1) a csapat nagyjából 70 pontra becsülte a fennmaradó munkamennyiséget, ami tulajdonképpen a sprint becsült sebessége.
* Augusztus 16-n a fennmaradó munkamennyiséget körülbelül 16 pontra becsülték. A szaggatott vonal
  alapján láthatjuk, hogy nagyjából menetrend szerint haladnak és a sprint végére mindent befejeznek.

Az x-tengelyen nem tüntetjük fel a szombat-vasárnapokat, mivel ilyenkor ritkán történik érdemi munka. Régebben ezeket is jeleztük, de ettől a grafikon elég furán mutatott, mivel hétvégenként lapos a gráf, ami viszont összetéveszthető más intő jelekkel.

Figyelmeztető jelek

Ránézésre meg kell tudni állapítanunk, hogy pontosan hogyan is halad a sprint. A Scrum Master

feladata a figyelmeztető jelekre reagálni.

<figure>

<figure>

<figure>

<figure>

Jó, de mi van a nyomon követéssel?

A legjobb megoldás a nyomon követésre hogy naponta lefényképezzük a feladat-térképet. De csak ha
tényleg muszáj. Néha én is le szoktam fényképezni, de őszintén szólva eddig még nem volt példa arra,
hogy valamiért szükség lett volna a felvételekre.

Ahol a nyomkövetés tényleg elengedhetetlen, ott elképzelhető, hogy a feladat-térképes megoldás nem
optimális.

Azt javaslom, hogy próbáljon utánagondolni, hogy mekkora valódi értéket képvisel egy részletes sprint történet. Ha egyszer a sprintnek vége és a a működő kódot kiadták, a dokumentáció fel lett töltve, tulajdonképpen kit érdekel, hogy a sprint ötödik napján hány sztori volt készen? Hasonlóképp, kinek érdekes, hogy mennyi volt a kezdeti becslés a "meghiúsuló teszt írása a Betét sztorihoz"?

Napokban becslés órák helyett

A Scrum-ról szóló irodalomban többnyire azt fogja találni, hogy napok helyett órákban történik a becslés. Egy ideig mi is ezt használtuk, nálunk 1 ideális ember-nap 6 ideális ember-órának felelt meg.

Ezt a legtöbb csapatban már nem használjuk a következő okok miatt:
* Az ember-óra túl finom és ezért több 1-2 órás feladathoz és így mikromenedzsmenthez vezet.
* Mint kiderült, mindenki amúgy is napokban gondolkodott és csak felszorozták 6-tal. "Hm, ez a feladat nagyjából egy nap, de ide órákat kell írni... oké, legyen 6 óra"
* Két különböző egység csak félreértéseket szül "Ez a becslés akkor most órában vagy napban volt?"

Úgyhogy most már minden becslésünk ember-nap egységben történik (persze sztori pontnak hívjuk). A
legkisebb érték 0.5, vagyis minden 0.5-nél kisebb feladatot vagy törlünk, vagy összevonunk egy másik
feladattal, esetleg csak szimplán 0.5-nek vesszük (egy kis túllövés nem okoz nagy problémákat). Így a legegyszerűbb.

7 Hogyan rendezzük be a csapatszobát

Tervezői sarok

Azt vettem észre, hogy a legérdekesebb tervezői viták általában spontán, egy tábla előtt történnek, ezért mindig ki próbálunk alakítani egy kimondottan az ilyen beszélgetéseket elősegítő "tervezői sarkot".

<figure>

Ez azért hasznos, mert nincs jobb módja egy rendszer megértésére, mint a tervezői sarokban a két falat áttekinteni, majd a számítógépen kipróbálni a legfrissebb buildet (már ha elég szerencsés, és használ folyamatos build rendszert - lásd "Hogyan elegyítjük az XP-t a Scrum-mal").

A "tervezői fal" igazából csak egy nagy tábla, amin a legfontosabb diagramok és kinyomtatott dokumentumok találhatóak (például szekvencia-diagramok, GUI prototípusok, üzleti modellek, stb...)

<figure>

Fent: egy Napi Scrum az előzőekben említett sarokban

Hm... az a burndown gyanúsan szép egyenes, nem? De a csapat szerint ez a valóság :)

Ültesse egybe a csapatot!

Egy dolog van az iroda ülési rendjének tervezésekor, amit nem tudok eléggé hangsúlyozni.

Ültesse egybe a csapatot!

Csak a világosság kedvéért ez a következőt jelenti:

!!! Ültesse egybe a csapatot !!!

Az emberek - legalábbis ahol eddig dolgoztam - nem szeretnek helyet cserélni. Nem szeretik, ha össze
kell szedniük a dolgaikat, kihúzni a számítógépeket és minden cuccukat áttelepíteni az új helyre, összerakni a gépeket, stb. Ráadásul minél közelebb költöznek, annál nagyobb ez a húzódzkodás: "Ugyan már, főnök, mi értelme 5 méterrel odébb ülni?".

Ha azonban hatékony Scrum csapatot szeretnénk építeni, akkor nincs más választásunk, muszáj a csapatot egy helyre ültetni, még ha személyesen is kell megfenyegetnie az összes tagot, magának kell
átcipelnie az összes cuccukat és letörölni a régi kávéfoltokat. Ha pedig nincs elég hely az egész csapatnak, akkor csináljon. Bárhol, ha kell akár az alagsorban is. Tolja szét az asztalokat, fizesse le az irodavezetőt, de csináljon elég helyet, hogy a csapat egy helyen dolgozhasson.

Az eredmény azonnal jelentkezni fog, már egy sprint után a csapat egyet fog érteni abban, hogy jó ötlet volt (személyes tapasztalatból beszélek, de elképzelhető, hogy a csapata túl makacs ahhoz ezt nyíltan bevallja).

És hogy mit is jelent pontosan az "egybeültetés"? Hogyan rendezze el az asztalokat, például? Nos, nincs igazán kialakult véleményem a "legjobb" asztal elrendezésről és még ha lenne is, nem hiszem, hogy minden csapatnak megadatik az a luxus, hogy teljesen szabadon döntsön erről. Gyakran szab határt a tér nagysága, a szomszédos csapatok, a vécéajtó, a szoba közepére rakott automata és hasonló dolgok.

Az "egybeültetés" a következőt jelenti:
* Hallhatóság: a csapat minden tagja képes bárki mással kiabálás nélkül beszélni anélkül, hogy a saját asztalától fel kelljen kelnie.
* Láthatóság: a csapat minden tagja lát mindenki mást és elég közel ülnek a feladat-térképhez, hogy probléma nélkül láthassák azt.
* Elszigeteltség: ha az egész csapat hirtelen feláll és heves vitába kezd a rendszer architektúráját illetően, ezzel nem zavarják a kívülállókat (és viszont).

Az "elszigeteltség" persze nem jelenti az, hogy a csapat teljesen "elszigetelt". Egy fülke-rendszerű környezetben elég lehet, ha a csapatnak saját tere van elég elhatároló elemmel, ami kiszűri a legtöbb külső zajt.

Ha pedig földrajzilag elosztott a csapata, akkor nincs szerencséje. Próbáljon minél több technikai eszközt bevetni, hogy minimalizálja a távolságot: videó-konferencia, webkamerák, távoli asztal, stb.

Legyen kéznél a Product Owner

A Product Owner legyen elég közel a csapathoz, hogy a tagok bármikor odamehessenek tisztázni valamit
és hogy ő is bármikor egyszerűen megtekinthesse a feladat-térképet, azonban ne ültesse közvetlenül egybe a csapattal, mert akkor nagy valószínűséggel nem fog tudni ellenállni a kísértésnek, hogy beleszóljon a részletekbe, ami viszont megakadályozza, hogy a csapat kellőképpen összeforrjon (azaz
hogy eljusson egy rendkívül hatékony, önmenedzselő állapotba).

Persze őszintén szóval ez mind spekuláció, mivel még sosem voltam abban a helyzetben, hogy a Product
Owner valóban együtt ült volna a csapattal, szóval csak a saját megérzéseimre és más Scrum Masterek
elbeszéléseire tudok hivatkozni.

Tartsa kéznél a menedzsereket és az oktatókat

Elég nehéz erről beszélnem, hiszen én menedzser is és oktató is voltam és így az volt a feladatom, hogy minél szorosabban dolgozzam a csapattal. Én alakítottam ki a csapatokat, pár-programoztam a tagokkal, én képeztem a Scrum Mastereket és én hívtam össze sprint planning meetingeket. Így visszatekintve a legtöbben azt gondolhatták, hogy ez egy Jó Dolog™, mivel nekem már volt

tapasztalatom az agilis fejlesztésben.

Ugyanakkor én voltam a (ide képzelje Darth Vader zenéjét) vezető fejlesztő is, ami gyakorlatilag egy menedzser szerepkör, vagyis ha a csapat része vagyok, az automatikusan azt jelenti, hogy az kevésbé önmenedzselővé válik. "A francba, itt a főnök, biztos rengeteg ötlete van, hogy mit és hogyan csináljunk, inkább hagyom, hadd beszéljen ő."

Arra próbálok kilyukadni, hogy ha Ön Scrum tréner (és esetleg egyben menedzser is), próbáljon minél
közelebb kerülni a csapathoz. De ezt csak rövid távon tegye, utána álljon félre és hagyja, hogy a csapat összerázódjon és saját magát irányítsa. Nézzen rájuk időről időre (ne túl gyakran), kövesse a sprint demókat, tartsa szemmel a feladattáblát és vegyen részt szemlélőként a napi scrumon. Ha valami olyat lát, amin javítani lehetne, vonja félre a Scrum Mastert és beszélje meg vele, de ne a csapat előtt. Az is jó ötlet, ha beül a sprint visszatekintőkre (lásd "Hogyan használjuk a Sprint Visszatekintőket"), már amennyiben úgy gondolja hogy élvezi annyira a csapat bizalmát, hogy nem fognak elhallgatni csak azért mert Ön megjelenik.

Egy jól működő Scrum csapatban győződjön meg róla, hogy minden szükséges támogatást megkaptak, majd
álljon félre az útjukból (a sprint demó kivételével).

8 Hogyan bonyolítjuk a Napi Scrum-ot
==========
A Napi Scrumjaink meglehetősen előírás szerintiek, mindig pontosan kezdődnek, minden nap ugyanabban
az időben. Eleinte a sprint planning idejére átvonultunk egy másik helyiségbe (azokban az időkben, amikor elektronikus sprint backlogot használtunk), de mostanra a napi scrum-ot mindig a csapat szobájában a feladat-térkép előtt végezzük. Ezt találtuk a legjobb megoldásnak.

A meeting természetesen állva zajlik, mivel ez csökkenti annak az esélyét, hogy tovább tart 15 percnél.

Hogyan frissítjük a feladat-térképet

Általában a napi scrum alatt frissítjük a táblát. Miközben valaki elmondja, hogy mit csinált tegnap és mit fog csinálni ma, egyúttal át is rendezi a cetliket. Ha egy előre nem tervezett feladatról számol be, feltesz egy új cetlit. Ha frissítette az időbecsléseit, azt rögtön felvezeti a táblára
is. Esetenként a Scrum Master végzi a cetlik frissítését, amíg az egyes tagok beszélnek.

<figure>

Egyes csapatok abban állapodtak meg, hogy mindenki frissíti a feladat-térképet a meetingek előtt, ami
ugyanolyan jól működik. A lényeg, hogy kialakítsanak egy stratégiát és hogy betartsák azt.

Függetlenül a backlog formátumától, próbálja a teljes csapatot bevonni a backlog naprakészen tartásába. Próbálkoztunk olyan sprintekkel, ahol egyedül a Scrum Master volt felelős a sprint backlog

frissítéséért és így minden nap körbe kellett mennie és mindenkitől egyenként megkérdeznie, hogy mennyi
idő maradt még a feladataik befejezéséig. Ennek a módszernek a hátrányai azonban a következőek:
* A Scrum Master túl sok időt tölt adminisztrációval ahelyett, hogy támogatná a csapatot a felmerülő akadályok elhárításával.
* A csapattagok nem látják át a sprint aktuális helyzetét, mivel nem szükséges foglalkozniuk a sprint backloggal, ami csökkenti a csapat flexibilitását és koncentrációját.

Egy jól átgondolt sprint backlog esetén nem okozhat problémát az egyes tagoknak frissíteni azt.

Közvetlenül a Napi Scrum után valaki összefoglalja az időbecsléseket (a "kész" feladatokat természetesen
kihagyva) és berajzol egy újabb pontot a sprint burndown grafikonra.

Mit tegyünk a későkkel

Néhány csapat tart egy perselyt, ha valaki késik - akár csak egy percet is - bedob ebbe valami előre megállapodott összeget. Akadékoskodás nélkül. Még ha előzőleg jelezte is, hogy késni fog, akkor is fizetnie kell és csak nagyon jó kifogással lehet kikerülni a fizetést (például orvosnál vagy esküvőn volt).

A pénzt aztán közösségi eseményekre fordítják, mint például pizza rendelésre az irodai LAN-partin.

Ez jól működik, de igazán csak akkor számít, ha gyakran késnek az emberek.

Mit tegyünk, ha valaki "nem tudja mit csináljon ma"

Nem ritka azt hallani, hogy "Tegnap ezt és ezt csináltam, de ma halvány lila gőzöm sincs, hogy mit csináljak". Most mi legyen?

Tegyük fel, hogy Joe és Lisa azok, akik nem tudják, min dolgozzanak ma.

Ha én vagyok a Scrum Master, gondolatban megjegyzem ezt, de továbblépek a következő emberre. Miután
mindenki végzett, végigmegyek a feladatlistán az egész csapattal és leellenőrzöm, hogy minden frissítve van-e és hogy mindenki tudja, hogy mi mit jelent. Lehetőséget adok új cetlik hozzáadására is. Ezután azokhoz fordulok, akik nem tudták, mit csináljanak és azt kérdezem: "most, hogy végigmentünk a feladatlistán, van már ötletetek, hogy mit csináljatok ma?". Remélhetőleg lesz.

De ha mégsem, akkor ez például remek lehetőség a páros programozásra. Tegyük fel, hogy Niklas ma az
adminisztrációs felület GUI-ján dolgozik. Joe-t vagy Lisá-t megkérhetem, hogy dolgozzon együtt Niklas-szal, ami általában jól működik.

És ha ez sem működik, van másik:

Scrum Master: "OK, ki szeretné bemutatni a bétára kész kiadást" (feltéve, hogy az volt a sprint célja)
Csapat: értetlen csend
Scrum Master: "Hát nem vagyunk készen?"
Csapat: "öööö... nem"
Scrum Master: "A fenébe. Miért, mi van még hátra?"
Csapat: "Hát még nincs egy teszt szerverünk se, ráadásul a build szkript se fut."
Scrum Master: "Aha" (feltesz két új feladatot a listára) "Joe, Lisa, hogyan tudtok segíteni ma?"
Joe: "Öööö... megpróbálhatok keríteni egy teszt szervert valahonnan"
Lisa: "... én pedig megnézem mit lehet kezdeni a build szkripttel."

