YAZILIM YAŞAM DÖNGÜSÜ (SDLC) VE MODELLERİ

Özet olarak sırası ile:

Yazılım Yaşam Döngüsü

Yazılım Yaşam Döngüsü Modelleri

Agile metodolojisi

Konu başlıkları anlatılacak.

Yazılım Yaşam Döngüsü Nedir?

Bir işe başlamak genelde o işin en zor kısımlarından birisidir. Hele de o işe nereden başlayacağını bilmiyorsan. Peki gerçekten bir işe nereden başlayacağız? Diyelim başladık sonrasında nasıl devam ettireceğiz? Ne zaman kontrol edeceğiz? Hata varsa ne yapacağız?..

Bu ve buna benzer birçok soru gün geçtikçe yazılımcıların zihnini meşgul etti; sadece meşgul etmekle de kalmadı bu soruların cevaplanmamış ve belli bir prosese bağlanmamış olması projelerde çok büyük aksaklıklara da sebep oldu. Zamanla birlikte gelen birikimse SDLC'yi doğurdu.

İlk zamanlarda yazılımcılar kodlarını gelişigüzel bir şekilde yazıyordu. Bu durum ise spagetti kod dediğimiz anlaşılması ve düzeltilmesi bırakın başkasını, kodun yazarı için bile neredeyse imkansız kodların oluşmasına sebep oldu. Bundan sonra ise bir barok model var ama SDLC modellerini sonra detaylandıracağım burayı atladığımızda ise "Waterfall" modeli ortaya çıkmıştır ve bu modelle birlikte SDLC'nin yapıtaşı olan

Planlama (Planning/ Gereksinim), Analiz (Analysis), Tasarım (Design), Gerçekleştirme (Implementation), Bakım (Maintenance) kavramları oluştu. Bu kavramlar SDLC'nin yapı taşı sayılabilecek kavramlar olarak yazılımcıların sözlüğüne girdive waterfall modelinden sonraki tüm modellerde de kullanıldı. Hatta biraz extrem bir tabir ile diğer tüm modellerin waterfall uyarlaması olduğu da söylenebilir. Peki bu kavramlar nedir ve bu kavramlar ne anlama geliyor?

Planlama: Yazılım yaşam döngüsünün ilk aşamasıdır. Temel ihtiyaçlar belirlenir, proje için **fizibilite** çalışmaları yapılır (maliyetlerin ve sistemin yararlarının tanımlanması) ve proje planlaması gerçekleştirilir

Analiz: Bu aşamanın amacı sistemin işlevlerini ve kesin gereksinimleri açıklığa kavuşturmak ve sonucunda bunları belirli bir formatta **dokümante** etmektir. Bu çalışma müşteri, yazılım mühendisi, sistem analisti, iş analisti, ürün yöneticisi vb. rollerin bir araya geldiği gruplar tarafından yapılabilir. İhtiyaçların net olmadığı durumlarda yazılım mühendisi ve müşteri arasında iletişim ve birlikte çalışmanın çok daha fazla olması gerekir. Çeşitli yazılım geliştirme metodolojilerinde bu aşamada kullanım dokümanları ve test plan dokümanları da oluşturulabilir.

Tasarım: Gereksinimler belirlendikten sonra bu gereksinimlere uygun olacak şekilde Tasarım yapılır. Tasarımda ürünün; özellikleri, yetenekleri ve arayüzleri belirlenir. Iki tür tasarımdan bahsedebiliriz.

- Mantıksal Tasarım: Önerilen sistemin yapısını anlatır. Olası örgütsel değişiklere önerilir.
- Fiziksel Tasarım: Yazılımı içeren bileşenler ve bunların ayrıntılarını içerir.

Burada **Soyutlama** kavramından da bahsetmeliyiz. Yazılım tasarımında kullanılan en önemli tekniklerden birisi **Soyutlama (Abstraction)** dır. Soyutlama, problemlerin çözümlerini kolaylaştırmak için nesnelerin, olayların ve durumların bazı özelliklerin görmezden gelinmesidir. Problemi basitleştirerek en önemli kısımlarına odaklanmamızı sağlar. **Modelleme** is temel tasarım aracı olup statik ve dinamik modellerden bahsetmekten mümkündür. Statik model, programın çalışması sırasında değişmeyen yönlerini ifade etmek için kullanılırken

(sınıf ve nesne modelleri), dinamik model programın çalışması sırasındaki işleyişi ifade etmek için kullanılır (durum ve sıra diyagramları).

