Agilis módszertan

Agilis módszertan

Amit nyújt az a contiuous iteration development and testing. A fejlesztési és tesztelési folyamatok lényegében egyszerre zajlanak egymásutáni kis ciklusokban.

Agilis szoftverfejlesztés

Az egyik legegyszerűbb és leghatékonyabb fejlesztési modell. Az agilis szót használják, amikor a fejlesztésben folyamatos tervezés, tanulás, fejlődés, csapatmunka és korai teljesítés a cél, vagy efelé akarunk menni. Lehetővé teszi és bátorítja a flexibilis reakciókat a változásokra.

Négy alapértéke van:

1. Egyéni és csoport kommunikáció és együttműködés a folyamatok és eszközök helyett
2. Működő szoftver a kiterjedt dokumentáció helyett
3. Együttműködés a megrendelővel a szerződésre támaszkodás helyett
4. Változásokra reakciók a hangsúlyos tervezés helyett

Összehasonlítás a vízesésmodellel

| Tulajdonság | Agilis | Vízesés |
| --- | --- | --- |
| Definíció | Inkrementális és iteratív megközelítése a szoftverfejlesztésnek | Szekvenciális elejétől a végéig megközelítése a szoftverfejlesztésnek |
| Folyamat | Többféle folyamatmodell (lást lejjebb) | Egyféle folyamatmodell |
| Delivery | Korai és gyakori lehetőségek arra, hogy a megrendelő megnézze a terméket és döntéseket hozzon, változtatásokat kérjen | A megrendelő először a folyamat végén látja a terméket |
| Strukturáltság | Kevésbé | Nagyon |
| Project típusok | Kis projectek gyorsan végig vihetők, nagy projectekre nehéz időt becsülni | Mindenféle méretű project végig vihető és esztimálható. |
| Hibajavítás | Project közben könnyen | Egyedül a végén tesztelünk, így pl ha követelmény hibát találunk, kezdhetjük elölről. |
| Fejlesztési folyamat | Iterációkban (2-4 hét), tervezés kevésbé hosszadalmas (lást alább) | A fejlesztési folyamat fázisokra van bontva, amik jellemzően nagyobban egy iterációnál. A fázisok egymás bemenetei és kimenetei. |
| Dokumentáció | Kevésbé hangsúlyos | Top prioritás. Dokumentáció használható oktatásra vagy a technológia felhasználására |
| Tesztelés | Minden iterációnak van tesztelés része. Ez lehetővé teszi a regresszió egyszerű felépítését | Szigorúan a fejlesztési fázis után. Később van funkcionális dolog a kezünkben. |
| Release | Elvileg minden iteráció végén lehet release-elni. Új funkciók használhatóak az új release-ben. Jó kommunikáció szükséges a megrendelőkkel. | Az összes funkció egyszerre kerül kiszállításra az implementáció után. |
| Developer-teszter | Együtt dolgoznak (egy csapatban is vannak) | Tesztelő és fejlesztő külön dolgozik (és gyakran másik csapatban) |
| UAT | Minden iterációban | A project végén |
| Követelménygyűjtés | Fejlesztők részt vesznek a követelmények gyűjtésében és a tervezésben | Fejlesztők nem vesznek részt a követelménygyűjtésben és tervezésben |

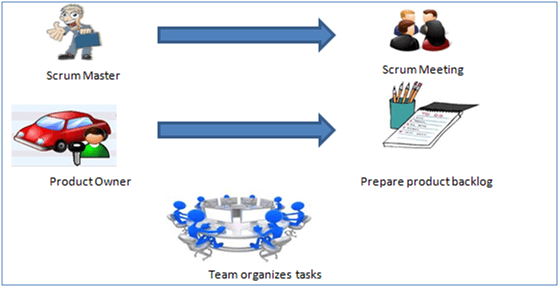
Folyamat modellek

* SCRUM
* Crystal
* DSDM (dynamic Software Development Method)
* FDD (Feature driven development)
* Lean
* Extreme programming

Scrum

A SCRUM egy agilis fejleszési módszertan (folyamat), ami arra koncentrál, hogy hogy lehet a feladatokat kezelni egy csoport alapú környezetben. Alapvetően SCRUM a rögbi meccsekből származik (időkérés). A csapat fejlesztésében és a csapatmunkában hisz (7-9 főre kell gondolni).