Ha mázlis, lehet, hogy valaki tényleg be fogja mutatni a bétára kész kiadást, amit kért. Ez szuper dolog, sikerült elérnie a sprint célját. De mi legyen, ha még csak a sprint közepén tart? Egyszerűen gratuláljon a csapatnak a sikeres munkához, majd tegyen át egy pár sztorit a feladat-lista "következő" részéből a "még nincs elkezdve" részre. Értesítse a Product Ownert, hogy hozzáadott néhány feladatot a sprinthez.

Mit lehet tenni akkor, ha a csapat még nem érte el a sprint célját, de Joe és Lisa továbbra sem állnak elő semmilyen használható dologgal. Én általában a következő taktikákat alkalmazom (egyik se
túl kedves, de végtére is ez a legutolsó dolog, amivel próbálkozom):

* Megszégyenítés: "Ok, ha nincs semmi kézzelfogható ötletetek, akkor menjetek haza, vagy olvassatok
  egy könyvet, esetleg csak üljetek csendben, amíg valakinek szüksége van rátok".
* Oldschool: Csak simán osszon ki rájuk egy feladatot.
* Csoportnyomás: "Nyugodtan gondolkodjatok csak, az egész csapat itt fogja várni, amíg előálltok valami használható ötlettel".
* Szolgalélek: "Ok, akkor közvetve segíthetitek a csapatot. Ma egész nap teljesíthetitek mindenki kívánságait, hozhattok kávét, kitakaríthattok, főzhettek ebédet". Meg fog lepődni, milyen gyorsan előállnak majd használható technikai jellegű feladatokkal.

Ha valaki gyakran kényszeríti ezen taktikák használatára, valószínűleg hasznos félrevonni az illetőt és komolyabb tréningben részesíteni. Ha a probléma ezután is fennáll, gondolkozzon el, hogy valóban
fontos-e a csapat számára.

Ha nem túl fontos, akkor próbálja eltávolítani a csapatból.

Ha viszont fontos, megpróbálhat kirendelni mellé egy "őrangyalt", akivel párban programozhat. Lehet, hogy Joe kitűnő fejlesztő és tervező, de szüksége van arra, hogy valaki megmondja neki, hogy min dolgozzon. Ez teljesen rendben van, kérje meg Niklast, hogy pesztrálja Joe-t. Esetleg saját maga is megpróbálkozhat ezzel. Ha Joe tényleg fontos tagja a csapatnak, megéri a fáradságot. Volt néhány ilyen szituációban részem és többé-kevésbé mindig működött.

9 Hogyan demonstrálunk
==========

A sprint demó (vagy "sprint áttekintés" ahogy mások hívják) a Scrum egy jelentős része, amit gyakran
alulbecsülnek.

"Ó, tényleg demózni kell? Nincs igazán sok, amit be lehetne mutatni."
"Nincs időnk erre a k&%$# demóra!"
"Nincs időm más csapatok demóira bemenni."

Miért ragaszkodunk, hogy minden sprint után legyen demó

A jól kivitelezett sprint demó, még ha nem is tűnik annak, drámai jelentőséggel bír.
* A csapat elismerésben részesül a munkájáért. Jól fogják érezni maguk.
* Más csapatok tudni fogják, hogy min dolgoznak.
* A demó után fontos visszajelzéseket kaphat.
* A demók közösségi események, ahol a csapatok érintkezhetnek egymással és beszélhetnek a munkájukról, ami értékes dolog.
* A demó arra kényszeríti a csapatot, hogy tényleg befejezze a feladatokat és, még ha csak a teszt környezetbe is, de telepítse azokat. Demók nélkül csak egy csomó 90%-ban befejezett dolgot kapnánk. A
demó hatására lehet, hogy kevesebb feladatot fejeznek be, de azok tényleg készen is lesznek, ami jobb, mint rengeteg, a következő sprintre is átnyúló, félkész dolog.

Amikor a csapatra rákényszerítik a demókat, akkor is ha nincs túl sok bemutatnivalójuk, a demó persze elég rosszul fog elsülni. A demó során rengeteg baki fog előfordulni és a végén kapott taps sem szívből szól. A kívülállók egy része sajnálni fogja a csapatot, míg egy másik része dühös lesz, hogy az idejét rabolják ilyen béna demókkal.

Ez elég fájdalmas tud lenni, de a hatása olyan, mint a keserű pirulának. A következő sprintben tényleg a legjobbat próbálják majd kihozni magukból, arra gondolva, hogy "lehet, hogy csak 2 dolgot tudunk majd bemutatni 5 helyett, de ezúttal tényleg MŰKÖDNI fog". A csapat tudni fogja, hogy nincs más választásuk, demózni kell, ami erősen növeli majd annak esélyét, hogy valami használható dolgot
mutassanak be. Ezt magam is többször tapasztaltam.

Sprint demó jó tanácsok
---
* Világosan ismertesse a sprint célját, illetve ha a jelenlevők között vannak olyanok, akik nem ismerik a terméket, pár percet szánjon annak bemutatására is.
* Ne töltsön túl sok időt a felkészüléssel, főképp ne előadás-fóliákra. Összpontosítson a működő kód bemutatására és ne a flancolásra.
* Diktáljon gyors tempót, az esztétika helyett inkább erre fordítsa a figyelmét az előkészületek során is.
* Üzleti és ne technikai szemszögből próbáljon demonstrálni. Azt hangsúlyozza, hogy mit és ne azt, hogy hogyan csinálták.
* Ha lehet, hagyja a hallgatóságot is kipróbálni a terméket.
* Ne demonstráljon hibajavításokat vagy triviális dolgokat. Meg lehet említeni, de nem szükséges bemutatni őket, mivel ez általában túl sok időt vesz igénybe és elvonja a figyelmet a fontos

sztorikról.

Mit tegyünk a "bemutathatatlan" dolgokkal

Csapattag: "Nem fogom bemutatni ezt a sztorit, mert ezt nem lehet bemutatni. A sztori az, hogy 'Skálázhatóság javítása 10000 konkurens felhasználóig'. Csak nem fogok behívni 10000 felhasználót a
demóra."
Scrum Master: "Végeztél ezzel a feladattal?"
Csapattag: "Persze."
Scrum Master: "Honnan tudod?"
Csapattag: "Telepítettem a rendszert a teljesítmény-tesztelő környezetre és ráküldtem egy csomó egyidejű kérést."
Scrum Master: "Rendben, van valami bizonyítékod, hogy a rendszer képes fogadni 10000 felhasználót?"
Csapattag: "Igen. Bár a teszt szerverek elég gyengék, mégis 50000 konkurens kérést ki tudtak szolgálni."
Scrum Master: "Honnan tudod?"
Csapattag: (idegesen) "Nos, itt a jegyzőkönyv! Láthatod te is hogy hogyan állítottam be a tesztet és hány kérést küldtem."
Scrum Master: "Ó, remek! Íme a demód. Mutasd meg ezt a jegyzőkönyvet és magyarázd el a hallgatóságnak. Több a semminél, nemde?"
Csapattag: "Ja, hogy ez elég? De ez elég ronda, kicsit szépíteni kellene rajta."
Scrum Master: "Ok, de ne pazarolj rá túl sok időt. Nem kell hogy szép legyen, csak a lényeg legyen rajta."

10 Hogyan bonyolítjuk le a sprint visszatekintőket
==========

Miért ragaszkodunk, hogy minden csapat végezzen visszatekintést
---

A legfőbb dolog a visszatekintésekkel kapcsolatban az, hogy tényleg megtartsuk őket.

Valamilyen megmagyarázhatatlan okból a csapatok nem mindig érzik fontosnak a lebonyolításukat. Laza felügyelet nélkül a legtöbben inkább kihagynák ezt és kezdenék már a következő sprintet. Persze elképzelhető, hogy ez csak a svéd kultúrára jellemző.

Abban azonban mindenki egyetérteni látszik, hogy a visszatekintők rendkívül hasznosak. Én továbbmennék és azt állítom, hogy a visszatekintő a Scrum második legfontosabb eleme (a sprint planning meeting után), mivel ez a legjobb lehetőség a dolgok javítására.

Persze visszatekintő nélkül is előállhat az ember jó ötletekkel, akár otthon a fürdőszobában is utolérhet minket az ihlet, de vajon ezeket az ötleteket a csapat is elfogadja majd? Elképzelhető, de annak a valószínűsége, hogy tényleg egyetértenek majd sokkal nagyobb, ha az ötlet "magától a csapattól származik", azaz a visszatekintő során vetődik fel, amikor bárki beleszólhat a vitákba.

A visszatekintők nélkül ráadásul a csapat nagy valószínűséggel ismételten el fogja követni ugyanazokat a hibákat.

Hogyan szervezzük a visszatekintőket

Az formátum gyakran változik, de nagyjából az alábbiakhoz hasonló:
* 1 - 3 órát különítünk el, attól függően, hogy mennyi vitára számítunk.
* Jelenlevők: Product Owner, az egész csapat és jómagam.
* Elvonulunk egy külön helyiségbe, egy kényelmes kanapéhoz, a tetőkertbe, vagy valami hasonló helyre, ahol zavartalanul beszélgethetünk.
* Nem szoktuk a csapatszobában tartani a visszatekintőket, mert ott túl sok a zavaró tényező.
* Valakit kijelölünk titkárnak.
* A Scrum Master bemutatja a sprint backlogot és a csapat segítségével összefoglalja a sprintet, illetve a főbb eseményeket és döntéseket.
* Körbe megyünk, mindenki lehetőséget kap elmondani, hogy mit tartottak jónak, min lehetett volna javítani és mit szeretnének másképp csinálni a következő sprint során.
* Összevetjük a becsült és valós sebességet és ha nagy eltérések adódnak, akkor megpróbáljuk megérteni az okát.
* Amikor a kijelölt idő a vége felé közeledik, a Scrum Master összefoglalja a konkrét javaslatokat.

A visszatekintőink nem túl szervezettek, de a mögöttes téma mindig ugyanaz: "min tudunk javítani a következő sprintben".

Íme egy példa egy nem túl régi visszatekintőkből:

<figure>

Három oszlop:
* Jó dolgok: Ha a sprintet újrakezdhetnénk, ezeket pontosan ugyanúgy csinálnánk
* Lehetett volna jobban: Ha a sprintet újrakezdhetnék, ezeket máshogy csinálnánk
* Javaslatok: Konkrét javaslatok a jövőre

Az első és második oszlop tehát a múltra vonatkozik, míg a harmadik a jövőre.

Miután a csapat előállt ezekkel a cetlikkel, pöttyökkel szavaztak arról, hogy melyik elemekre összpontosítsanak a következő sprint alatt. Minden tag három mágneses pöttyöt kapott, amelyet a szerintük legfontosabb javaslatokra oszthattak el tetszésük szerint. Ha akarták, mindhárom szavazatukat ugyanarra a javaslatra is leadhatták.

Ezek alapján öt javaslatot választottak ki a következő sprintre, aminek eredményét aztán a következő visszatekintőben meg fogjuk vizsgálni.

Fontos, hogy ne akarjunk mindent egyszerre csinálni, hanem sprintenként csak néhány javaslatra összpontosítsunk.

Tanulságok megosztása a csapatok között

A visszatekintők során előkerülő információk rendkívüli értékkel bírnak. Vajon azért nem tud a csapat a célra összpontosítani, mert az egyik sales menedzser állandóan berángatja a programozókat a
sales meetingekre "szakértőknek"? Ez egy fontos észrevétel, de elképzelhető, hogy más csapatok is ugyanezzel a problémával küszködnek. Vajon érdemes lenne-e jobban felkészíteni a termékmenedzsereket, hogy akár ők is támogathassák a sales menedzsereket?

A sprint visszatekintő valójában nem csak arról szól, hogy ez a konkrét csapat hogyan végezheti hatékonyabban a munkáját a következő sprint során, hanem sokkal mélyebb jelentőséggel is bír.

Mi ezt a következőképp kezeljük. Egy kijelölt személy (esetünkben ez én vagyok) minden visszatekintőre beül és információs hídként viselkedik.

Egy másik megoldás az lehet, hogy minden csapat beszámolót készít a visszatekintőről. Ezt mi is kipróbáltuk, de az a tapasztalat, hogy nem sokan olvassák ezeket a beszámolókat és még kevesebben
váltják ezt tényleges tevékenységekre, szóval inkább az egyszerűbb megoldást választottuk.

Fontos szabályok az "Információs Híd" számára:
* Meg kell hallgatnia az embereket
* Ha a visszatekintő túl csendes, jól irányzott kérdésekkel kell tudnia felélénkíteni a beszélgetést. Például: "Ha visszamehetnél az időben és az elejéről kezdhetnéd a sprintet, mit csinálnál másképp?"
* Hajlandónak kell lennie beülni az összes visszatekintőre
* Elég tekintéllyel kell rendelkeznie, hogy a csapat hatáskörén kívül eső javaslatokra ténylegesen és tevőlegesen reagálhasson.

Ez nálunk jól működik, de ha olyan stratégiát talál, ami ennél sokkal jobban működik, nyugodtan ossza meg velem.

Változtatni vagy nem változtatni

Tegyük fel, hogy a csapat egyetért abban, hogy "nem egyeztettünk eleget a csapaton belül, ezért folyton zavartuk egymás kódját."

Mit tegyünk ez ellen? Vezessünk be napi megbeszéléseket, esetleg telepítsünk új eszközöket a kommunikáció hatékonyságának javítására? Talán írjunk több wiki cikket? Esetleg. De lehet az is, hogy
ezek nem segítenek.

Az a tapasztalatunk, hogy gyakran elég ha felismerjük a problémát és az a következő sprintre önmagától megoldódik. Kiváltképp, ha a sprint visszatekintőt kitűzi a csapatszoba falára (amit mi szégyenletes módon mindig elfelejtünk). Minden apró változtatás ugyanakkor valamilyen költséggel jár, ezért első közelítésben érdemes nem tenni semmit, annak reményében, hogy a probléma magától
eltűnik (vagy legalábbis kisebbé válik).

A fenti "nem egyeztettünk eleget..." tipikus példa arra az esetre, amikor a probléma legjobb esetben éppen magától tűnik el.

Ha minden kis apró-cseprő probléma hatására változásokat eszközöl, szinte biztos, hogy valaki előbb tiltakozni fog majd lassanként inkább nem hozakodnak elő a problémákkal, ami rendkívül káros hatással tud lenni.

Az alábbiakban álljon néhány példa a visszatekintőben felmerülő problémákra és a rájuk adott tipikus válaszokra.

"Több időt kellett volna fordítanunk a sztorik feladatokra bontására"

Ez egy meglehetősen gyakori probléma. A napi scrum során a csapattagok gyakran "nem tudják, hogy mit
csináljanak az adott napon", ezért a meeting után sok idő megy el a konkrét feladatok keresésével, amit sokkal hatékonyabb lenne előre megcsinálni.