Gerçekleştirme: Tasarım aşamasının belirli bir olgunluğa ulaşmasıyla birlikte **Kodlama** aşaması başlar. Müşteriye teslim edilecek ürünü programlama aşamasıdır. Kodlama süresince ve kodlama sonrasında yapılan diğer önemli aşama **test'tir**. Erken test et yaklaşımı ile (**early testing**) hareket edip, analiz aşamasından itibaren test bakış açısına sahip olmamız hata yapma oranımızı ve maliyetleri (zaman, para, prestij vb.) düşürecektir. Birim testleri, duman testleri, yanlış değer testleri, kabul testleri, kullanım senaryo testleri, yük testleri, kullanıcı kabul testi, yoldan geçen adam testi, test otomasyonu gibi sürece ve duruma göre uygulanabilecek çok farklı kategoride ve derinlikte test türü bulunmaktadır.

Teslim ve Bakım: Tüm testler gerçekleştikten sonra ürün sahaya çıkabilir hale gelir ve ürün teslim edilir. Teslim ederken yalnızca ürün verilmez. Ürün ile birlikte kullanıcı kılavuzu ve versiiyon fark dökümanı da verilmelidir. Teslim ile birlikte bakım aşaması da başlar. Hata giderici, önleyici, altyapıyı iyileştirici, ürüne yeni özellikler ekletici gibi farklı şekillerde bakımı yapılabilir.

SDLC'nin temel adımları Core Processes olarak ta adlandırılır. Kimi kaynaklarda madde sayısı 4'e kadar indirilirken kimi kaynaklarda ise 10'a kadar çıkartılabilinir. Ben ana olarak sayılabilecek kısımları aldım. Örnek olarak sağdaki yer alan şekile bakılabilir. Bu süreçleri gerçekleştirmek için Yazılım Belirtim Yöntemleri ve Yazılım Süreç Modelleri kullanılır. Peki nedir bu terimler?

Yazılım Belirtim Yöntemleri (Specifications):

Core Processes'e ait fonksyonları yerine getirmek amacı ile kullanan yöntemlerin tümüdür. Süreç akışı için Nesne/Sınıf Şemaları,Veri Akış Şemaları, Yapısal Şemaları kullanılabilir. Süreçler tarafından kullanılan verilerin tanımlanması için ise Nesne İlişki Modeli, Veri Tabanı Tabloları, Veri Sözlüğü kullanılabilir. Süreçlerin iç işleyişini göstermek için ise Düz Metin, Algoritma, Karar Tabloları, Karar Ağaçları, Anlatım Dili kullanılabilir.

Yazılım Süreç Modelleri (Processes):

SDLC'de aşamaların nasıl yürüyeceğine açıklık getiren, işleri düzene koyan ve sırasını belirleyen proseslerdir. Bu prosesler sayesinde karmaşıklık ve krizler önlenmiş olur. Başlıca SDLC modelleri şöyledir:

- Gelişigüzel Model
- Şelale Modeli (Waterfall Model)
- V Modeli (V-shaped Model)
- Spiral Model
- Formal Sistem Geliştirme (Formal System Development)
- Yeniden kullanıma yönelik geliştirme (Re-use based development)
- Artımlı Geliştirme (Incremental Development)
- Birleşik Süreç (Unified Process)
- Çevik Modeller (Agile models: XP, Scrum)

