# SDLC

Software development life cycle.

Egy folyamat, amely több tervezett lépésből áll, amivel fejlesztünk vagy módosítunk szoftvereket. Főbb részei a tervezés (design) a fejlesztés és a tesztelés. A cél magas minőségű szoftver létrehozása, ami eléri, vagy túlteljesíti az ügyfél (customer) igényeit, illetve befejeződik bizonyos határidőn belül és költség alatt.

Miért life cycle.

A szervezetnek ez az előre megtervezett leírása arra, hogy milyen módon fogja fejleszteni, deploy-olni, fenntartani, megváltoztatni, kiegészíteni vagy lecserélni a szoftvert. A ránk leginkább vonatkozó része a szoftver és a fejlesztés minőségének biztosítása és emelése.

ISO/IEC 12207 egy nemzetközi szabvány. Ennek a célja, hogy standardizálja a szükséges lépéseket a szoftver fejlesztésére és fenntartására (maintenance). Ennek kell megfelelni egy SDLC-nek, amit használ egy cég. A minőségirányítás (QA vagy Quality Assurance) foglalkozik azzal, hogy a folyamatok megfeleljenek a szabványoknak, ami alapján a fejlesztés folyik. Amikor ilyet láttok, hogy CE vagy TÜV, akkor az annak az eredménye, hogy egy szabvány testület auditálta a céget és a terméket és megfelelőnek minősítette a jelzésének a használatára. Innen tudja egy mezei vásárló, vagy egy viszonteladó, hogy mire számíthat egy terméktől.

Szóval egy konkrét fejlesztési modell lehet több féle is:

* Vízesés modell (Waterfall)
* Iteratív modell
* V modell
* Agilis (Agile)

Főbb részei:

* Tervezés (planning)  
    
  Baromi lényeges, erre épül minden.   
  Senior csapattagok végzik (5+ év tapasztalat).   
  Információ kell az ügyféltől, az eladástól, piacról (kutatás vagy hozzáértő), illetve a mérnöki terület szakértőitől (domain expert). Itt az expert nem azt jelenti, mint az e-mail-ekben. Ez tényleg expert.   
  Létrejön egy project terv és egy megvalósíthatósági tanulmány (feasibility study) gazdasági, működtetési és technológiai téren.   
  Itt van megtervezve, hogy a minőségbiztosítás hogy tudja elérni a megkövetelt szintet.  
  Itt azonosítják be az előre látható kockázatokat, amiket kezelni kell.   
  Kijelölnek lehetséges technikai megközelítéseket, amiket követni lehet, hogy a project-et sikeresen végre tudják hajtani a kockázatok minimalizálása mellett.   
    
  Veszély (hazard): ami képes kárt okozni (személyi sérülés, gazdasági károk, termék vagy vállalat jó hírének a sérülése)  
  Kockázat (risk): kár bekövetkezésének a valószínűsége (magas, közepes, alacsony)  
  Kockázatkezelés (risk management): valamelyiket csökkentjük a kettő közül. Csökkenthetjük a bekövetkezés valószínűségét pl oktatással. Csökkenthetjük a veszélyt úgy, hogy pl a kár bekövetkezésének esetére valamit odateszünk (poroltó) vagy áttervezzük az architektúránkat (pl biztonsági öv).
* Követelmények leírása   
    
  Ezen a ponton fogalmazzuk meg, írjuk le és íratjuk alá az ügyféllel vagy a piacszakértővel (ha nincs ügyfél) a termékkövetelményeket. Ennek lesz a terméke az PRS (product requirement specification). Ebben vannak benne a követelmények, amiket használunk a tervezéshez és fejlesztéshez. a termék életciklusa alatt.
* Design

Ezen a ponton a mérnökök az SRS-t alapul véve elkészítenek legalább egy architektúrát ami alapján a termék fel lesz építve. Ezek a DDS-ek (design document specification). Ezeket reivew-zzák a stakeholderek (pl kockázatfelmérés, robusztusság, modularitás, költség, munkaóra (effort) igény) és kiválasztják a nekik legjobban tetszőt.   
A végső DDS pontosan leírja a termék alkotórészeit (modulok), a köztük lévő kommunikációt, a modulok és a külvilág közötti kommunikációt (hardware, 3rd party software, user stb) valamint a modulok belső felépítését.