Tipikus válasz: semmi. A csapat valószínűleg megoldja ezt magától is a következő sprint planning meetingen. Ha mégis többször előfordulna, javasoljon hosszabb időkeretet a sprint planningre.

"Túl sok külső zavarforrás"

Tipikus válaszok:
* javasolja, hogy a csapat csökkentse a koncentrációs tényezőjét a következő sprint idejére
* javasolja, hogy a csapat jegyezze fel ezeket a zavarokat. Ki zavarta meg őket és mennyi idejüket vette igénybe? Ez a későbbiekben segíthet a probléma megoldásában.
* kérje meg a csapattagokat, hogy minden zavaró tényezőt irányítsanak a Scrum Master vagy a Product
  Owner felé.
* kérje meg a csapatot, hogy jelöljön ki egy "védőt", aki egy személyben foglalkozik az összes zavaró tényezővel, hogy a csapat többi tagja a saját feladataira koncentrálhasson. Ez lehet a Scrum Master, vagy egy állandóan változó személy.

"Túl sokat vállaltunk és ezért csak a feladatok felét sikerült befejezni"

Tipikus válaszok: semmi. A csapat valószínűleg nem fog túl sokat vállalni a következő sprintben. Legalábbis nem ennyivel.

"Mindig káosz van az irodában"

Tipikus válaszok:
* próbáljon jobb környezetet kialakítani, vagy költöztesse a csapatot az irodán kívül, béreljen például egy hotelszobát. Lásd: "Hogyan alakítjuk ki a csapatszobát"
* ha ezek nem lehetségesek, javasolja, hogy a csapat a káoszra hivatkozva csökkentse a koncentrációs
  tényezőjét a következő sprint idejére. Remélhetőleg ez elegendő lesz ahhoz, hogy a Product Owner beszéljen a felső vezetéssel.

Szerencsére nekem még sosem kellett azzal fenyegetőznöm, hogy a csapatot az irodán kívül költöztetem, de ha muszáj, nem fogok habozni. :)

11 Pihenőidő a sprintek között

A való életben nem lehet folyamatosan sprintelni, szükség van néha pihenni két sprint között. Ha állandóan sprintel, akkor valójában legfeljebb kocog.

Ugyanez igaz a Scrum-ra és nagy általánosságban a szoftverfejlesztésre is. A sprintek elég kimerítőek tudnak lenni. A fejlesztők nem igazán kapnak pihenőt, mert minden áldott nap ott kell állniuk azon a nyavalyás napi scrumon és elmondani, hogy mit csináltak az előző nap. Elég kevesen fogják azt mondani, hogy "tegnap a nap nagy részében a lábamat lógattam és cappuccinózás közben
blogokat olvasgattam".

A pihenésen túl van még egy jó ok arra, hogy szüneteket iktasson a sprintek közé: a sprint demó és visszatekintő után a csapat és a Product Owner egyaránt rengeteg megemésztendő információval fog
rendelkezni. Nagy a veszélye, hogy ha rögtön nekifutásból tervezni kezdik a következő sprintet, nem lesz elég idejük ezeket az információkat feldolgozni és levonni a megfelelő tanulságokat, és ezért például a Product Owner nem fogja tudni rendesen felmérni a feladatok fontosságát.

Rossz megoldás:

<figure>

A sprintkezdés előtt (egész pontosan a sprint visszatekintő és a következő sprint planning meeting között) elég időt próbálunk hagyni pihenésre, ami persze nem mindig sikerül.

Legalább annyit megpróbálunk elérni, hogy a sprint visszatekintő és a sprint planning meeting ne essen egy napra. Mindenkinek legalább egy kipihent éjszakát próbálunk biztosítani a következő sprint
előtt.

Jobb megoldás:

<figure>

Még jobb megoldás:

<figure>

Egy érdekes módszer a "20%" (vagy ahogy hívni szeretné) bevezetése. Ezek olyan napok, ahol a fejlesztők gyakorlatilag azon dolgozhatnak, amin csak akarnak (igen, bevallom, a Google ihlette). Például utánaolvashatnak a legújabb eszközöknek és API-knak, felkészülhetnek egy vizsgára, kocka dolgokról beszélhetnek a kollégákkal vagy esetleg egy hobbi projekten dolgozhatnak.

A célunk az, hogy mindig legyen egy "20%" nap két sprint között, ami biztosítja az megfelelő kipihentséget ráadásul olyan csapatot eredményez, amely naprakész tudással rendelkezik. Továbbá vonzó "kiegészítő juttatás" tud lenni.

A legjobb?

<figure>

Jelenleg havonta egyszer, a hónap első péntekén tartunk "20%" napot. Azért nem két sprint között, mert úgy éreztem, jobb, ha az egész vállalatban egy időben tartjuk, mert így tényleg komolyan is veszik az emberek. Mivel a sprintjeink nincsenek szinkronizálva az egyes termékek között, kénytelen voltam egy sprint-től független "20%" napot bevezetni.

Lehet, hogy egy nap majd összehangoljuk a sprinteket (azaz hogy az összes termék és csapat sprintje
azonos napon kezdődik és végződik), amely esetben a "20%" napot biztos, hogy a sprintek közé fogom
beiktatni.

12 Hogyan tervezzük a kiadásokat és mit teszünk fixáras szerződések esetén
==========
Néha több sprintre előre kell terveznünk, kiváltképp, ha egy fixáras szerződésünk van, amelyben ha nem szeretnénk azt kockáztatni, hogy nem végzünk időben, muszáj előre terveznünk.

A kiadások tervezése általában arra a kérdésre ad választ, hogy "mi az a legkésőbbi időpont, amelyre
az új rendszer 1.0 változata elkészül".

Ha behatóbban szeretné tanulmányozni a kiadás-tervezést, akkor azt javaslom, hogy ugorja át ezt a fejezetet és helyette olvassa el Mike Cohn "Agilis Becslés és Tervezés" című könyvét. Magam is azt kívánom, bárcsak előbb tettem volna (én is csak azután olvastam, miután már egyedül rájöttem a dolgokra...). Az én verzióm eléggé leegyszerűsített, de éppen ezért jó kiindulópontot adhat.

Az elfogadás feltételeinek meghatározása

A szokásos Product Backlog-on kívül a Product Owner definiálja az elfogadás feltételeit, ami tulajdonképpen azt határozza meg, hogy a product backlog fontossági szintjei mit jelentenek a szerződés szempontjából.

Íme egy példa erre:
* A 100-nál magasabb fontossági szintű elemeket be kell fejezni az 1.0 verzióig, mert ha nem, a bírságok csődbe viszik a céget.
* Az 50 és 99 közötti fontosságú elemeket be kell venni az 1.0 verzióba, de elképzelhető, hogy ezek kiadhatóak egy közvetlenül az 1.0 utáni frissítésben is.
* A 25 és 49 közötti elemek szükségesek, de elég az 1.1 verzióra befejezni őket.
* A 25-nél kisebb elemek nincsenek lefixálva és lehet, hogy nem is kell majd implementálni őket.

Íme továbbá egy példa a fenti szabályok alapján színkódolt product backlog-ra.

<figure>

Piros = mindenképpen benne vannak az 1.0-ban (banán - körte)
Sárga = ha lehet, bele kell venni az 1.0-ba (mazsola - hagyma)
Zöld = későbbre lehet halasztani (grapefruit - barack)

Tehát ha a banántól a hagymáig mindent sikerül befejezni a határidőre, minden rendben. Ha kifutni látszunk az időből, kihagyhatjuk a mazsolát, mogyorót, fánkot vagy hagymát. A hagyma utáni feladatok
pedig opcionálisak.

A legfontosabb feladatok becslése

A release planninghez a Product Ownernek szüksége van legalább azoknak az elemeknek a becsléseire,
amik a szerződésbe lettek foglalva. Ahogy a sprint planning is, ez is egy csapattevékenység, azaz a a csapat végzi a becsléseket, míg a Product Owner szolgáltatja a feladatok részleteit és válaszol a felmerülő kérdésekre.

A becsült idő hasznos dolog, ha jól fedi a valóságot, kevésbé hasznos, ha utólag kiderül, hogy mondjuk 30%-kal eltér a valóságtól és teljesen értéktelen, ha nincs is köszönőviszonyban a valósággal.

Az alábbi ábrával szeretném érzékeltetni, hogy szerintem milyen kapcsolat van a becslések és az azt végző személyek, illetve a ráfordított idő között.

<figure>

Végső soron ezzel azt szerettem volna hangsúlyozni, hogy:
* Hagyja a csapatot becsülni.
* Ne töltsenek túl sok időt vele.
* Győződjön meg arról, hogy a csapat megértette, hogy ezek csak közelítések és nem jelentenek kötelezettséget.

Általában a Product Owner összehívja a csapatot egy szobába, biztosít frissítőket, majd elmagyarázza, hogy a meeting célja, hogy a legfontosabb 20 (vagy akármennyi) sztorit megbecsüljék.
Ezután egyesével végigmegy a sztorikon és hagyja a csapatot dolgozni. Továbbra is a szobában marad
azonban, hogy válaszolhasson a felmerülő kérdésekre. Fontos megjegyezni, hogy ahogy a sprint planning meetingen is, itt is rendkívül hasznos kitölteni a "demó módja" mezőt, az esetleges félreértéseket elkerülendő.

A meeting idejét szigorúan meg kell szabni, mert egyébként a csapat túl sok időt tölt néhány sztori

becslésével. Ha a Product Owner úgy találja, hogy több időre van szükség, természetesen összehívhat
egy újabb meetinget később. A csapat kötelessége emlékeztetni a Product Ownert, hogy a meeting kihatással van az éppen futó sprintre, vagyis nem ingyen van.

Íme egy példa az becslések eredményeire (sztori pontban számítva):

<figure>

Becsült sebesség

Rendben, most hogy már van valami fogalmunk a legfontosabb sztorikra szükséges időről, a következő
lépés, hogy a sprintek átlagos sebességét megbecsüljük, ami azt jelenti, hogy meg kell egyezni a koncentrációs tényezőben is. (Lásd: "Hogyan dönti el a csapat, hogy mely sztorik kerüljenek a sprintbe?"

A koncentrációs tényező gyakorlatilag azt határozza meg, hogy "a csapat mennyi időt tud a jelenleg kiválasztott feladatokra fordítani". Ez sosem 100%, mivel a csapatok időt vesztenek az előre nem tervezett feladatokkal, a feladatok közötti váltásokkal, más csapatok segítésével, e-mail olvasással, az elromlott számítógépük megjavításával, politikai viták folytatásával a konyhában, stb...

Tegyük fel, hogy a csapat koncentrációs tényezőjét 50%-ban állapítottuk meg (ami elég alacsony, mi általában 70% környékén szoktunk mozogni) hogy a sprint három hétig (15 nap) tartani és hogy a csapat 6 főből áll.

Egy sprint tehát 90 ember-nap, azonban a tényleges sztorikra fordítható idő 45 ember nap az 50%-os koncentrációs tényező miatt. Ebből tehát a becsült sebességünk 45-re adódik.

Ha minden sztorit 5 napra becsültük (ami általában persze nem igaz), akkor ez a csapat sprintenként nagyjából 9 sztorit tud befejezni.

A kiadási terv összeállítása

Most, hogy megvannak a becsléseink és a sebességünk (45), a product backlogot feloszthatjuk sprintekre:

<figure>

Minden sprint annyi sztorit tartalmaz, ami még belefér a becsült 45 pontos sebességbe.

Ez azt jelenti, hogy valószínűleg 3 sprintre lesz szükségünk a "mindenképpen" és a "lehetőleg" sztorik befejezéséhez.

3 sprint = 9 naptári hét = 2 naptári hónap. Ez összeegyeztethető a megrendelőnek ígért határidővel? Minden persze a szerződés jellegétől függ, azaz hogy mennyire rögzítettek a feladatok. Ezen felül mi

általában hozzá szoktunk még adni egy jelentős puffert, hogy csökkentsük a pontatlan becslések, előre nem látott problémák, előre nem tervezett feladatok, stb. hatásait, így például ebben az esetben mondjuk három hónapban egyeznénk meg, ami nagyjából egy hónapnyi "vésztartalékot" biztosít.

Az igazán jó dolog az, hogy három hetente be tudunk mutatni a megrendelőnek egy működőképes dolgot,
ami alapján menet közben is változtathat a követelményeken (a szerződéstől függően persze)

A kiadási terv frissítése

A valóság nem szokott a tervekhez igazodni, ezért a terveket kell a valósághoz igazítanunk.

Minden sprint után kiszámítjuk a valós sebességünket az adott sprintre. Ha ez radikális mértékben eltér a becsülttől, akkor a jövőbeni sprintek várható sebességét is módosítjuk, ami a kiadási terv változásával jár. Ez azonban problémát jelent, így a Product Owner megpróbálhat egyeztetni a megrendelővel, vagy esetleg megpróbálhatja valamely feladat méretét a szerződés keretein belül lecsökkenteni. További lehetőség, hogy a csapattal közösen megoldást próbálnak találni a sebesség növelésére vagy a zavaró tényezők kiiktatásával a koncentrációs tényező növelésére.

A Product Owner felhívhatja a megrendelőt: "Helló, egy kicsit kiszaladni látszunk az időből, de be tudjuk fejezni határidőre, ha kivesszük ezt a "beépített Pacman" funkciót-t, ami túl sok időt vesz el. Ezt kiadhatjuk három héttel később a következő verzióban."

A megrendelő nem fog ennek örülni, de legalább becsületesen bevalljuk és hagyunk időt, hogy a megrendelő eldöntse, hogy a legfontosabb szolgáltatásokat kiadjuk időben, vagy minden csússzon. Általában nem nehéz a döntés :)

13 Hogyan egyesítjük a Scrum-ot az XP-vel
==========
Nem túlzás azt állítani, hogy a Scrum és XP (eXtrém Programozás) remekül egyesíthető, az interneten
található cikkel is mind erre utalnak, ezért én nem fogom azzal húzni az időt, hogy elmondjam miért.

Azonban egy dolgot hadd említsek meg: a Scrum a menedzsmentre és szervezési gyakorlatra koncentrál,
míg az XP szinte kizárólag a programozásra. Épp amiatt működnek olyan jól együtt, mert remekül kiegészítik egymást.

Tehát itt szeretnék ünnepélyesen csatlakozni azok táborához, akik szerint a Scrum és XP kiválóan egyesíthetőek.

A továbbiakban kiemelem majd a főbb XP előírásokat, illetve kitérek arra, hogy ezek a napi gyakorlatban hogyan hasznosíthatóak. Nem minden csapatunk alkalmazta ezeket a gyakorlatokat, de
összességében azért a Scrum és XP szinte minden kombinációját kipróbáltuk. Az XP néhány előírása

közvetlenül megjelenik a Scrum-ban is, míg mások esetében vannak legalább átfedések. Példának okáért
a "Teljes Csapat", "Együtt Ülés", "Sztorik" vagy a "Tervezési Játék". Ezekben az esetekben egyszerűen a Scrum-ra hagyatkoztunk.