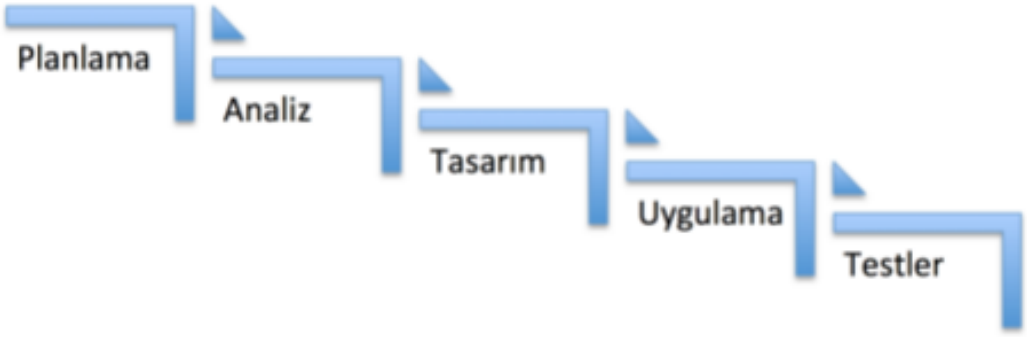
Şimdi birkaçını açıklayalım

Gelişigüzel Model: Bu model (ne kadar model denirse) tamamen gelişigüzel bir şekilde başlanır devam ettirilir. Yani belli bir kuralı yoktur. Bu çok karmaşık kodlara (spagetti kod) sebep olur. Profesyonelliğe uymaz. Bu yöntemi okulda yapılan başlangıç projelerde görebilirsiniz.

Barok Model: Bu modelde ise adımlar belirlenmiş ve doğrusal bir şekilde uygulanmış fakat dökümantasyonu en son ayrı olarak yapılan bir süreç olduğundan günümüzde uygulanan bir model değildir.

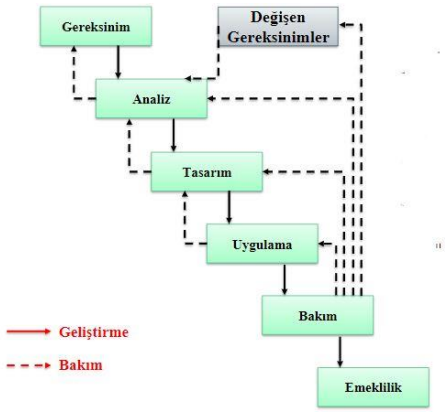
Waterfall(Şelale) Modeli:

Esas olarak şekillenmiş ilk model olduğunu söyleyebiliriz. Çok statik bir modeldir. Çok statik yapısı güncel projelere uygun değildir. Bu yüzden günümüzde nerdeyse hiç kullanılmaz. Askeri projelerde zaten talep sabit olduğu ve bü yüzden waterfall modelinin kullanılabileceği söylense de günümüzde çeviik model çok daha fazla kullanılmaktadır. **Planlama** ile başlar ve ardından analiz edilir. **Analizde** genelde iki adımda uygulanır. **Durum analizi** ile kaynaklar sorgulanır, hangi işletim sistemi kullanılacak vb. Analizler yapılır. **İstek analizi** ile ise ne istiyoruz sorusuna yanıt bulunur. **Tasarım** aşamasında ise analiz kısmındaki veriler ışığında ekran ve fonksiyonlar tasarlanır. Tasarım aşaması biraz geniş bir aşamadır. Ayrıntılı bir aşamadır. Nesne yönelimli ortamda kullanılacak bir yazılımda use case'lerinin çizilmesi, class diagramları'nın hazırlanmasına kadar giden tasarım aşamaları vardır. Daha sonra **Uygulamaya** aşamasına geçilir. Yazılım projelerinde, uygulamayı (implementation) kodlama olarak düşünebiliriz. Geliştirme (Development) aşamasıdır ve burada bir sürü alt aşamasından bahsedilebilir. Uygulama aşamasındaki testler, uygulama aşamasındaki tasarım yanlışları, analiz yanlışlarına geri dönülmesi. Sonra sistemin genel olarak test edilmesi ve müşteriye teslim edilmesi aşamaları görülebilir.