Következőkből áll:

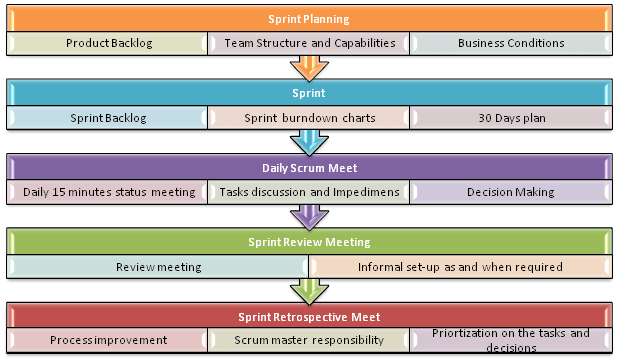


* Scrum master: A csapat működéséért és fejlesztéséért (pl csapat felépítése) felelős. A csapat meetingeket ő rakja össze és igyekszik megoldani a tagok problémáit, amik akadályozzák őket az iterációs célok elérésében.
* Product owner: Ő csinálja meg a product backlogot, priorizálja a feladatokat és felel az iteráció végi delivery-ért.
* Scrum team: A csapat szervezi a saját munkáját, azzal a céllal, hogy elkészüljenek a bevállalt dolgokkal a sprint végére.

Product backlog

Egy repo, ahol a követelményeket és elkészültségüket tudjuk követni. A követelmények user story-kban vannak kifejezve. A product owner tartja karban és priózza. A scrum csoport tudja olvasni és kérhet változtatásokat (követelmény változtatása, hozzáadása, törlése).

Scrum folyamat



Fejlesztői oldalról mi látszik:

* Az iterációkat nevezzük sprinteknek
* A product backlog egy lista arról, hogy mi szükséges még a végtermék létrehozásához
* Minden sprintben a legfontosabb felhasználói történetek a product backlogból átkerülnek a sprint backlogba
* A csapat dolgozik a sprint backlogon
* A csapat naponta egyeztet a napi munkáról
* A sprint végén a csapat kiszállítja a terméket az új funckionalitással

Metrikák

* Drag factor
  + Effort órákban, ami nem járul hozzá a sprint céljához
  + Lehet javítani a közösen használt erőforrások managelésével
  + Használjuk esztimációkban
* Velocity
  + A backlogból elégetett (kiszállítható funkcionalitássá változtatott) felhasználói történetek száma sprintenként
* Hozzáadott unit testek száma
* A napi build elkészítésének ideje
* Talált bugok iterációnként
* Nem megtalált bugok száma

Sokszor riportokban használt metrika a burn down chart és a velocity és a defect story-k valamilyen metrikája.

Agilis tesztelés

A vízesésmodellel szemben itt a tesztelés integráltan folyik a fejlesztés mellett. Az agilis tesztelés nem szekvenciális, hanem folyamatos.

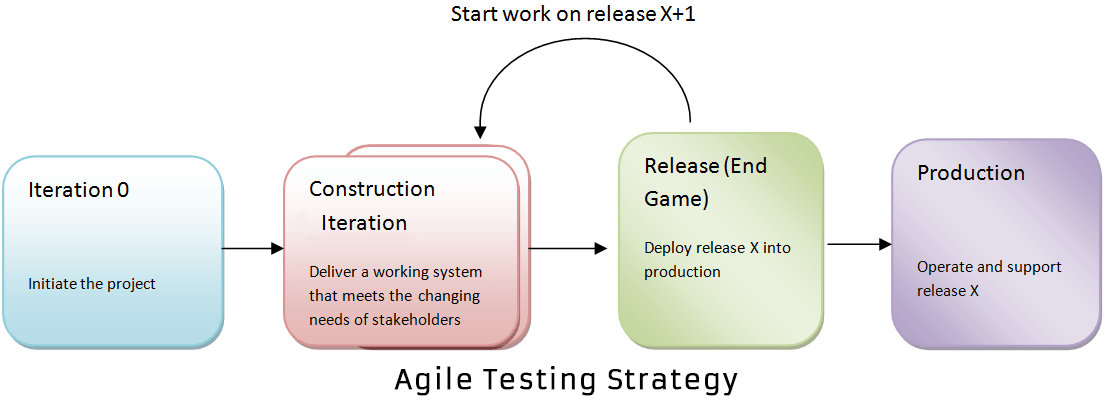
Test plan

Ebben van benne, ami ahhoz szükséges, hogy egy iteráción belül le tudjuk folytatni a tesztelést, pl információk a teszt adatokról, infrastruktúráról és teszt környezetről. Ezt minden iterációhoz megírjuk (szemben a vízesés modellével).