Páros Programozás

Ezt nemrég kezdtük el az egyik csapatban és ami azt illeti, elég jól bevált. A legtöbb csapat nem használja túl gyakran, de most, hogy egy pár sprint erejéig kipróbálhattam, valószínűleg több csapatban is be fogjuk vezetni.

Néhány megjegyzés a páros programozással kapcsolatban:
* A páros programozás tényleg javít a kód minőségén.
* A páros programozás tényleg javítja a csapat összpontosítását (például amikor a hátad mögül azt hallod: "hé, ez tényleg kell ebben a sprintben?")
* A páros programozástól ódzkodók között meglepően sokan vannak, akik még igazából sosem próbálták ki de valójában gyorsan hozzászoknak és megszeretik.
* A páros programozás rendkívül kimerítő tevékenység, amit nem szabad egész nap csinálni.
* A párok gyakori cseréje jó hatással van.
* A páros programozás valóban segít a tudás gyors megosztásában.
* Néhányan nem szeretik a páros programozást. Csak ezért azonban ne mondjon le egy egyébként kiváló programozóról.
* A code review jó alternatívája a páros programozásnak.
* A "navigátornak" (aki nem programozik) szintén legyen saját számítógépe. Nem fejlesztésre, hanem például a dokumentáció böngészésére amikor a "pilóta" (a programozó) elakad.
* Ne kényszerítse a páros programozást az emberekre, csak ösztönözze őket és tegye elérhetővé a megfelelő eszközöket, de hagyja, hogy a saját tempójukban kísérletezzenek a páros programozással.

Teszt-vezérelt fejlesztés (TDD)

Ámen! Ez számomra sokkal fontosabb, mint a Scrum és XP együttvéve. Megfoszthatnak a házamtól, elvehetik a tévém, elvihetik a kutyám, de a TDD-t nem adom! Ha ki nem állhatja a TDD-t, akkor jobb, ha nem enged az épületbe, mert így vagy úgy, de megpróbálom majd becsempészni. :)

Íme egy 10 másodperces áttekintő:

> A teszt-vezérelt fejlesztésben legelőször is írunk egy automatizált tesztet, majd elég kódot, hogy ez a teszt sikeresen lefusson, ezután a kód olvashatóságának javítása és a duplikációk eltávolítása végett refaktorálásra kerül. Majd újra az elejétől.

Néhány észrevétel a TDD-vel kapcsolatban:
* A TDD nehéz és eltart egy ideig, amíg valaki valóban megérti. Ami azt illeti, néha nem számít, hogy mennyi időt tölt tréninggel, oktatással és demonstrációkkal - a legtöbb esetben az

egyetlen mód a TDD elsajátítására, ha egy TDD-t már jól ismerő programozóval párba állunk. Viszont
ha egyszer valaki tényleg megérti, nem nagyon akar majd visszamenni a TDD előtti programozáshoz.
* A TDD alapjaiban javítja a rendszer felépítését.
* Egy új rendszer esetében eltart egy ideig a TDD bevezetése, főként a fekete doboz alapú integrációs
tesztek esetében, de mindez nagyon hamar megtérül.
* Fektessen elég időt a könnyen írható tesztekbe. Ez azt jelenti, hogy használja a megfelelő eszközöket, biztosítson megfelelő oktatást mindenkinek, használják a megfelelő kiegészítő- és alap osztálykönyvtárakat, stb...

Mi a következő eszközöket használjuk a teszt-vezérelt fejlesztés során:
* jUnit / httpUnit / jWebUnit. Fontolgatjuk továbbá a TestNG és Selenium használatát.
* HSQLDB, egy beépített, memória-rezidens adatbázis.
* Jetty, egy beépített, memória-rezidens webtároló.
* Cobertura a teszt-lefedettségi mértékekhez.
* Spring keretrendszer a különböző teszt fixtúrák felépítéséhez (mock-kal, mock nélkül, külső adatbázissal, memória-rezidens adatbázissal, stb.)

A - TDD szempontjából - legbonyolultabb termékeinkben automatizált fekete doboz-alapú elfogadási teszteket alkalmazunk. Ezek a teljes rendszert adatbázissal és webszerverekkel együtt memóriába töltik és kizárólag a nyilvános interfészeket (mint például HTTP) használva tesztelik.

Ez rendkívül gyors fejlesztés-telepítés-teszt ciklust eredményez, ami védőhálóként is szolgál, ezzel adva önbizalmat a gyakori refaktoringhoz, ami cserében azt jelenti, hogy az architektúra a rendszer növekedésével sem lesz rosszabb.

TDD új kód esetén

Mi minden új fejlesztésnél TDD-t használunk még akkor is, ha ez azt jelenti, hogy a projekt előkészítése több időt vesz igénybe (mivel több eszközre és teszt keretrendszerre van szükségünk). A megtérülés olyan jelentős, hogy meg sem fordul a fejünkben nem alkalmazni a TDD-t.

TDD meglévő kód esetén

A TDD önmagában is nehéz, de egy olyan kódbázison, amely a kezdetektől nem használt TDD-t... nos, az
embertelenül nehéz. Több oldalt tudnék írni arról, hogy ez miért is van így, de inkább itt megállok és az értekezést a "TDD a frontvonalról" című következő írásomra hagyom. :)

Elég sok időt fordítottunk az integrációs tesztek automatizálására az egyik bonyolultabb, régóta meglevő és eléggé rossz állapotban levő, teszteket hallomásból sem ismerő rendszerben.

Eredetileg a rendszer minden egyes kiadásához teszterek egy csapatát kellett alkalmaznunk, akik - jórészt kézzel - végrehajtottak egy csomó regressziós és teljesítménytesztet, ami jelentősen lassította a fejlesztési és kiadási ciklusainkat. A célunk ezen tesztek automatizálása volt, ám jó

pár hónap fejtörés után sem kerültünk közelebb ehhez a célhoz.

Ezután stratégiát váltottunk. Beletörődtünk, hogy a regressziós teszteket kézzel kell végrehajtani, ehelyett inkább arra a kérdésre próbáltunk összpontosítani, hogy "hogyan csökkenthetjük az ezekhez a
tesztekhez szükséges időt?". Ez egy játékkonzol volt és arra kellett rádöbbennünk, hogy a teszterek idejének java része triviális előkészítésekkel telik, mint például az adminisztrációs felületen tournamentek létrehozása vagy éppen az időzített tournamentek kezdetére várás. Így hát ehhez kezdtünk segédeszközöket készíteni, olyan apró, hozzáférhető shortcutokat és szkripteket, amelyek elvégzik a munka dandárját és hagyják a teszterekeket a tényleges tesztekre koncentrálni.

Ez elég jól működött. Valószínűleg a kezdetektől fogva ezt kellett volna tennünk, de annyira arra összpontosítottunk, hogy a tesztek automatizálva legyenek, hogy elfelejtettük ezt lépésről-lépésre csinálni, ahol az első lépés az lett volna, hogy a manuális teszteket hatékonyabbá tevő segédeszközöket készítünk.

A tanulság: ha kénytelen kézzel végezni a regressziós teszteket, de szeretné azokat automatizálni, inkább ne tegye (hacsak nem nagyon könnyű). Ehelyett inkább készítsen olyan segédeszközöket, amelyek
leegyszerűsítik a kézi regressziós tesztek futtatását. Később elgondolkodhat a tesztek tényleges automatizálásán.

Inkrementális tervezés

Ez azt jelenti, hogy a kezdetektől az egyszerűségre törekedünk, amit lépésenként javítunk és nem arra, hogy már az elején minden tökéletes legyen, amit aztán befagyasztunk.

Ezzel elég jól haladunk, azaz időnk java részét refaktorálással és az eddigi architektúra fejlesztésével töltjük és viszonylag ritkán tervezünk meg mindent előre, Bár néha elkövetünk olyan hibákat, mint például hogy egy gyenge elemet olyan sokáig tűrünk meg a rendszerben, hogy a módosítása magában egy nagyobb projektté válik. Összességében azonban nagyon elégedettek vagyunk.

Az architektúra folyamatos javulása egyébként nagyrészt a TDD egy automatikus mellékhatása.

Folyamatos integráció (CI)

Termékeink jelentős részében egy Maven-re és QuickBuild-re épülő fejlett folyamatos integrációs rendszert használunk, ami rendkívül hasznosnak bizonyult. Tökéletes megoldás az olyan olyan problémákra, mint a "de az én gépemen fut". A CI szerverünk játssza a "bíró" vagy referenciapont szerepét a rendszer állapotának felmérésében. Amint valaki feltölt valamit a verziókövető rendszerbe, a CI szerver feléled, nulláról elkészíti a buildet egy üres szerveren és lefuttatja az összes tesztet. Ha valamilyen probléma adódik, e-mailt küld a csapat összes tagjának a meghiúsult build-ről, illetve hogy pontosan melyik változtatás eredményeként hiúsult meg továbbá linkeket a tesztek eredményeihez stb.

A CI szerver minden éjjel nulláról felépíti a terméket és feltölti a binárisokat (ear, war, stb),

a dokumentációt, teszt eredményeket, a teszt lefedettségi kimutatást, függőségi dokumentációt a belső dokumentációs portálunkra. Bizonyos termékeket feltelepít a teszt környezetre is.

Az egész rendszer felállításában rengeteg munka van, de minden egyes perce megérte.

A kód a csapat tulajdona

Ugyan még nem minden csapatunk alkalmazza ezt az elvet, támogatjuk azt a nézetet, hogy a kód a csapat egészének tulajdona. Tapasztalatunk szerint a gyakran váltott párokkal való páros programozás
nagymértékben hozzájárul ennek kialakulásához. Azok a csapatok, akik közösen birtokolják a kódot, rendkívül hatékonynak bizonyulnak, mivel a sprintjeik nem omlanak össze ha például egy fontos tag megbetegszik.

Áttekinthető munkatér

Minden csapatunk rendelkezésére állnak táblák és üres falfelületek és ezt általában jól ki is használják. A legtöbb irodánkban a falakat a termékről és a projektről szóló különböző papírok borítják. A legnagyobb probléma, hogy régi, már elavult információk gyűlnek össze a falon, amire néha egy "házvezetőt" jelölünk ki a csapatból.

Támogatjuk a feladat-térképek használatát is, bár még nem minden csapatunk használja azt. Lásd "Hogyan alakítjuk ki a csapatszobát".

Kódolási szabályok

Az utóbbi időben elkezdtük definiálni a jó kódra vonatkozó szabályokat, ami rendkívül hasznos és azt kívánom, bárcsak hamarabb tettük volna. Szinte semmibe nem kerül, elég pár egyszerű elemből kiindulni és idővel kibővíteni. Csak olyan dolgokat írjon le, ami egyébként nem triviális és ha lehet mindig kapcsolja az új szabályokat már létező dolgokhoz.

A legtöbb programozó saját kódolási stílussal rendelkezik. Olyan apró részletekre gondoljon, mint hogy hogyan kezelik a kivételeket, hogyan írják a megjegyzéseket, mikor adnak vissza null-t, stb. Az esetek egy részében ezek a különbségek nem számítanak, míg más esetekben inkonzisztens rendszerhez
és olvashatatlan kódhoz vezethetnek. A kódolási szabályok sokat segítenek, amennyiben ténylegesen a
lényeges dolgokra koncentrálódnak.

Az alábbiakban álljon néhány példa a mi kódolási szabályainkra:
* Bármely szabály megszeghető ha arra jó indok van és az dokumentálva van.
* Alapértelmezésben használja a Sun kódolási szabályait:
  http://java.sun.com/docs/codeconv/html/CodeConvTOC
* Soha, soha, de soha ne kezeljen kivételeket a stacktrace logolása vagy a kivétel újradobása nélkül. A log.debug() elfogadható, de ügyeljen a stacktrace megőrzésére.
* Használjon setter-alapú dependency injection-t (függőség injektálás) az osztályok szétválasztására (kivéve persze ha az összekapcsolás a cél).

* Kerülje a rövidítéseket. A jól ismert rövidítések, mint a DAO használata elfogadható.
* Azok a metódusok, amelyek tömböt, vagy collection-t adnak vissza, nem adhatnak vissza null-t. A null helyett használjon üres tömböt vagy collection-t.

Fenntartható iram / stimulált munka

Sok agilis fejlesztésről szóló könyvben írták már, hogy a huzamos ideig tartó túlórázás erősen csökkenti a hatékonyságot. Néhány akaratomon kívüli kísérlet után azt kell mondjam, hogy tökéletesen
egyetértek velük.

Úgy egy éve az egyik legnagyobb csapatunk embertelen mennyiségű túlórát dolgozott. A kód minősége
egyszerűen pocsék volt, ezért idejük legnagyobb részét tűzoltással töltötték. A teszterek (akik szintén túlóráztak) esélyük sem volt rendes minőségellenőrzést végezniük. A felhasználó magukon kívül voltak, míg a bulvárlapok rajtunk csámcsogtak.

Pár hónappal később sikerült a munkaidőt normális keretek közé visszaszorítani. Mindenki normális mennyiségű időt dolgozott (néhány kivételtől eltekintve) és meglepő módon a minőség és termelékenység érzékelhetően javultak.

Persze a munkaidő lecsökkentése nem az egyetlen tényező, ami közrejátszott, de meg vagyunk győződve,
hogy igen nagy szerepe volt.

14 Hogyan tesztelünk
==========
Ez a legnehezebb rész. Nem tudom, hogy csak a Scrum legnehezebb része, vagy úgy általában a szoftverfejlesztésé.

A tesztelés az a rész, amely vállalatról vállalatra a leginkább változik. Olyan tényezőktől függ,
mint a teszterek száma, az automatizálás aránya, a rendszer típusa (szerver és webalkalmazás vagy esetleg dobozos szoftver?), kiadási ciklusok hossza, a rendszer kritikussága (blog szerver vagy repülésirányítási rendszer), stb...

Elég sokat kísérletezgettünk a teszteléssel a Scrum kereteiben, a következőkben megpróbálom majd leírni, hogy mit próbáltunk ki és mi működött eddig.

Valószínűleg nem hagyhatja el az elfogadási teszteket

A tökéletes Scrum világában, a sprint eredménye egy telepíthető verzió, amit csak telepít és kész. Vagy nem?

<figure>

Hát nem.

Az a tapasztalatunk, hogy ez a gyakorlatban ritkán működik. Helyette csúnya bugokat fog találni, szóval ha Önnek egy kicsit is számít a minőség, nem ússza meg valamiféle kézi elfogadási tesztek nélkül. Ez az, amikor a csapaton kívüli teszterek nekiesnek a rendszernek olyan tesztekkel, amikre a Scrum csapat álmában se gondolt volna, vagy legalábbis nem volt ideje gondolni. A teszterek pontosan
úgy férnek hozzá a rendszerhez, mint a felhasználók, ami azt jelenti, hogy a teszteket manuálisan kell végezni (feltéve persze, hogy a rendszer emberek számára készül).