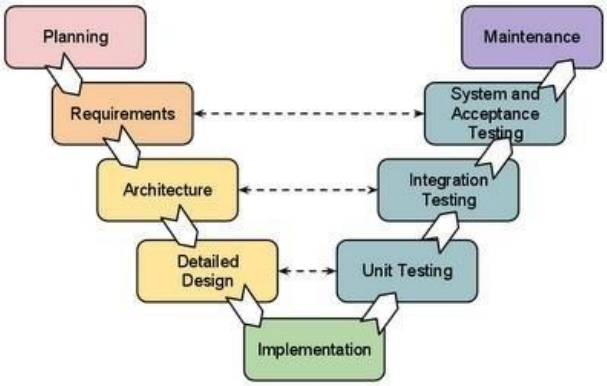
Planlama ile başlayıp testlere ulaşana kadar sisteme çok dokunma şansımız yoktur. Ancak yeni bir şey yapılacaksa tekrar planlamaya gelinip yeni bir proje olarak ele alınabilir. Uzun süren projeler için çok ciddi bir tehdittir. Çünkü bir proje 1-2 yıl sürüyorsa şayet 1-2 yıl boyunca bu projenin hangi aşamasında ise o aşamasının bitmesini beklemek zorundayızdır. Müdahale etme şansımız yoktur. Bilişim projeleri için çok uygun bir yönetim sistemi değildir. Ama en basit, en etkili ve hepsinin temeli olan yöntem olduğu için bilmekte fayda vardır. Küçük projeler için uygulanabilir. Hangi aşamada hata var, kimden problem kaynaklanıyor veya nerede problemi çözebiliriz gibi bilgileri verir. Genelde burada geri dönüşler de söz konusudur. Tasarımda bir problem olduğunda analize, uygulamada bir problem olduğunda tasarıma dönülmesi gibi geri dönüşler yapılabilir. Hataların maliyeti de giderek artar. Analiz aşamasındaki bir hatanın düzeltilmesi çok düşük maliyeti olurken, bu problem test aşamasında yakalanırsa tekrar analize dönülüp bunun telafisi tekrar bütün süreçlerin sıfırdan ele alınmasını gerektirir ve maliyet artışını beraberinde getirir!

Fıskiye Modeli: Fıskiye modeli şelale modelinden öykünerek oluşturulmuş bir modeldir. Henderson-Sellers and Edwards tarafından geliştirilmiş bir modeldir. Şelale modelinden farkı, döngüleri vardır. Döngüler şeklinde her bir aşamada belli döngüler elde edilir. Tasarım aşamasında tekrar koddan tasarıma dönülmesi, tasarımdan istek analizlerine geri dönülmesi ve operasyona geçtikten sonra testlere geri dönülmesi gibi döngülere sahiptir.x



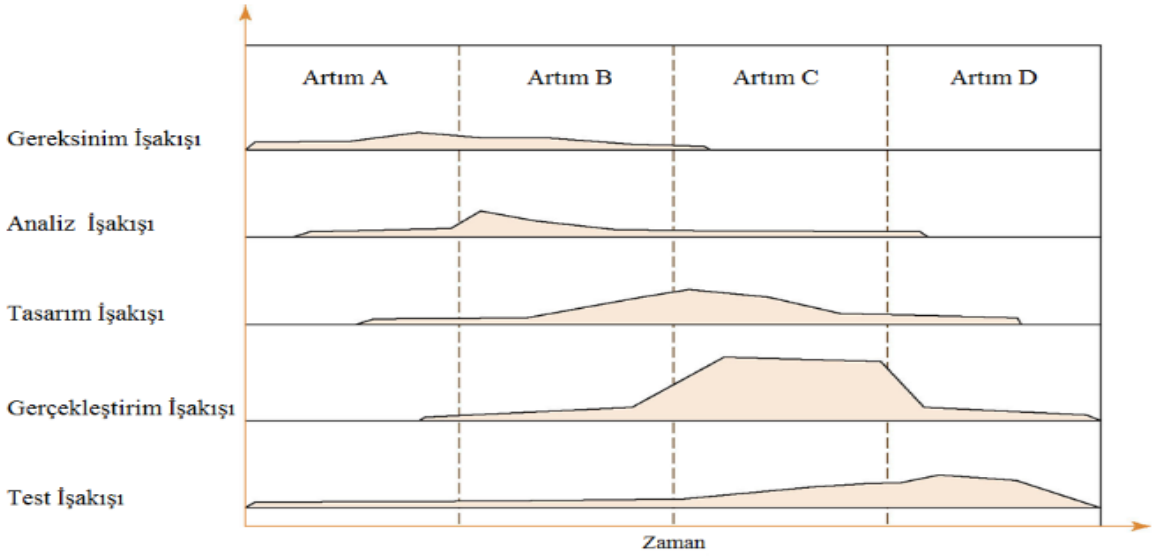
V-Şeklinde Model (V-Shaped Model)