Példa felépítés:

1. Teszt scope
2. Új fukcionalitások, amiket tesztelünk
3. Teszt szintek vagy típusok a funkciók bonyolultságától függően
4. Terheléses és performancia tesztek
5. Infrastruktúra
6. Kockázatanalízis- és kezelés
7. Erőforrások
8. Mérföldkövek és termékek

Test stratégia



Nulladik iteráció

Ezek az elején szükséges feladatok. Megkeressük, hogy ki fog tesztelni, telepítjük a teszteléshez szükséges eszközöket és összerakjuk az ütemtervet az erőforrásokra (elvben).

1. A business case megértése
2. A project scope megértése (mit fogunk csinálni és mit nem)
3. Lényeges követelmények megértése és leírása, amik a designt és architektúrát meghatározzák (döntések felmérése és minél korábbi meghozása)
4. Architektúra (egy vagy több) meghatározása
5. Kockázatok felmérése
6. Költségek becslése és a project előkészítése

Fejlesztés iteráció

A második fázis az, ahol a tesztelés nagy része zajlik. A teljes megoldás egy részét implementáljuk egy vagy több iterációban. Erre használjuk a fent kifejtett metodológiák valamelyikét (vagy a keveréküket).

Ebben a fázisban a csapat a legfontosabb még meg nem valósított követelményeket implementálja.

Két részre bomlik itt a tesztelés, **confirmatory** (developer testing és agile acceptance testing) és **investigative** tesztelésre. Az első azt ellenőrzi, hogy a termék teljesíti e a megrendelő igényeit abban az állapotban ahogy azok velük meg lettek osztva (testing to verification). A második megkísérli azonosítani azokat a defekteket, amin az első átsiklott, vagy amivel nem foglalkozott. Itt azonosítják a potenciális problémákat és defect story-kat készítenek. Itt végzünk pl integrációs, load/stressz és security testinget.

Agile acceptance testing a tradícionális funkcionális tesztelés (pl unit tesztek és szolgáltatás integrációs tesztek) és a tradícionális UAT kombinációja. Utóbbit a csapat a stakeholderekkel közösen végzi.

Kibocsátás

Ennek a fázisnak a célja az, hogy a rendszert deploy-oljuk a felhasználási környezetébe. A feladatok például a következők lehetnek: végfelhasználók, fenntartók és működtetők képzése, marketing, rendszer- és felhasználói dokumentáció véglegesítése.

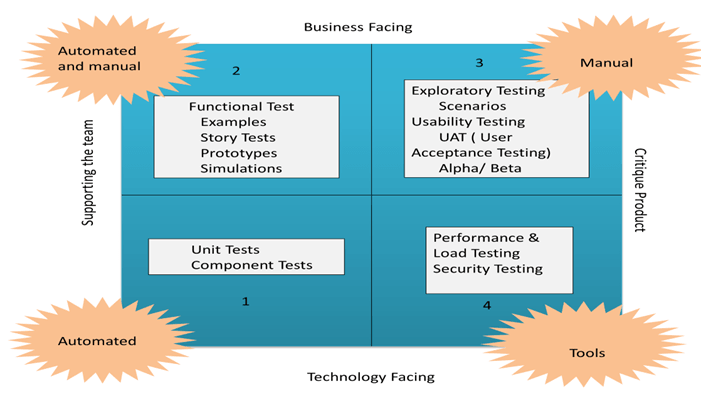
A final agile methodology testing fázisban teljes rendszer- és user acceptance tesztelést végzünk. Ezt alaposabban tesszük mint az előző fázisban. Itt a tesztelők a defect story-kra fókuszálnak.

Élő termék

Itt kerül ki a termék a felhasználási helyére.

Tesztek

Agilis metodológiában a tesztelést elkezdjük kb amikor csak lehet. Magas részvételt igényel stakeholder részről. A sprint végén létrejött kódnak elég jónak kell lennie a rendszertesztelésre. Lehetőség van kiterjedt regressziós tesztelés végzésére. Itt is kulcs a kommunikáció a csapaton belül.