* Kódolás  
    
  A kód írása történik a DDS szerint.  
  Kódoláshoz guide-okat ír a fejlesztés, hogy egységes (és ezért az írón kívül másoknak is olvasható) és hatékony kód szülessen. Érdemes egységesíteni a használt fejlesztői környezeteket és toolokat (pl IDE integrated development enviroment). A nyelv kiválasztása érdekes kérdés. Elvileg minden feladathoz van egy legjobban illeszkedő nyelv. Viszont vannak tényezők, mint a dolgozók tudása, meglévő környezet, korábbi modulok nyelve, ami befolyásolja a döntést. Emellett a legtöbb programozónak van egy preferált nyelve, amit használ.
* Tesztelés  
    
  Fontos megjegyezni, hogy a tesztelés minden lépésnek része (pl követelmények tesztelhetőségének vizsgálata, legjobb architektúra kiválasztása, kódolás közben debugolás stb, ügyfél által észrevett hibák kezelése).   
  Viszont a szoftver életciklusának vannak különböző részei, ahol a defekteket reportoljuk, nyomon követjük, kijavítjuk, és újra teszteljük, amíg a kívánt minőséget elérjük (vagy a határidőt, de erről később)  
  Fontos még megjegyezni, hogy a már kiadott (és ügyfélnél lévő) termék tesztelése ettől elkülönül.
* Deployment

Tesztelés után formálisan kibocsátjuk (release) a terméket. Ez az üzleti stratégiának megfelelően történhet lépésekben is. Ezután a fenntartás (maintenance) fázis következik.  
Itt fontos megjegyezni, hogy a tesztelés stratégiájától, a management-től és az ügyféllel való kommunikációtól függően a visszajelzések különböző szinten tesztelésnek tekinthetők (lásd béta tesztelés, vagy on site testing (god help us…)).

Tapasztalatom a DDS-ről:

Kis cég (esetemben outsourced tesztelés):

Annak az oka, hogy az ügyfél ki tudta szervezni a tesztelést, az, hogy a dokumentációs szintjük elégséges volt ahhoz, hogy a tesztelő cég végig tudjon vinni egy tesztelési ciklust elfogadható idő alatt.

Nagyvállalat:

Jó definiált dokumentációs szintek voltak SRS-el, DDS-el, teszt tervekkel, ezeknek a felelőseivel és a tesztek futtatására kiépített rendszerrel. Ezt többek között a nagy létszám és a stabil piaci háttér tette lehetővé (előfordultak 3-6 hónapos késések release-eknél „a tesztelés miatt”). A jól definiált szintek viszont arra vezettek, hogy a korábban lévő teszt csoportok (és managementjük) „megbízott” a későbbi csapatokban, hogy ők majd elfogják az esetleges hibákat.

Közepes cég:  
Úgy viselkednek, amíg csak tudnak, hogy a kódolás az egyetlen lényeges lépése a fejlesztésnek. Marha nagyot lehet bukni ezzel a szemlélettel. A DDS írást szeretik minimalizálni a fejlesztésen. Management nem akar vele foglalkozni, mert nagy effortot igényel (ez önmagában igaz). Senior fejlesztő nem akar vele foglalkozni, mert (érthető módon) nem akar dokumentálni, kódolni akar. Junior fejlesztő pedig valószínűleg nem rendelkezik akkora domain tudással, hogy meg tudjon írni egy DDS-t.