İlk yarısı şelale modeline çok benzerdir. Farklılık olarak üst seviye tasarımların daha sonra daha detaylı alt seviye tasarımlarının yapıldığı görülür. Birim testleri yapılır. Her üretilen modülün testi yapılır. Daha sonra bu modüllerin/alt seviye tasarım ürünlerinin birbiri ile çalışma durumu, entegrasyon testleri yapılır ve kabul testleri aşamasında müşterinin kabul edip etmemesi ile ilgili müşteriye gidilir. Müşteri uygulamayı test eder. Sonra bakım başlar. Başka bir ayrıntı da şekilde de görüleceği gibi aynı hizada olan aşamalar birbirini karşılar. Detaylı tasarım yapıldı. Bunun hemen karşısında kodlama bittikten sonra birim testleri yapılır. Yapmış olduğumuz tasarıma ait testler yapılır. Üst seviye tasarıma ait entegrasyon testleri yapılır. Her bir alt modülün birbiri ile uyumlu çalışıp çalışmadığı test edilir. İhtiyaç analizleri müşteriden istenir ve nelere ihtiyaç olunduğu çıkarılır. İhtiyaçların ise kabul testleri yapılır. "Evet o ihtiyaçlar sağlandı ve tamamdır, uygundur" şeklinde onunla ilgili test senaryoları hazırlanır. Planlamanın da bakım aşamasında karşılığını görüyoruz. Yaptığımız planlamaya uygun mu? Bakım aşamasından gelen bilgiler ile bu sorunun cevapları aranır. V-Şeklinde model aslında biraz daha alta inip yukarı çıkan ve her aşamasının karşılığının olduğu bir modeldir. "Bir şey yapıldı ama bunun karşılığı doğru mudur?" sorgulamasının yapıldığı, en alta kadar inip kodlamaya kadar inip sonra da yukarıya doğru sistemi taşıyan bir yapıya sahiptir.



Arttırımlı Model (Incremental Model)

Bu modelde Sistemi bir kerede yapıp teslim etmek yerine geliştirme ve teslim parçalara bölünür. Her teslimde beklenen fonksiyonların bir kısmını karşılar. Bu modelde kullanıcı gereksinimleri ön plana alınmıştır ve öncelikli gereksinimler erken teslim edilir. Gereksinimler önem sırasına göre her yinelemede bir kısmı tamamlanır. Bir kısmın geliştirmesi başlandığında o parçanın gereksinimleri dondurulur. Olası değişiklikler sonraki gerçekleştirme sürecinde gerçekleştirilir. Proje parça parça teslim edildiğinden somut veriler oluşur ve eksiklik yada yanlışlıklar daha önceden fark edilir. Ayrıca projenin ortaya çıkması için her şeyin bitmesi beklenmediğinden büyük, uzun süreli ve eksik işlevsellikle çalışabilecek projeler için uygun olabilir. Bu yöntem sayesinde bir taraftan üretim, aynı zamanda bir taraftan da kullanım yapılabilir.



Kodla ve Düzelt Modeli: Çok masraflıdır. Genelde küçük şirketler kullanır. Bir hataya veya eksiği başlangıçta farkedip düzeltmek maaliyeti çok düşürür. Bu modelde ise hata en son düzeltildiğindenc maaliyet inanılmaz artar.

Eveet gelelim en önemli ve güncelde en çok kullanılan metodolojiye. Şu ana kadarki yöntemler belki farklı sektörlerde kullanılabilse de yazılıma pek uygun değildi. Peki yazılım için en uygun metodoloji nedir?

Agile (Çevik) Yazılım Geliştirme

Çevik modelin en iyi yönü değişimlere çok çabuk adapte olabilmesidir. Ayrıca hızlı olması ve sürekli müşteri ile temasta olması da büyük artı sağlar. Bu model verimliliği yüksek, esnek, hata oranı düşük, hızlı ve ucuz çözümler sunar. Çevik yazılım geliştirme metodları özünde aynı ama uygulamada bazı farklılıkları olan modellerdir.