<figure>

A teszterek hibákat találnak majd, amit Scrum csapatnak kell kijavítani bugfix kiadásokban és előbb vagy utóbb (remélhetőleg előbb) a felhasználóknak is kiadható az 1.0.1-es hibajavított verzió.

Amikor "elfogadási teszt fázis"-ról beszélek, akkor arról a hosszabb tesztelés-debuggolás-kiadás ciklusról van szó, aminek a végén egy elfogadható minőségű ténylegesen kiadható rendszert kapunk.

Minimalizálja az elfogadási teszt fázist

Az elfogadási teszt fázis fájdalmas és nagyon nem tűnik agilisnak. Bár nem lehet teljesen megszabadulni tőle, megpróbálhatjuk minimalizálni, egész pontosan minimalizálni az elfogadási teszt fázisra fordított időt. Ezt a következő módon érjük el:
* A Scrum csapat által szállított kód minőségének maximalizálásával
* A kézi tesztek hatékonyságának növelésével (azaz alkalmazza a legjobb tesztereket, biztosítsa számukra a legjobb eszközöket és gondoskodjon arról, hogy jelezzék a leginkább időigényes feladatokat, amiket aztán automatizálhat).

Hogyan is maximalizálhatjuk a Scrum csapat által szállított kód minőségét? Nos erre számtalan mód van, de íme kettő ami nekünk elég jól működött:
* Teszterek hozzáadása a Scrum csapathoz
* Sprintenként kevesebb feladat vállalása

Minőség javítása teszterek hozzáadásával

<figure>

Igen, már hallom is az ellenérveket:
* "De ez egyértelmű! A Scrum csapat tagjai több téren is képzett kell legyenek!"
* "A Scrum csapat tagjainak nincs kijelölt szerepköre! Nincs olyan, hogy valaki csak teszter!"

Hadd pontosítsak. Ebben az esetben a "teszter" olyan személyt jelent, akinek az "elsődleges képzettsége tesztelés" és nem olyan személyt, akinek "az egyetlen feladata a tesztelés".

A fejlesztők gyakran elég rossz teszterek, főleg amikor a saját kódjukat kell tesztelni.

A teszter az "utolsó ellenőrzőpont"

Amellett, hogy a csapat egy "normális" tagja, a teszternek van egy fontos felelőssége, mégpedig hogy
az utolsó ellenőrzőpont legyen. Egy feladat addig nem tekinthető késznek, amíg ő azt nem mondja, hogy készen van. Az a tapasztalatom, hogy a fejlesztők gyakran mondanak olyan dolgokat is "kész"-nek, amik valójában nincsenek készen még akkor is, ha a "kész" definíciója világos (amiről egyébként érdemes gondoskodnia, Lásd: "A "kész" definíciója"), mert a fejlesztők erről gyakran megfeledkeznek. Mi programozók türelmetlen népség vagyunk, akik amint lehet, szeretnénk belekezdeni
a következő feladatba.

Honnan tudja tehát Mr. T (a teszterünk), hogy valami készen van? Először is - meglepő módon - tesztelnie kell. Sok esetben előfordul, hogy amit a fejlesztő "kész"-nek gondolt, azt nem lehet még tesztelni sem, mert nincs a verziókövetőben, nincs a teszt környezetre telepítve, nem fut, stb. Ha Mr T elvégezte a funkció tesztjét, a fejlesztővel közösen végigmennek a "kész" szabálylistán (már ha van ilyen). Például elképzelhető, hogy ez a lista előírja, hogy kiadási megjegyzést kell készíteni. Ilyenkor Mr T gondosan ellenőrzi, hogy ez megtörtént-e. Ha tartozik a funkcióhoz formális specifikáció (esetünkben ilyen nem nagyon van), akkor Mr T ezt is leellenőrzi.

Egy kellemes mellékhatás, hogy a csapatban így biztosan van legalább egy ember, aki tökéletes a sprint demó megszervezésére.

Mit csinál a teszter, amikor nincs mit tesztelni?

Elég gyakran felmerülő szituáció, hogy Mr T azt mondja a Scrum Master-nek, hogy épp nincs mit tesztelni, most mit tegyen. Akár egy hétig is eltarthat, amíg a csapat befejezi az első sztorit, mit tegyen ez alatt a teszter?

Először is fel kell készülnie a tesztekre, például teszt specifikációt írni vagy a teszt környezetet konfigurálni, hogy amikor a fejlesztő elkészül, ne kelljen várnia és Mr T azonnal elkezdhesse a teszteket.

Ha a csapat TDD-t használ, akkor a tagok eleve a kezdetektől fogva írnak teszteket. A teszter párba állhat az éppen teszteket író fejlesztőkkel, még akkor is, ha egyébként a teszter nem tud programozni (de ekkor csak navigátori szerepben). Egy jó teszter általában másféle tesztekkel áll elő, mint egy jó programozó, ezért remekül kiegészítik egymást.

Ha a csapat nem használ TDD-t, vagy nincs elég teszt, hogy kitöltse a teszter idejét, bármi mással is foglalkozhat, ami közelebb viszi a csapatot a sprint céljának eléréséhez - pontosan mint a csapat többi tagja. Ha a teszter tud programozni, nagyszerű. Ha nem, akkor a csapat összegyűjtheti számára
a programozást nem igénylő feladatokat.

Amikor a sztorikat feladatokra bontjuk a sprint planning meetingben, a csapat általában programozási feladatokra koncentrál, azonban rengeteg programozást nem igénylő feladat is van a sprintben. Ha a sprint planning során megpróbálja azonosítani ezeket a feladatokat, nagy valószínűség szerint Mr T is sokat fog tudni segíteni még ha nem is tud programozni és nincs éppen tesztelendő funkció.

Néhány példa gyakran felmerülő, programozást nem igénylő feladatokra:
* Teszt környezet telepítése.
* Követelmények tisztázása.
* A telepítés részleteinek egyeztetése az üzemeltetőkkel.
* Telepítési dokumentáció írása (kiadási megjegyzések, RFC-k, vagy amire éppen szükség van).
* Kapcsolattartás külsősökkel (például GUI dizájnerek).
* A build szkript fejlesztése.
* Sztorik lebontása feladatokra.
* A fejlesztők főbb kérdéseinek összegyűjtése és megválaszolása.

Mit tegyünk, ha épp ellenkezőleg, Mr T a szűk keresztmetszet? Mondjuk a sprint utolsó napján egy csomó dolog hirtelen elkészül és Mr T-nek esélye sincs mindent tesztelni. Ebben az esetben a csapat
minden tagját kinevezhetjük Mr T segédjének. Mr T dönti el, hogy mi az, amit magának kell megcsinálnia és a fennmaradó dolgokat feloszthatja a csapat többi tagja között. Végtére is ez a lényege a több téren is képzett csapatnak.

Tehát Mr T igenis rendelkezik külön szerepkörrel a csapaton belül, de ettől még ugyanúgy végezhet más feladatokat is, ugyanúgy, ahogy a csapat többi is nyugodtan tesztelhet.

A minőség javítása sprintenként kevesebb feladat vállalásával
---
Ez a sprint planning meeting-ig nyúl vissza. Leegyszerűsítve, ne próbáljon túl sok sztorit belezsúfolni a sprintbe. Ha gondja van a minőséggel vagy túl hosszú az elfogadási teszt fázisa, egyszerűen vállaljon kevesebbet sprintenként. Ez majdnem teljesen automatikusan jobb minőséghez,
rövidebb elfogadási teszt időszakhoz, kevesebb bughoz és hosszú távon jobb hatékonysághoz vezet,
mivel a csapat ideje java részét a régi dolgok állandó javítása helyett új funkciók fejlesztésével töltheti.

Szinte mindig kifizetődőbb kevesebb, de stabilabb funkciót fejleszteni, mint többet, amit viszont később rohamtempóban kell javítgatni.

A sprint része legyen-e az elfogadási teszt?

Itt elég sokat szoktunk vacillálni. Bár néhány csapatban az elfogadási teszt a sprint rész, a legtöbb esetén nem, méghozzá két okból:
* Egy sprint fix időkorláttal rendelkezik. Az elfogadási tesztet (ideértve a debuggolást és újra-kiadást) azonban rendkívül nehéz keretek közé szorítani. Mi legyen, ha kifutunk az időből és még mindig vannak kritikus hibák? Adjuk ki a rendszert hibákkal együtt, vagy várjunk a következő sprintig? A legtöbb esetben mindkét megoldás elfogadhatatlan, azért inkább kihagyjuk az elfogadási tesztet a sprintből.
* Ha egy terméken több Scrum csapat is dolgozik, az elfogadási tesztet az összes sprint eredményén kell végezni még akkor is, ha egyébként mindkét csapat sprintjében benne volt egy elfogadási teszt fázis.

<figure>

Ez nem tökéletes megoldás, de a legtöbb esetben megfelelőnek bizonyult.

Sprint ciklus és elfogadási teszt ciklus

Az ideális Scrum világban nincs szükség elfogadási tesztekre, hiszen minden Scrum csapat minden sprint végén gyakorlatilag kiadásra kész terméket készít.

<figure>

A valóság azonban kissé más:

<figure>

Az első sprint után egy hibás 1.0.0 kerül kiadásra. A második sprint alatt elkezdenek beérkezni a hibajelentések, ezért a csapat ideje nagy részét debuggolással tölti és arra kényszerül, hogy a sprint közepén kiadjon egy 1.0.1-es hibajavítást. A második sprint végén kiadnak egy új funkciókat tartalmazó 1.1.0-t, ami persze még az előzőnél is bugosabb, mivel kevesebb idejük volt helyesen működő szoftvert készíteniük az előző verzió folyamatos javításai miatt.

A második sprintben található piros vonalak jelképezik ezt a káoszt.

Elég rondán néz ki, nem? Az igazán szomorú dolog, hogy a problémát nem oldja meg egy kimondottan
elfogadási tesztekre szakosodott csapat sem, mert úgy legfeljebb nem a dühös felhasználóktól érkeznek a hibajelentések. Ez ugyan üzleti szempontból jelentős különbség, de a fejlesztők szemszögéből gyakorlatilag ugyanaz azzal a különbséggel, hogy ezek a teszterek általában kevésbé agresszívek, mint a felhasználók. Általában.

<figure>

Bár sok modellel kísérleteztünk, nincs igazán jó megoldásunk erre a problémára.

Először is, maximalizálja a kiadott kód minőségét. A hibák feltárásának és javításának költsége sokkal alacsonyabb a sprinten belül, mint az után.

Ettől függetlenül, még ha a hibák számát minimalizálta is, biztos, hogy érkeznek majd hibajelentések a sprint vége után is. Hogyan kezelhetjük ezeket?

1. Stratégia: "Új fejlesztésekbe csak azután kezdünk, hogy a előző dolgok már élesben működnek"

Jól hangzik, nem? Ön is átérezte mennyire jónak tűnik?

Párszor már elég közel kerültünk ennek a stratégiának a bevezetéséhez és sokat agyaltunk, hogy hogyan is fogjuk majd csinálni. De valahogy mindig meggondoltuk magunk a módszer hátulütői miatt,

ugyanis ezzel megjelenik egy nem időkorlátos elem a sprintek között, ahol csak tesztelünk és debuggolunk amíg össze nem áll egy kiadásra kész verzió.

<figure>

Nem igazán barátkoztunk meg egy nem időkorlátos elem gondolatával a sprintjeink között, mivel ez teljesen felborítaná a sprintek szívverés-szerű ütemét. Nem mondhatnánk többé, hogy "három hetente
új sprint indul". Ráadásul ez sem orvosolja teljesen a problémát. Ha be is iktatunk ilyen időszakokat, attól még fel kell készülnünk az időről-időre felbukkanó sürgős hibajelentésekre.

2. Stratégia: "Elkezdhetünk dolgozni új fejlesztéseken, de az előző dolgok éles rendszerre telepítése élvez elsőbbséget."

Mi ezt a megoldást javasoljuk, legalábbis jelenleg.

Amint egy sprintet befejezünk, kezdjük a következőt, de arra számítunk, hogy az időnk egy részét az előző sprint hibáinak javításával fogjuk tölteni. Ha a következő sprint jelentős hátrányba kerül az előző sprintbe felmerülő hibák miatt, akkor megpróbáljuk megérteni, hogy ez miért történt és hogy hogyan kerülhetjük ezt el a jövőben. Fontos, hogy a sprintjeink elég hosszúak legyenek, hogy elég időnk legyen az előző sprint hibáinak javítására.

Ezt a stratégiát alkalmazva néhány hónap alatt az előző sprintekben előforduló hibák javítására fordított idő, továbbá a hibákon dolgozó emberek száma is fokozatosan csökkent. Mára már sokkal elfogadhatóbb szinten vagyunk.

<figure>

A sprint planning meeting során elég alacsonyra választjuk a koncentrációs tényezőt, hogy maradjon elég idő a hibák javítására. Idővel a csapat elég gyakorlottá vált ennek a tényezőnek a becslésében. A sebességgel kapcsolatos információk is sokat segítenek (Lásd: "Hogyan dönti el a csapat, hogy mely
sztorik kerüljenek a sprintbe?").

Rossz stratégia - "új fejlesztésekre koncentrálni"

Ez voltaképp azt jelenti, hogy "új fejlesztésekre koncentrálunk ahelyett, hogy a már meglevő dolgokat az éles rendszerbe integráljuk". Ugyan ki akarná ezt? A kezdetekben mégis gyakran beleestünk ebbe a csapdába és biztos vagyok benne, hogy mások vállalatokban is előfordul. Ez egy stressz miatt fellépő tünet. Rengeteg menedzser nem igazán tudja, hogy ha a kódolás készen van, az
még nem jelenti azt, hogy közel van az éles kiadás - legalábbis bonyolult rendszerek esetén - így a menedzser (vagy a Product Owner) egyre több új funkciót kér a csapattól, miközben a régebbi, "majdnem-kész-az-éles-telepítésre" funkciók listája egyre csak bővül, lassítva a fejlesztést.

Ne hagyja le a leglassabb láncszemet

Mondjuk, hogy az elfogadási teszt a leglassabb láncszem: túl kevés teszterrel rendelkezik, vagy a horribilis kód-minőség miatt sokáig tart a tesztelés. Tegyük fel, hogy a teszter csapat heti három funkciót képes tesztelni (nem, nem használjuk a "funkció per hét" mértékegységet, ezt kizárólag a példa kedvéért találtam ki), illetve, hogy a fejlesztők 6 funkciót fejlesztenek hetente.