Első negyed

Ennek a fókusza a belső kódminőség. Olyan teszteseteket írunk, amik inkább nyelv és technológia, mint a követelmények alapján íródnak azért, hogy támogassuk a fejlesztést.

* Unit tesztek
* Komponens tesztek

Második negyed

Itt olyan teszteket írunk, amik a követelményeket ellenőrzik, hogy támogassuk a csapatot.

* Példa esetek és végrehajtási útvonalak tesztelése (user scenario)
* Felhasználói élmény tesztelése pl prototípuson keresztül
* Pl lehet párban tesztelés

Harmadik negyed

Itt az előző két negyednek adunk visszajelzést. Ezek a teszt esetek képezhetik automatizáció alapját. Itt sok iterációt végezhetünk, hogy növeljük a bizalmat a termék minőségében.

* Usability testing
* Exploratory testing
* Felhasználó bevonásával párban tesztelés
* UAT

Negyedik negyed

Ez a rész a nemfunkcionális tesztelésre fókuszál.

* Stress testing
* Performance testing
* Infrastructure testing
* Data migration testing
* Scalability testing
* Load testing
* …

Nehézségek és kockázatok

* A hibázás kockázata nagyobb lehet a dokumentáció kisebb prioritása miatt
* Az új funkciók gyorsan vannak bevezetve ezért kevés az idő az ellenőrzésre a követelményekkel szemben (akár product, akár business)
* A tesztelő és fejlesztő szerep gyakran összecsúszik
* A teszt végrehajtásra korlátozott az idő
* A teszt plan megírására kevés az idő
* Regressziós tesztekre kevés az idő
* A követelmények változása itt igaz, hogy kezelve van, de változatlanul többlet munkát jelent, sokszor ez képviseli a legnagyobb nehézséget

Scrum tesztelés

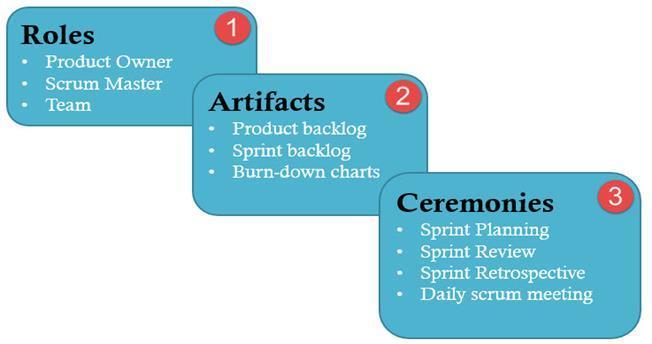
Ez a metodológia alkalmas arra, hogy összetett szoftvereket építsünk. Segít egyszerűbbekre bontani az egyébként bonyolult feladatokat. Átláthatóságot, alkalmazkodóképességet és betekintési lehetőséget nyújt, hogy elkerüljük, hogy belebonyolódjunk a feladatokba.

Eredetileg nincs dedikált tesztelő, a fejlesztők írják a unit teszteket, de kicsit is komplex feladatok esetében van tesztelő feladatkör.

Mi történik

A következők az alapvetései:

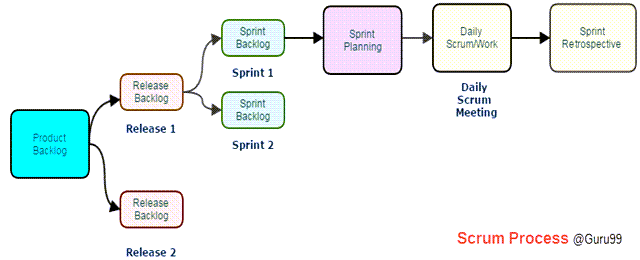
* Rövid, fix ciklusok vannak igazítható scope-al, ezeket nevezzük sprinteknek. Ezek segítik az alkalmazkodást a változó igényekhez (akár belső akár külső). Minden kiadásban lehet több sprint. Egy scrum project-ben lehet több kiadás is.
* Ismétlődő eseményeket használ: meetingek, eventek és milestone-ok.
* A fejlesztést user story-kba bontják fel, és ezek alapján vannak a kiadások (sprint és release)