Stakeholder:

* ügyfél
* felhasználó
* ISO/IEC 12207
* QA
* fejlesztő
* tesztelő

Vízesés modell (waterfall)  
  


Lineáris, szigorúan egymás után következnek a szakaszok.   
A legkorábbi modell, egyszerű megérteni és használni.   
Ebben a modellben egy szakasz szigorúan csak akkor kezdődhet el, hogyha az előző befejeződött. Átfedés sincs.

* Követelmény gyűjtés és elemzés  
    
  Az összes lehetségesen felmerülő követelményt felveszik és dokumentálják (PRS).
* Rendszer tervezés  
    
  Elkészül egy magas szintű rendszer terv.   
  Ez és a PRS alapján elkészül az architektúra, és a rendszer követelmények (SRS) (hardware és software).
* Implementáció  
    
  Ebben a modellben az SRS alapján kis alkotórészekből elkezdik megírni a szoftvert. Ezek a unit-ok vagy egységek (esetleg modulok). Ezeknek a tesztje a unit vagy egység teszt.
* Integráció és tesztelés  
    
  Miután a unit-ok elkészültek, ezeket összerakják (integráció). Így áll össze a rendszer (system). Illesztés különböző állomásainál mennek az integrációs tesztek.
* Deployment  
    
  Miután a funkcionális és nemfunkcionális tesztek lementek, a terméket kiviszik a felhasználási helyére vagy piacra dobják.
* Fenntartás  
    
  A felhasználás közben fellépő bug-okra a javítások lesznek a patch-ek.   
  Következő, javított verziókat új release-eket lehet készíteni.

A bugok dokumentálása (pl bejelentések, megkeresések, user portal, periodikus találkozók, fórumok, JIRA) és a javítások piacra vitele vagy az ügyfélhez eljuttatása a maintenance.

Felhasználás

* A követelmények jól dokumentáltak, tiszták és nem változnak
* A termék (felé támasztott igények) nem változnak
* A technológia kiforrott és nem változékony
* Nincsenek nem tisztázott követelmények (nem olyan egyszerű ezt előre látni)
* A project rövid

Előnyök  
Jól lehet munkacsoportra bontani a fejlesztést és jól irányítható. Tiszta napirend és határidők adhatók hamar.   
Koncepció -> tervezet -> kódolás -> tesztelés -> telepítés -> hibaelhárítás-> működés -> fenntartás

* Egyszerű megérteni és használni
* Egyszerű managelni, mert merev. Mindegyik fázisnak jól értelmezhető terméke és review része van.
* Egyszerre egy fázis megy
* Jól működik kis projectekre, ahol a követelmények jól le vannak írva
* Egyszerű megérteni, hogy hol vagyunk éppen (milestone-ok)
* Egyszerű task-okat kiírni (írd le, teszteld le, review-zd le)
* A folyamat és az eredmények jól dokumentáltak (audithoz nem árt)

Hátrányok  
Utólag nehéz felülvizsgálni a lépéseket. A tesztelés fázis után nagyon nehéz visszamenni és változtatni. Ha dokumentálva volt és ha nem, akkor is.

* Nincs működőképes szoftver kéznél a végéig
* Nem jó modell bonyolult és objektum orientált project-ekhez (pl JAVA)
* Rossz modell hosszú vagy már folyamatban lévő projeck-ekhez
* Nem használható hatékonyan, ha a követelmények nem stabilak. Minél kevésbé, annál magasabb a kockázat.
* Amíg nem áll elő kimenet egy fázison belül, nehéz mérni a haladást.
* A scope változás katasztrofikus lehet
* Big bang integrációs megközelítés megakadályozza a gyors visszacsatolást

Kérdések:

* Milyen különböző feladatköröket (role), vagy pozíciókat tudunk azonosítani a modellből?
* Különböző részekben próbáljuk meg beazonosítani, hogy milyen bemenetet adhat a tesztelés!
* Azt is próbáljuk meg meglátni, hogy melyik lépésnek milyen hatása lehet a tesztelésre!
* Mik lehetnek a funkcionális és nem funkcionális tesztek?