Elég csábítóan hangzik egy menedzser vagy Product Owner (esetleg akár a csapat egésze) számára, hogy
heti 6 funkciós ütemet diktáljon.

Ne tegye! A valóság így vagy úgy, de utoléri majd és az elég fájdalmas.

Ehelyett ütemezzen 3 funkciót hetente és a fennmaradó időt fordítsa a tesztelési problémák feloldására. Például:
* Kérjen fel néhány fejlesztőt, hogy teszteljenek (nagyon fognak ennek örülni...)
* Automatizálja a teszteket
* Növelje a sprint hosszát és vegye be az elfogadási teszt fázist a sprintbe.
* Néhány sprintet jelöljön ki "teszt sprint"-nek, amikor az egész csapat az elfogadási teszteken dolgozik.
* Vegyen fel további tesztereket (még ha ez fejlesztők elbocsájtását is jelenti)

Mi az utolsón kívül az összes felsorolt stratégiát kipróbáltuk. Hosszú távon a 2-es és 3-as pont működik a legjobban.

A visszatekintők remek lehetőséget biztosítanak a leglassabb elem azonosítására.

Vissza a valóságba

Ezek után azt gondolhatná, hogy minden csapatunkban külön tesztereket és minden termékhez külön
elfogadási teszter csapatot alkalmazunk, hogy minden sprint után kiadás történik, stb.

Pedig nem.

Volt már rá példa, hogy ezeket meg tudtuk tenni és meggyőződtünk az előnyeiről, de még elég messze
vagyunk egy használható minőségellenőrzési folyamattól és még sokat tanulhatunk ezen a téren.

15 Hogyan kezelünk egyszerre több Scrum csapatot
==========
A dolgokat jelentősen bonyolítja, ha több Scrum csapat dolgozik egyszerre ugyanazon a terméken. A probléma univerzális és nem kizárólag a scrum jellegzetessége: több fejlesztő = több komplikáció.

Mint máskor is, ezzel is sokat kísérletezgettünk. Az ugyanazon a terméken dolgozó legnagyobb csapatunk 40 főt számlált.

Az alapvető kérdések:
* Hány csapatot alakítsunk

* Hogyan osszuk el az embereket a csapatok között

Hány csapatot alakítsunk

Ha a több scrum csapat kezelése olyan nehéz, akkor miért is vesződünk? Miért nem osztunk mindenkit
egy csapatba?

A legnagyobb csapat, amit kipróbáltunk 11 főt számlált. Működött, de korántsem olajozottan. A napi scrum rendre kicsúszott a 15 perces keretből, a csapattagok nem tudták, hogy a többiek min dolgoznak, a Scrum Masternek rengeteg erőfeszítésébe került a koordinálás és az akadályok elhárítása.

Ehelyett két csapatra is oszolhatunk, de vajon az jobb-e? Nem minden esetben.

Ha a csapatnak van már gyakorlata a Scrum-ban és logikusan feloszthető az ütemterv két párhuzamos
részre úgy, hogy ezek a részek nem érintik ugyanazt a kódot, akkor úgy gondolom, hogy érdemes két
csapatra oszlani, de egyébként a hátrányok ellenére is inkább az egy csapatot javasolnám.

Az a tapasztalatom, hogy jobb kevesebb nagy csapat, mint sok kicsi, amik állandóan zavarják egymást.
Csak akkor alakítson kis csapatokat, ha kicsi a valószínűsége, hogy zavarni fogják egymást.

Virtuális csapatok

Honnan tudni, hogy helyesen döntött a csapatméreteket illetően? Ha nyitott szemmel jár, előbb-utóbb azt tapasztalja majd, hogy virtuális csapatok formálódnak.

1. példa: Egy nagy csapat mellett dönt, de amikor megfigyeli, hogy a sprint alatt ki kivel beszél, azt találja, hogy a csapat gyakorlatilag két kisebb alcsoportra oszlott.

<figure>

2. példa: Három kisebb csapat mellett dönt, de amikor megfigyeli, hogy a sprint alatt ki kivel beszél, azt találja, hogy az 1. és 2. csapat állandóan kommunikál egymással, míg a hármas csapat önállóan dolgozik.

<figure>

Ez azt jelenti, hogy a csapatfelosztási stratégiája helytelen volt? Amennyiben a virtuális csapatok állandónak bizonyulnak, igen. Ha a virtuális csapatok ideiglenesek, a válasz nem.

Vessünk újra egy pillantást az 1. példára. Ha a két virtuális alcsoport néha változik (azaz
a tagok mozognak a virtuális alcsoportok között), akkor valószínűleg helyes volt azon döntése, hogy egy nagy csapatot alakítson. Ha azt találja, hogy a virtuális csapatok a sprint egészében

változatlanok, akkor érdemes lehet két csapatra oszlani a következő sprintben.

Most nézzük meg újra a 2. példát. Ha az 1-es és 2-es csapat kommunikál egymással (de a 3-as csapattal nem) az egész sprint alatt, akkor valószínűleg érdemes összevonni őket egy csapattá a következő sprintben. Ha a sprint felében az 1-es és 2-es csapat kommunikál, de másik felében inkább
az 1-es és 3-as, akkor megfontolandó a három csapat egybeolvasztása, bár akár békén is hagyhatja a
felosztást. Hozza fel a kérdést a sprint visszatekintőben és hagyja, hogy a csapatok maguk döntsenek.

A csapatfelosztás a Scrum egyik legnehezebb része. Ne próbálja túlanalizálni a helyzetet, inkább kísérletezzen, figyelje meg a virtuális csapatok alakulását és biztosítson elég időt a vitára a sprint visszatekintőben. Előbb-utóbb meg fogja találni az arany középutat. A lényeg, hogy a csapatok elégedettek legyenek és ne kelljen lépten-nyomon más csapatokba botlaniuk.

Optimális csapatméret

A szakirodalom az "optimális" csapatméretet általában 5 és 9 fő között definiálja, amivel eddigi tapasztalataim alapján én is egyetértek, bár én inkább 3 és 8 közé helyezném az optimális méretet. Úgy vélem, hogy érdemes harcolni ezen méret eléréséért.

Mondjuk egy 10 fős Scrum csapat esetén fontolja meg a két leggyengébb tag kiiktatását. (Hoppá, tényleg ezt mondtam?)

Tegyük fel, hogy adott két termék, amin egy-egy 3 fős csapat dolgozik és mindkettő túl lassú. Érdemes lehet egy nagyobb, 6 fős csapatot alakítani, amely mindkét termékért felelős. Ebben az esetben az egyik Product Owner-nek mennie kell (vagy helyezze például tanácsadói pozícióba).

<figure>

Tegyük fel, hogy annyira rossz állapotban van a kódbázisa, hogy képtelenség két különböző csapatra
bízni a fejlesztést és így egy nagy 12 fős csapata van. Tegyen komoly erőfeszítéseket a kód kijavítására (akár az új funkciók rovására is), amíg olyan helyzetbe kerül, hogy a csapat könnyen felosztható. A befektetett munka rendkívül gyorsan meg fog térülni.

Szinkronizált sprintek - vagy ne?

Tegyük fel, hogy három Scrum csapat dolgozik ugyanazon a terméken. Szinkronizáljuk a sprintjeiket (azaz kezdődjenek és végződjenek egyszerre) vagy inkább legyen időbeli átfedés a sprintek között?

Első közelítésben átfedő sprinteket használtunk.

<figure>

Ez jól hangzik, mivel mindig lesz egy sprint, amely nemsokára véget ér és egy, ami nemsokára

kezdődik. A Product Owner terhelése egyenletesen oszlik el és folyamatosan érkeznek kiadások. Minden
héten demó! Halleluja!

Tudom, tudom, de akkoriban tényleg meggyőzően hangzott.

Épp elkezdtük használni ezt a stratégiát, amikor szerencsém volt beszélni Ken Schwaber-rel (a Scrum
tanúsítványom kapcsán). Ő mutatott rá, hogy ez egy nagyon rossz ötlet és jobb lenne szinkronizálni a
sprinteket. Nem emlékszem már a konkrét érveire, csak arra, hogy a beszélgetés során meggyőzött.

<figure>

Azóta is ezt a stratégiát alkalmazzuk és ezt nem bántuk meg. Sosem fogom megtudni, hogy az átfedő
sprintek tényleg szétesnének-e, de az az érzésem, hogy igen. A szinkronizált sprintek előnyei:
* Természetes keretet biztosít a csapatok átrendezésére: a sprintek közötti időt. Ha a sprintjeink átfednék egymást, a csapatok átrendezése legalább egy csapat sprintjét megzavarná.
* Az összes csapat ugyanazon a sprint cél elérésén dolgozhat és együtt tervezhetik a sprintjeiket, ami hatékonyabb együttműködéshez vezet a csapatok között.
* Kevesebb adminisztrációval jár: kevesebb sprint planning meeting, kevesebb sprint demó és kiadás.

Miért vezettük be a team lead szerepkört

Legyen egy termékünk három csapattal.

<figure>

A P-vel jelölt vörös figura a Product Owber. Az S-sel jelölt fekete figurák a Scrum Master-ek. A többiek a rabszolgák... izé... megbecsült csapattagok.

Ebben a felállásban ki dönti el, hogy ki melyik csapathoz tartozik? A Product Owner? Esetleg a három
Scrum Master közösen? Vagy mindenki maga dönt, hogy melyik csapathoz szeretne tartozni? Mi van akkor, ha mindenki az 1-es csapatban szeretne lenni (mert az 1-es csapat Scrum Master-e olyan jóképű)?

Mi legyen akkor, ha később kiderül, hogy ezen a kódon legfeljebb két csapat tud egyszerre párhuzamosan dolgozni, így két kilencfős csapatra kell oszlani három 6 fős helyett. Ez két Scrum Mastert igényel, a jelenlegi három közül melyik legyen megfosztva a címétől?

A legtöbb vállalatnál ezek elég kényes kérdések.

Elég vonzónak tűnhet az a megoldás, hogy a Product Owber döntsön az emberek szétosztásáról, de ezek

nem igazán tartoznak a Product Owner feladatköréhez, mivel ő normális esetben a szakértő szerepét játssza, aki a termék irányvonalát szabja meg és nem az ilyen apró-cseprő dolgokkal foglalkozik. Főként mivel a szerepe "csirke" szerep (ha esetleg nem ismerné a csirke és a malac metaforáját, keressen rá a google-ön).

Ezt végül a team lead szerepkör bevezetésével oldottuk meg. Lehetne még "Scrum of Scrums Master"-nek, vagy "A főnök"-nek, esetleg "fő Scrum Master"-nek is nevezni. Ez a személy nem irányít egy csapatot sem, ehelyett a csapatok közt felmerülő kérdésekben hivatott dönteni, mint például a Scrum Masterek kijelölése, a csapatok beosztása, stb.

Próbáltunk egy megfelelő nevet találni erre a szerepkörre, végül a "team lead" hangzott a legkevésbé idiótán.

Ez a megoldás nagyon jól működött nálunk és csak ajánlani tudom (attól függetlenül, hogy végül hogy
fogja nevezni ezt a szerepkört).

Hogyan rendeljük a tagokat a csapatokhoz

Két alapvető stratégia van az emberek csapatokhoz rendelésére, ha egy termékhez több csapat is tartozik:
* Egy kijelölt személy, mint például a fentiekben említett "team lead", a Product Owner, vagy egy menedzser osztja be a csapatokat.
* A tagok maguk oszlanak csapatokra.

Mi mindhárom stratégiát kipróbáltuk. Hogy hogyhogy mindhármat? Az 1-es stratégiát a 2-es stratégiát
és a kettő kombinációját. Végeredményben a kettő kombinációja tűnik a legjobbnak.

A sprint planning meeting előtt a team lead összehív egy csapat-elosztó meetinget a Product Owner és
az összes Scrum Master között. Itt összefoglaljuk az előző sprintet és eldöntjük, hogy szükséges-e a csapatok újrarendezése. Elképzelhető, hogy két csapatot össze szeretnénk vonni, vagy néhány embert
egyik csapatból a másikba mozgatni. Összeírjuk ezeket a javaslatokat, amit aztán ismertetünk a sprint planning meeting során.

A sprint planning meetingen először végigmegyünk a legfontosabb sztorikon, majd a team lead valami
hasonlóval áll elő:

"Jó napot mindenkinek. A következő sprint idejére az alábbi javaslataink vannak a csapatfelosztást illetően."

<figure>

"Mint láthatjátok, ez azt jelenti, hogy négy helyett három csapatunk lenne, amibe előzetesen

beosztottunk mindenkit. Az egyes csapatok találjanak maguknak egy fal-részt."

(a team lead megvárja, amíg mindenki megtalálja a helyét és ki nem alakul három csoport, akik egy-egy üres falfelület mellett állnak)

"Ez nem feltétlenül a végleges felosztás, csak egy kiindulópont. A sprint planning meeting során az ésszerűség határain belül bárki átmehet bármelyik csapatba, kettévághatjátok vagy egyesíthetitek a csapatotokat tetszés szerint. Próbáljátok figyelembe venni a Product Owner által meghatározott fontossági sorrendet."

Nálunk ez a stratégia működik a legjobban. Egy előzetes javarészt központosított döntés, amit aztán javarészt decentralizált optimalizáció követ.

Szakosodott csapatok - vagy ne?

Tegyük fel, hogy a következő három komponens alkotja a technológiáját:

<figure>

Tegyük továbbá fel, hogy 15 ember dolgozik ezen a terméken, ezért nem szeretne egy nagy csapatot alkotni. Milyen csapatokra oszoljanak?

1. Stratégia: az egyes komponensekre szakosodó csapatok

Egy megoldás, hogy a csapatokat a komponensek mentén osztjuk fel, például "kliens csapat", "szerver
csapat", "adatbázis csapat".

<figure>

Mi ezt használtuk először, azonban ez nem működik túl jól, legalábbis ha a legtöbb sztori több komponenst is érint.

Legyen mondjuk az egyik sztori az, hogy "üzenőfal, amelyen a felhasználók üzenetet küldhetnek egymásnak". Ez az üzenőfal funkció érinti a kliens felhasználói felületét, új logikát jelent a szerveren és új táblákat az adatbázisban.

<figure>

Ez azt jelenti, hogy mindhárom - kliens-, szerver-, adatbázis- - csapat együtt kell, működjön a sztori befejezéséhez. Nem túl jó ötlet.

2. Stratégia: Multi-komponens csapatok

A második lehetőség, hogy az összes komponenssel egyszerre foglalkozó csapatokat hozunk létre.

<figure>

Ha a legtöbb sztori több komponensen is átnyúlik, akkor ez a csapatfelosztási stratégia hatékonyabb. A csapatok egész sztorikon dolgozhatnak még ha a sztorik az összes szinte (kliens, szerver és adatbázis) módosítását igénylik is, ráadásul a csapatok egymástól függetlenül dolgozhatnak, ami Jó Dolog.