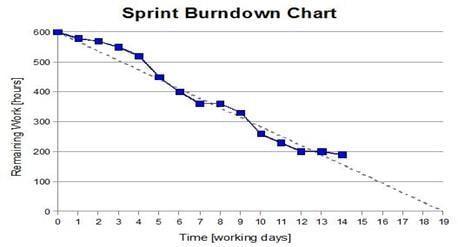
Feladatkörök

| Product owner | Scrum master | Csapat |
| --- | --- | --- |
| Ő definiálja a termék feature-eit | A csapatot manageli és figyel a produktivitásra | Általában 5-9 ember |
| Ő dönti el a kiadás idejét és a kiadandó feature-öket | Fenntartja a backlogot és igyekszik elhárítani az akadályokat | Fejlesztő, designer és tesztelő (meg ami még kell) |
| Ő priorizálja a feature-öket | Ő koordinálja a többieket | Megszervezi a saját munkáját és menetrendjét |
| Ő felel a termék értékéért | Ő védi meg a csapatot a külső behatásoktól (pl micormanagement) | Bármit megesznek azért, hogy a sprint céljait elérjék |
| Ő fogadja el vagy utasítja el a munka eredményét | Végzi a napi meetingeket, review-kat és planning-et | Aktívan részt vesznek a napi eseményeken |

Artifaktok



* User story: Rövid leírása annak, hogy mit kell tudnia a terméknek. Pl „Prémium fizethető az online rendszeren keresztül is” egy biztosítás adminisztráló rendszerben
* Product backlog: User story-k gyűjteménye (lást korábban)
* Release backlog: Azok a user story-k, amik belekerülnek az aktuális release-be.
* Sprint: (lásd korábban)
* Sprint backlog: User story-k, amiket a sprint alatt készít el a csapat. A csapathoz tartozik. A csapat tagok vállalkoznak ebből a munkákra. Naponta nézik, hogy mi van még hátra.
* Block list: A scrum masterhez tartozik. Valami miatt nem megvalósítható feladatok, vagy meg nem hozott döntések. Naponta nézik, mi van még rajta.
* Burndown chart: Ez jeleníti meg az elvégzett munkát.



Események

* Sprint planning: Először a release backlogból a sprint backlogba helyezünk user story-kat. A scrum master tartja. Itt becsüljük meg, hogy melyik feladat mennyi idő lesz.
* Daily scrum: A csapattagok elmondják, mivel foglalkoztak az előző nap, mit terveztek aznap csilnálni és milyen akadályok vannak számukra. A scrum master tartja. Itt követik a haladást sprint szinten.
* Sprint review/Retro(spective): 2-4 óra, ami arról szól, hogy mit hoztunk létre és milyen tanulságokat tudunk levonni. A scrum master tartja.

Mit csinál a tesztelő

Sprint planning

* Kiválasztja, melyik user story tesztelésében vesz részt
* Részt vesz annak becslésében, hogy melyik user story tesztelése mennyi időt fog igénybe venni
* Részt vesz a priózásban
* Tisztába kerül a sprint célokkal

Daily scrum

* Támogatja a unit tesztelést
* Leteszteli a user story-t, amikor elkészül. Ezt lehet a fejlesztővel közösen. A defekteket logolják a megbeszélt helyre, ahol a napi meetingeken tudják követni és analizálni őket. Megoldott defekteteket újratesztelünk.
* Részt vesz a napi meetingeken
* Átviszi az előző sprintben nem teljesített itemeket a következőre (ez inkább általános)
* Ő írja az automatizáló scripteket. Ő illeszti be őket a CI-ba.
* Megvizsgálja a CI eredményeket és reportokat küld a stakeholdereknek
* Nemfunkcionális teszteket végez
* Együttműködik a product ownerrel és a megrendelővel az acceptance criteria-k megírásában az UAT-hoz
* A sprint végén a tesztelő részt vesz az UAT-ban (vagy ő végzi) és megállapítja a teszteltség állapotát (completeness).

Sprint review/Retro(spective)

* Összegzi, hogy mi ment jól és mi ment rosszul a sprintben
* Összegzi, milyen tanulságokat lehet levonni és milyen best practice-eket lehet érdemes adaptálni

Automatizálás

Further reading: https://www.guru99.com/automation-testing-agile-scrum.html