A Scrum bevezetésekor az egyik legelső változtatásunk épp az egyes komponensekre specializált (1.
stratégia) csapatok felbontása és az összes szinten átnyúló csapatok (2. stratégia) létrehozása volt. Ezzel jelentősen csökkent az olyan esetek száma, amikor például "azért nem tudtuk befejezni ezt a sztorit, mert a szerver csapatra várunk".

Időnként azonban, ha erre igény van, továbbra is alkotunk időlegesen egyes komponensekre specializált csapatokat.

Csapatok újraosztása sprintek között - vagy ne?

Attól függően, hogy épp mi a legfontosabb cél vagy épp milyen sztorijaink vannak, a sprintek elég eltérőek tudnak lenni, aminek eredményeképp sprintenként más és más lehet az optimális csapatfelosztás.

Az az igazság, hogy szinte minden sprintre azt mondtuk, hogy ez a sprint azért nem igazán átlagos sprint, mert... és itt valami blabla következett. Egy idő után aztán feladtuk az "átlagos" sprintekbe vetett hitünket: nincsenek "átlagos" sprintek, csakúgy, mint ahogy "átlagos" családok vagy ahogyan "átlagos" emberek sincsenek.

Az egyik sprintben hasznos lehet egy kizárólag kliens-oldallal foglalkozó csapatot alakítani azokkal a tagokkal, akik jól ismerik a kliens kódját. Egy másik sprintben hasznos lehet két komponenseken átnyúló csapatot alakítani és szétosztani a kliens kódját jól ismerő tagokat a kettő között.

A Scrum egyik alapvető kérdése a csapat "olajozottsága", vagyis ha a csapattagok huzamosabb időn keresztül együtt dolgoznak, akkor előbb-utóbb szoros kapcsolatok alakulnak ki, ami egy közös, rendkívül hatékony állapothoz vezet. Ehhez azonban több sprintre van szükség, így ha túl gyakran cserélgeti a csapattagokat, nem fogja elérni ezt az olajozottan működő állapotot.

Amennyiben a csapatok átrendezését fontolgatja tehát, gondoljon a következményekre is. Hosszú távra
szól ez a változás? Ha igen, nyugodtan változtasson, de ha nem, akkor lehet, hogy érdemesebb nem bolygatni a csapatfelosztást.

Az egyetlen kivétel amikor egy nagyobb csapatnál először vezeti be a Scrum-ot. Ebben az esetben megéri kísérletezgetni különböző csapatfelosztásokkal addig, amíg olyan felosztást nem talál, amivel mindenki elégedett. Tegye egyértelművé, hogy az első pár alkalommal az sem baj, ha teljes katasztrófa a felosztás, feltéve, hogy ezen mindenki folyamatosan javítani próbál.

Részidős csapattagok

Én is csak azt tudom mondani, amit a Scrum irodalom: nagy általánosságban rossz ötlet részidős csapattagokat alkalmazni.

Tételezzük fel, hogy Joe-t részidőben alkalmazná a csapatában. Fontolja meg a következőket. Először
is tényleg szüksége van Joe-ra a csapatban? Nem tudja Joe-t teljes idejű csapattagként alkalmazni? Joe-nak milyen más feladatai vannak? Nem lehet valaki mást megkérni, hogy Joe ezen feladatait - esetleg Joe passzívabb felügyelete mellett - átvegye? Elképzelhető-e, hogy Joe a következő sprinttől teljes időben csatlakozzon és addigra átadja a jelenlegi feladatait valaki másnak?

Néha nincs más megoldás, Joe-ra égető szüksége van, mert ő az egyetlen adatbázis-adminisztrátor az
épületben, de persze a többi csapatnak is ugyanígy szüksége van rá, így sosem érheti el, hogy teljes időben csatlakozzon az Ön csapatához, ráadásul az is elképzelhetetlen, hogy a vállalat új DBA-t vegyen fel. Rendben, ebben az esetben tökéletesen elfogadható, hogy részidőben alkalmazza Joe-t (egyébként mi pontosan ebben a helyzetben voltunk), de ne feledje el minden esetben mérlegelni ezeket a kritériumokat.

Személy szerint én szívesebben dolgoznék 3 teljes idejű taggal, mint 8 részidőssel.

Ha van is valaki, aki az előző példában látott DBA-hoz hasonlóan több csapat között osztja meg az idejét, akkor is érdemes elsődlegesen valamelyik csapathoz osztani. Fontolja meg, hogy melyik csapatnak van a legnagyobb szüksége erre a személyre és rendelje ehhez a csapathoz. Hacsak nincs
dolga máshol, ennek a csapatnak a napi scrumjain, sprint planning meetingjein és visszatekintőin vesz részt.

Hogyan alkalmazzuk a "Scrumok Scrumját"

A Scrumok Scrumja egyszerűen egy rendszeres meeting az összes Scrum Master között.

Egy ponton egyszerre négy terméken dolgoztunk, amiből háromhoz egy-egy Scrum csapat volt rendelve,
míg a negyediken 25 ember dolgozott több kisebb csapatba osztva az alábbi ábrához hasonlóan:

<figure>

Ez azt jelenti, hogy a Scrumok Scrumja két szintű volt, egy "termék szintű" Scrumok Scrumja, amibe a
D termékhez tartozó csapatok tartoztak és egy "vállalati szintű" Scrumok Scrumja, ami az összes terméket magába foglalta.

Termék szintű Scrumok Scrumja

Ez a heti rendszerességű, vagy esetenként akár gyakoribb meeting rendkívül fontos volt. Itt beszéltük meg az integrációs- és a csapategyensúlyt érintő kérdéseket, itt készültünk fel a következő sprint planning meetingekre, stb. Fél óra volt kiszabva rá, de gyakran kicsúsztunk az

időből. Az alternatíva az lett volna, hogy minden nap összeülünk, de végül ezt sosem próbáltuk ki.

A meeting menetrendje a következő volt:
1) Sorban mindenki elmondja, hogy a csapat mit csinált a múlt héten, mit terveznek csinálni ezen a héten és milyen akadályokkal néznek szembe.
2) Bármi más, több csapatot is érinthető kérdés megvitatása (mint például integrációs problémák)

Ez a menetrend igazából annyira nem fontos, a lényeg, hogy rendszeresen tartson Scrumok Scrumja meetinget.

Vállalati szintű Scrumok Scrumja

Ezt a meetinget csak "Pulzus"-nak becéztük. Sokféle felállással próbálkoztunk, de mostanában az egészet inkább elhagytuk és helyette hetente tartunk teljes céges helyzetjelentést (a teljes cég alatt persze a fejlesztésben érintett személyeket kell érteni). Az egész 15 perc. Hogy micsoda? 15 perc? A teljes céggel? Az összes termék összes csapatának összes tagjával? Ez működőképes?

Igen, de csak akkor ha a szervezője ténylegesen betartatja a rövid időkorlátot.

A meeting a következőképp néz ki:
1) A fejlesztés vezetője ismerteti a főbb híreket, következő eseményeket.
2) Minden termék csoportjából egy ember elmondja, mit csináltak a múlt héten, mit terveznek erre a hétre és esetleg ha valamilyen akadállyal küszködnek. Néhány további ember is megszólal (konfiguráció-menedzsment, minőség-ellenőrzés, stb.)
3) Bárki más szabadon hozzáadhat vagy kérdezhet.

A meeting rövid és tömör információcserére szolgál, nem pedig vitákra vagy elmélkedésekre. Ha így használjuk, akkor 15 perc általában elég és ha ki is futunk néha az időből, nagyon ritka, hogy fél óránál többet vegyen igénybe. Ha valahol vita alakulna ki, egy pillanatra megállunk és megkérem az érdekelteket, hogy a meeting után maradjanak és ott folytassák a megbeszélést.

Azért tartjuk ez a teljes céges "pulzus" meetinget, mert tapasztalatunk szerint a vállalati szintű Scrumok Scrumja amúgy is javarészt információközlésből állt, szinte sose volt tárgyalás jellegű. Ráadásul a jelenlevőkön kívül is sokakat érdekeltek és érintettek ezek az információk, úgyhogy inkább úgy döntöttünk, hogy ha már egyszer időt fordítunk arra, hogy megosszuk, hogy az egyes csapatok hogyan állnak, miért ne osztanánk ezt meg mindenkivel.

A Napi Scrumok időzítése

Problémát okoz, ha több ugyanazon a terméken dolgozó Scrum csapat is ugyanabban az időben tartja a
Napi Scrumját, mert így a Product Owner (és a hozzám hasonló kíváncsiskodók) csak egy napi scrumra
tudnak bemenni.

Ezért aztán arra kérjük a csapatokat, hogy ha lehet ne ugyanabban az időben tartsák a Napi Scrumjaik.

<figure>

A fenti példa abból az időből származik, amikor a Napi Scrumjainkat egy külön irodában és nem a csapat szobában tartottuk. Az egész általában 15 perc, de minden csapat egy 30 perces időkeretet kap
arra az esetre, ha egy kicsit túlszaladnának a 15 percen.

Ez két szempontból is hasznos:
1. A Product Owner és a hozzám hasonló emberek az összes napi scrumon jelen tudnak lenni minden
reggel, aminél nincs jobb módszer a sprint pontos állapotának és az esetleges kockázatok felmérésére.
2. A csapattagok beülhetnek más csapatok napi scrumjaira is. Bár ez nem túl gyakran fordul elő, néha
több csapat is hasonló dolgon dolgozik, ezért néhányan belehallgatnak a másik csapatok napi scrumjaiba, hogy naprakészek legyenek.

A dolog hátránya, hogy kevesebb szabadságot biztosít a csapatoknak, mivel nem választhatnak tetszőleges időpontot a napi scrumra. Nálunk azonban ez eddig sosem okozott gondot.

Tűzoltó csapatok

Egy esetben egy nagyobb terméket fejlesztő csapatban képtelenek voltunk bevezetni a Scrum-ot, mivel
túl sok idejüket töltötték tűzoltással, azaz a félkészen kiadott rendszerük pánikszerű foltozásával. Ez egy ördögi kör, azzal hogy az idejük nagy részét a tűzoltásra kellett fordítaniuk, nem jutott idejük a tűzvész megelőzésére (mint például az architektúra javítása, tesztek automatizálása, monitoring rendszer bevezetése, riasztások beállítása, stb.)

Ezt azzal orvosoltuk, hogy egy külön tűzoltó- és egy külön Scrum csapatot hoztunk létre.

A Scrum csapat feladata volt - a Product Owner beleegyezésével - a rendszer stabilitásának javítása és gyakorlatilag a tüzek megakadályozása.

A tűzoltó csapatnak (amit igazából "terméktámogató" csapatnak hívtunk) két feladata volt:
1) Tűzoltás
2) Elhatárolni a Scrum csapatot az olyan zavaró tényezőktől, mint például a derült égből váratlanul felbukkanó ad-hoc kérések.

A tűzoltó csapat az ajtóhoz közel, míg a Scrum csapat az iroda másik végében ült, így fizikailag is védelmet nyújthattak például egy túlbuzgó sales-es vagy egy dühös felhasználó ellen.

Mindkét csapatba osztottunk senior fejlesztőket, hogy egyik csapatnak se kelljen a másikra hagyatkoznia.

Ez tulajdonképpen a Scrum bevezetésének problémáját volt hivatott megoldani. Hogyan kezdhetünk

neki
a Scrumnak, ha a csapatnak esélye sincs egy napnál előbbre tervezni? Mi ezt úgy oldottuk meg, hogy a
csapatot kettéosztottuk, ami remekül működött.

Mivel a Scrum csapat elég teret kapott, hogy előrelátóan dolgozzon, végre stabilizálni tudták a rendszert. Mindeközben a tűzoltó csapat teljesen lemondott arról, hogy előre tervezzenek és reaktív módon dolgoztak mindig az éppen felfedezett hibákon.

Persze a Scrum csapat hétköznapjai nem voltak teljesen zavarmentesek, a tűzoltó csapat gyakran kellett, hogy konzultáljon a Scrum csapat egyes tagjaival, vagy rossz esetben az egész csapattal.

Végül aztán egy pár hónap múlva a rendszer elég stabillá vált, hogy teljesen megszabadulhattunk a tűzoltó csapat gondolatától és helyette egy újabb Scrum csapatot alakíthattunk ki. A tűzoltók nagy örömmel akasztották a szögre kormos sisakjaik, hogy végre a Scrum csapatokhoz csatlakozzanak.

A product backlog szétbontása - vagy ne?

Mondjuk van egy terméke két Scrum csapattal. Hány product backlogot készítsen? Hány Product Owner
legyen? Ezzel kapcsolatban három stratégiát próbáltunk ki. A választás nagyban ki fog hatni a sprint planning meeting menetére.

1. Stratégia: Egy Product Owner, egy backlog

Mi ezt a "csak egy maradhat" modellt favorizáljuk.

A modell szépsége, hogy a csapatokat hagyja maguktól kialakulni a Product Owner aktuális prioritásai
alapján. A Product Owner meghatározza, hogy mire van szüksége, majd a csapatokra bízza a munka felosztását.

<figure>

Még közelebbről a sprint planning meeting az alábbiak szerint működik:

A sprint planning meetinget egy közeli konferenciaközpontban tartjuk.

Közvetlenül a meeting előtt, a Product Owner kijelöl egy falat "product backlog fal"-nak és ide teszi fel a sztorik cetlijeit fontossági sorrendben amíg a fal be nem telik (ami általában több is, mint amire a sprintben szükség van)

<figure>

Minden Scrum csapat választ egy üres faldarabot magának. Ezután minden csapat a legfontosabbakkal
kezdve sztorikat tesz át a product backlogból a saját falára.

A következő ábra hivatott ezt illusztrálni. A sárga nyíl jelképezi a sztori cetlik haladási irányát a product backlogból a csapatok falaira.

<figure>

A meeting folyamán a product owner és a csapatok egyezkednek a cetliket illetően, egyik csapattól a másikba tehetik, felfelé vagy lefelé mozgathatják őket hogy megváltoztassák a prioritásokat vagy kisebb sztorikra bonthatják azokat, stb. Nagyjából egy órán belül minden csapat falán kialakul egy előzetes sprint backlog. Ezután a csapatok már függetlenül dolgozhatnak a feladatokra bontáson és az
időbecsléseken.

<figure>

Ez elég kaotikus és fárasztó tud lenni, de ugyanakkor hatékony, szórakoztató és közösségi tevékenység is. Az idő lejártával az összes csapatnak elég információja lesz, hogy elkezdjen a sprinten dolgozni.

2. Stratégia: Egy Product Owner, több backlog

Ebben a modellben a product owner csapatonként egy product backlogot készít. Ugyan közel jártunk ennek a stratégiának a használatához, de igazából sosem próbáltuk ki. Ha az 1. stratégia nem működne, ez lenne a következő, amit kipróbálnánk.

A stratégia gyenge pontja, hogy a Product Owner oszt sztorikat az egyes csapatokra, ami egy olyan tevékenység, amit hatékonyabb a csapatokra bízni.

<figure>

3. Stratégia: Több Product Owner saját backloggal

Ez nagyban hasonlít a 2. stratégiára azzal a különbséggel, hogy csapatonként nem csak egy külön backlog, de külön Product Owner is van.

<figure>

Ezt sosem próbáltuk ki és nagyon valószínű, hogy sosem fogjuk.

Ha a két product backlog ugyanazt a kódbázist érinti, ez valószínűleg komoly érdekellentéthez vezet a Product Ownerek között.

Ha a két product backlog két külön kódbázist érint, ez gyakorlatilag ugyanolyan, mintha a terméket magát bontanánk két független részre, azaz visszajutunk az egy csapat per termék szituációhoz, ami viszont egyszerűen kezelhető.

Branchek

Ha több csapat dolgozik ugyanazzal a kódbázissal, akkor elkerülhetetlen, a branchek használata. Rengeteg könyv foglalkozik azzal a kérdéssel, hogy mit tegyünk ha több fejlesztő dolgozik egyszerre ugyanazon a kódon, ezért a részletekbe nem mennék bele és mivel nincs semmi forradalmi újdonság,
amit hozzáadhatnék a témához, inkább összefoglalnám a csapatunkban megtanult fontosabb leckéket.

* Biztosítsa, hogy a master branch (trunk) mindig konzisztens, azaz hogy a kód minimum lefordul és a
  tesztek sikeresen lefutnak, így bármikor működő kiadást készíthet. Remélhetőleg a folyamatos build rendszer minden éjjel automatikusan telepíti a legfrissebb változatot a teszt környezetre.
* Minden kiadáshoz készítsen tag-et. Minden egyes elfogadási tesztre- vagy élesbe szánt kiadáshoz készítsen egy olyan tag-et, ami egyértelműen jelzi, hogy pontosan mi került kiadásra. Így a jövőben bármikor visszatérhet és egy karbantartási ágat készíthet ettől a ponttól.
* Csak akkor készítsen új branch-eket, ha ez feltétlenül szükséges. Az ökölszabály, hogy akkor indítunk új ágat, ha a jelenlegi ág szabályait nem tudunk betartani az új fejlesztéssel. Ha kételyei vannak, inkább ne készítsen új branch-et, mert minden aktív ág jelentősen növeli az adminisztrációt és a komplexitást.
* A brancheket elsődlegesen a különböző életciklusok szétválasztására használja. Az nem olyan fontos
  kérdés, hogy az egyes Scrum csapatoknak saját ága legyen-e, de ha vegyíti a rövid- és hosszú távú hibajavításokat egy ágon belül, az sok problémát fog okozni amikor a rövid távú javításokat ki szeretné adni.
* Szinkronizáljon gyakran. Ha külön ágon dolgozik, mindig szinkronizáljon a fő ághoz amint működőképes állapotba kerül. Minden reggel frissítsen a fő ágból, hogy szinkronban maradjon a többi tag változásaival. Ha ez sok merge konfliktust eredményez, fogadja el, hogy ha várna, a helyzet ennél csak rosszabbá válna.

Visszatekintő több csapattal

Hogyan szervezzük a sprint visszatekintőket, amikor több csapat dolgozik ugyanazon a terméken?

A sprint demó végeztével, amikor elhal a tapsvihar és a csapatoknak volt némi idejük keveredni, minden csapat elvonul egy külön irodába vagy egy kényelmes külső helyre és többé-kevésbé ugyanúgy
bonyolítják le a visszatekintőjüket, mint ahogy a "Hogyan szervezzük a visszatekintőket" pontnál leírtam.

A sprint planning meeting alatt (amin az összes csapat részt vesz, mivel a sprintjeink szinkronizáltak a termékeken belül) a legelső, ami történik, hogy minden csapatból feláll egy kijelölt személy, aki összefoglalja a visszatekintőn megbeszélteken. Ez csapatonként nagyjából 5 percet vesz igénybe, ami után 10-20 percnyi időt biztosítunk vitákra. Ezt követően rövid szünetet tartunk, ami után elkezdődhet az igazi sprint tervezés.

Nem próbáltunk ki más módszereket, mivel ez megfelelően működik. A legnagyobb hátrány talán az, hogy
nincs pihenőidő a sprint visszatekintő rész és a sprint planning meeting rész között. (Lásd

"Pihenőidő a sprintek között")

Azoknál a termékeknél, ahol csak egy csapat van, a sprint planning meeting során nem szükséges összefoglalni a visszatekintőn megbeszélteket, mivel mindenki részt vett a visszatekintőn.

16 Mit teszünk földrajzilag elosztott csapatok esetén

Mi történik, ha a csapat tagjai különböző helyeken vannak? A Scrum és XP "varázsa" pont az egy helyen levő, szorosan együttműködő csapattagokra épül, akik páros programozást használnak és nap
mint nap találkoznak.

Nálunk is vannak földrajzilag elválasztott csapatok, illetve olyan tagok, akik időnként otthonról dolgoznak.

A stratégiánk rendkívül egyszerű: minden rendelkezésünkre álló eszközt bevetünk annak érdekében, hogy a különböző helyeken lévő tagok közti kommunikáció sávszélességét maximalizáljuk. És itt nem csak Mbps jellegű sávszélességekről beszélek (bár persze az is lényeges), hanem a szó tágabb értelmében:

* Páros programozás lehetősége
* A Napi Scrum-on szemtől-szembeni találkozás lehetősége.
* Lehetőség bármikor szemtől-szemben beszélgetni
* Lehetőség fizikailag találkozni és szocializálni
* Spontán meetingek összehívásának lehetősége
* Annak lehetősége, hogy ugyanazt a sprint backlogot, burndown-t, product backlogot és egyéb információt láthassuk

Azon dolgok, amiket már bevezettünk (vagy éppen bevezetés alatt állnak, mivel még nem végeztünk minddel):

* Webkamera és headset minden géphez.
* Távolról is elérhető konferenciatermek, webkamerákkal, mikrofonokkal, mindig készen álló számítógépekkel, távoli asztal megosztó szoftverrel, stb.
* "Távoli ablakok". Minden helyen üzembe helyeztünk nagyméretű monitorokat, amik a többi helyet mutatják. Ez olyan, mint valami virtuális ablak két részleg között. A monitor előtt állva integethet a túlsó oldalon lévőknek. Láthatja, hogy ki van az asztalánál, vagy hogy ki kivel beszél éppen. Ez arra szolgál, hogy kialakuljon egy egészséges "mindenki közös célért dolgozik" érzés.
* Csereprogram, aminek kereteiben az egyes helyekről rendszeresen látogatást tehetnek az emberek más
helyekre.

Ezekkel a technikákkal felfegyverezve lassan de biztosan kialakítottunk egy jól működő rendszert a sprint planning meeting-ek, demók, visszatekintők, napi scrumok, stb bonyolítására földrajzilag elosztott csapatok esetén.

Mint minden esetben ez is a kísérletezésen múlik. Megfigyelés => Változtatás => Megfigyelés => Változtatás => Megfigyelés => Változtatás

Offshore

Több "offshore" csapatunk is van és többféle módszert is kipróbáltunk ezek hatékony kezelésére a Scrum keretein belül.

<figure>

Az első stratégia a csapatok szétválasztása, ami elég vonzónak tűnik. Ettől függetlenül mi a második stratégiával, a csapattagok szétválasztásával kezdtük, amire több indokunk is volt:
1. Azt szerettük volna, ha a csapattagok jól ismerik egymást.
2. Kiváló kommunikációs infrastruktúrát szerettünk volna látni a két helyszín között és megfelelő ösztönzést akartunk biztosítani a csapatoknak, hogy ezt maguktól kialakítsák.
3. Kezdetben az offshore csapat túl kicsi, hogy hatékonyan működjön, mint önálló Scrum csapat.
4. Szerettünk volna biztosítani az akadálytalan információ-megosztást legalább addig, amíg az offshore csapat önállóan is képes működni.

Hosszú távon elképzelhető, hogy a "szétválasztott csapat" stratégia felé fogunk elmozdulni.

Otthonról dolgozó csapattagok

Otthonról dolgozni néha nagyon hasznos tud lenni. Van úgy, hogy otthon egy nap alatt többet sikerül elérni, mint a munkahelyen egy teljes hét alatt. Legalábbis, ha nincsenek gyerekei :)

A Scrum egyik alapvetése azonban, hogy az egész csapat fizikailag egy helyen tartózkodik. Mit tegyünk ilyenkor?

Gyakorlatilag a csapatokra hagyjuk annak eldöntését, hogy mikor és mennyit lehessen otthonról dolgozni. Néhány tag gyakran dolgozik otthonról, mivel sokáig tart bejutni a munkahelyre. Ennek ellenére ösztönözzük a csapatokat, hogy az idő nagy részét együtt töltsék.

Ha valaki otthonról dolgozik, a napi scrumban Skype videokonferencián keresztül csatlakozik, továbbá
egész nap elérhetőek chaten, ami ugyan nem ugyanolyan, mintha fizikailag a szobában lennének, de teljesen megteszi.

Egyszer megpróbáltuk bevezetni "Fókusz Szerdákat", ami azt jelenti, hogy ha valaki otthonról szeretne dolgozni, az "rendben van, de csak szerdánként, ha csapattal előre egyeztetett". Ez elég jól működött. A csapat nagy része otthonról dolgozott szerdánként, amikor rengeteg feladatot sikerült befejezniük és ennek ellenére is elég összehangoltan dolgoztak. Mivel ez csak egy napot jelentett, a csapattagok szinkronban tudtak maradni egymással. Azonban ez valamilyen ismeretlen okból kifolyólag nem terjedt át más csapatokra.

Nagy általánosságban az otthonról dolgozó tagok sosem jelentettek nagy problémát számunkra.

17 Scrum Master checklista
==========

Ebben az utolsó fejezetben gyűjtöttem össze azokat a dolgokat, amelyek a mi Scrum Mastereink adminisztratív jellegű feladatait alkotják, mivel túl könnyű ezekről megfeledkezni. Az olyan triviális dolgokat, mint az "akadályok elhárítása" kihagytam a listáról.

A sprint kezdetén
---
* Készítsen Sprint Info oldalt a sprint planning meeting után
  o Linkelje be az oldalt a wiki főoldalán
  o Nyomtassa ki az oldalt és tegye ki a csapat falára
* Küldjön e-mailt az új sprint kezdetéről. Írja le a sprint célját és a sprint oldalára mutató linket
* Frissítse a sprint statisztikáit tartalmazó dokumentumot. Adja meg a becsült sebességet, a csapat méretét, a sprint hosszát, stb.

Minden nap
---
* Biztosítsa, hogy a napi scrum időben kezdődik és időben végződik
* A sprint menetrendjének betartása érdekében ha szükséges, módosítsa a sztorikat
  o Értesítse a Product Ownert ezekről a változásokról
* Győződjön meg arról, hogy a csapat naprakészen tartja a sprint backlogot és a burndown-t
* Győződjön meg arról, hogy az akadályok vagy ténylegesen elhárításra kerültek, vagy jelentve lettek a Product Ownernek vagy a fejlesztés vezetőjének

A sprint végén
---
* Bonyolítsa le a nyitott sprint demót
* Értesítsen mindenkit a demóról egy-két nappal a demó előtt
* Szervezze meg a sprint visszatekintőt a Product Owner és a teljes csapat részvételével. Hívja meg a fejlesztés vezetőjét is, hogy a tanulságokat továbbadhassa
* Frissítse a sprint statisztikáit tartalmazó dokumentumon. Adja meg a tényleges sebességet és a visszatekintő főbb kérdéseit

18 Zárszó
==========
Hű, sose gondoltam volna hogy eljutunk a végére.

Remélhetőleg ezen írás tartalmazott hasznos ötleteket az Ön számára, akár újonc, akár veterán a Scrum terén.

Mivel a Scrum-ot az egyes környezetekhez kell igazítani, elég nehéz "bevett gyakorlatokról" vitázni, de ettől függetlenül szívesen veszek mindenféle visszajelzést. Mondja el, hogyan különbözik az Ön gyakorlata az enyémtől. Adjon ötleteket a fejlesztésükhöz.

Nyugodtan írjon nekem a henrik.kniberg@crisp.se címen. Szemmel szoktam tartani a scrumdevelopment@yahoogroups.com levelezési listát is.

Ha tetszett ez a könyv, időről-időre nézzen fel a blogomra is. Remélhetőleg hasznos cikkeket talál

a Java-ról és az agilis szoftverfejlesztésről: http://blog.crisp.se/henrikkniberg/

És végezetül ne felejtse:
Ez csak egy állás.

Ajánlott Irodalom
==========
Íme néhány könyv, ami rengeteg ihletet és ötletet adott.

A szerzőről
==========
Henrik Kniberg (henrik.kniberg@crisp.se) a stockholmi Crisp (www.crisp.se) tanácsadója, aki Javára és Agilis fejlesztési módszertanokra specializálódik.

Az első XP könyvek és az agile manifesto megjelenése óta az agilis alapelvek által vezérelten a módszertanok különböző vállalatokban történő hatékony bevezetését kutatja. 1998 és 2003 között a Goyada társalapítója és CTO-jaként a cég technikai platformjának illetve harmincfős fejlesztői csapatának felépítése és igazgatása során számtalan alkalma nyílt a teszt-vezérelt fejlesztéssel és az agilis módszertanokkal való kísérletezésre.

2005 vége felé Henrik egy svéd játékfejlesztő cég fejlesztési vezető pozíciójába igazolt. A cég rengeteg sürgősen megoldandó szervezeti és technikai problémával küszködött és gyakorlatilag válsághelyzetben volt. Henrik a Scrum és XP használatával és az agilis és lean alapelvek egész cégre
kiterjedő bevezetésével segítette ki a céget ebből a krízisből.

2006 novemberének egyik péntekén Henrik lázasan feküdt ágyában, amikor az az ötlete támadt, hogy
feljegyezze magának az előző év tanulságait. Amint azonban írni kezdett, nem volt megállás: három napon keresztül eszeveszett tempóban írt és rajzolt, aminek eredményeképpen született meg egy 80 oldalas cikk "Scrum és XP a frontvonalról" címmel, ami aztán ennek a könyvnek az alapjául szolgált.

Henrik hozzáállása holisztikusnak mondható, szeret többféle szerepkört is kipróbálni a menedzsertől a fejlesztőn vagy épp tanáron át a trénerig. Szenvedélyesen segít a vállalatoknak kiváló szoftvert és kiváló csapatokat építeni mindig olyan szerepkörben, amire épp szükség van.

Henrik Tokióban nőtt fel és jelenleg feleségével és két gyermekükkel Stockholmban élnek. Szabadidejében zenét szerez és basszusgitáron vagy zongorán játszik helyi együttesekben.

További információk: http://www.crisp.se/henrik.kniberg