Bu yöntemde projenin büyüklüğü ne olursa olsun küçükyinelemelere ayrılır ve her yineleme kendi başına bir proje gibi ele alınarak geliştirilir. Her yineleme (iteration) sonunda da müşteriye proje hakkında bilgi verilir. Her yineleme 2-4 hafta sürmesi planlanır. Çevikte hassas nokta iletişimdir. Ayrıca küçük parçalardan oluşması da hataları daha kolay tespit edip düzeltmeye yardımcı olur.

Agile Manifestosu: 2001 yılında yayınlanmıştır. Bu manifestoda Süreçler ve

- Araçlar Yerine Bireyler Ve Etkileşimler,
- Kapsamlı Belgeler Yerine Çalışan Yazılım,
- Sözleşme Görüşmeleri Yerine Müşteri İlişkileri,
- Plan İzleme Yerine Değişikliğe Açıklığın,

Daha önemli Ve öncelikli Olduğu belirtilmektedir.

Çevik manifestonun prensipleri şunlardır:

- En önemli önceliğimiz değerli yazılımın erken ve devamlı teslimini sağlayarak müşterileri memnun etmektir.
- Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir. Agile süreçler değişimi müşterinin rekabet avantajı için kullanır.
- ***Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır.***
- İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdırlar.
- Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır.
- Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüzyüze iletişimdir.
- Çalışan yazılım ilerlemenin birincil ölçüsüdür.
- Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.
- Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır.
- Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca arttırılması sanatı, olmazsa olmazlardandır.
- En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar.
- Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler.

En çok kullanılan agile metodlar:

XP(Extrem programing)

SCRUM

Agile Unified Process

Test Driven Development (TDD)

Feature Driven Development (FDD)

Dynamic System Development Methodology (DSDM)

Extrem Programming (XP)
4 temel değere ve 12 farklı pratiğe sahiptir.

Bu değerler:

- İletişim -Basitlik -Geri Bildirim -Cesaret

İletişim**:** Hem takım içi hem de mmüşteri ile ilişki çok önemlidir. XP iletişimin yüz yüze olmasını ister. Bir yerde bilgi alınması gerekiyorsa en kısa sürede bilgi alınarak hız kesilmemesiniister.

Basitlik: Sağlaması en zor olgu basitiliktir. O anki gereksinimleri karşılayan en basit çözüm kullanılmalıdır.

Geri Bildirim: Geri bildirim sayesinde ortaya çıkabilecek hatalar ortadan kaldırılır ve maaliyet çok küçültülmüş olur. XP'de müşteride takımın bir üyesidir. 2-4 haftalık sürümlerle sistem durumu belirlenir. Müşteriler de belli aralar ile bir araya gelinip gelinen ilerleme gösterilir ve geri bildirimalınır.

Cesaret: Basitlikten de zor bir basamak... Başarısızlıktaen korkmak yerine üzerine gidilmelidir. Memnun etmeyen ürünleri elimizi korkak alıştırmayıp direk atabilmeliyiz. Başarısızlıktan korkmak değil en kolay telafisini bulmak önemlidir. Korkmak yalnızca bizi yavaşlatır!

XP Pratikleri:

1-)Planlama Oyunu**:**Ufak kağıtlara yazılmış önerilerdir.Planlama işinin nasıl geliştiğini gösterir.Waterfallda planlama işi uzun sürdüğünden çevik süreçlerde süreden kazanç olur.Planlama zamanının azaltılması başlangıç için şarttır.
2-)Ufak Sürümler:2 ya da 6 aylık sürümler çıkar.En çok kullancılacak olan yerler ilk başa konulur.Riski azaltır.Sürümler üst üste eklenerek gider.Gereksinimler yanlış belirlenirse, sürümler çok kısa olur anlamlı birşeyler ortaya konulmazsa zaman kaybı olur.Süre uzun olursa başa dönme zor olur.
3-)Metafor:Basitleştirilmiş sistem mimarisidir.Sistem içerisindeki alt sınıfları gösteren modellemedir.Proje gelişirken ilk akla gelen şeyler basit bir şekilde çizilir.
4-)Basit Tasarım:Xp de projeye başlarken tasarım olmadan kod ilede başlayabiliriz.Ama yinede başlangıç için ufak bir tasarım olabilir.Tasarım kod geliştikçe kötüye gidecektir.
5-)Test Güdümlü Geliştirme(TDD):Gerekinimler waterfall da koda dönüştürülür.Kodu oluşturur.XP de gerekinimler kodu değil testleri güdümler.Önce testi yaz gereksinim neyi istiyorsa sonra kodu yaz.Testler müşteri ile yapılır.Basitleştirir.Tek amaç vardır oda testi geçebilecek kadar kod yazmak.Sistemin güvenliği daima kontrol altındadır.Testler iyi bir dökümantasyon olur., amaç sadece testi geçmek olunca kod kısa olur.Dezavantajları, amaç sadece testi geçmek olunca tasarım bozulabilir.Gerekli test yazılamayabilir.Takımı ikna etmek zordur.
6-)Refactoring(Yeniden Yapılandırma):Devamlı bir iyileştirme ile gitmeyi sağlar.Testler yazılınca tasarım daha karmaşık oluyor.Testler tamam olunca tasarıma bakıp iyileştirme yapılır.Avantajı değişim maliyetini düşürür.Dezavantajı ise tasarım değişirken koda zarar verebilir.Örneğin bir üniversite projesinde tasarım öyle değiştiki fakülte silindi bu duruma testler göçer.
7-)Pair Programming (İkili Programlama):İkili takımlar halinde çalışıldığından hergün takım değişebilir.İki kişi farklı düşünebilir.Maliyet yükselir.Herkesin sistemdeki tüm bileşenlerden haberi olur.
8-)Kollektif Sahiplik:Tüm takım herşeyi bilir.Herkes kendi gereksinimi için başkasının kodu ile uyum olmazsa diğer kişinin kodunu değiştirebilir.Dezavantaj olarak aynı anda 2 sürüm olabilir.Değişim için kodu bozabilir.
9-)Sürekli Bütünleşme:Bir test ileride 3 4 teste bağımlı olur.Bütünleşerek gider.Hergün değişiklikler sisteme eklenir.Uyumluluk sorunları erkenden önlenmiş olur.Dezavantajı, ayrı bir bilgisayarda zaman ayırmak gerekir.
10-)Kodlama Standartı:Kodlama için standart oluşturulmalıdır.Büyük harf küçük harf vb metod isimleri falan

herkes farklı şekilde adlandırabilir onu önlemeliyiz.Bunun için kodda ortak bir anlayış olmalıdır.Tüm ekip standar koda uymalıdır.Ekip tarafından standart kabul edilmelidir.
11-)Haftada 40 Saat:Ortalama günde 8 saatten 5 gün yeterlidir.Dinlenme sayesinde ekipten tam performans alınır.Fazla mesai verimliliği düşürür.
12-)Müşterinin Yanında Olmak:Müşteriden bir kişinin proje bitene kadar ekiple beraber olmasıdır.Bu duruma müşteriyi ikna etmek zordur.Ek maliyet getirecektir.Müşteriden ekibe katılan kişi kullanıcı hikayesini yazar.Takıma erken geribildirim (feedback) yapar.

Scrum:

Rugby de şekildeki gibi tüm oyuncuların birbirine kenetlenip atak yapmasından gelir. Karmaşık yazılım işlerini küçük birimlere bölerek geliştirmeyi öngörür. Bu metodoloji karmaşık ortamlar için çok uygundur. Özellikle gereksinimler tam olarak tanımlanamıyorsa ve kaotik bir durum oluşacaksa en uygun metodoloji SCRUM'dur. Bu metodolojide bir yineleme maximum 30 gün sürer ve günlük 15 dk'lik toplantılar ile iş takibi yapılır.

SCRUM'da temel kavramlar:

1 Roller:

Ürün sahibi: Projenin değerinden sorumludur

Scrum Yöneticisi: Takımı scrum değerlerine göre adapte eder.

Svrum Takımı: Arasında sürekli iletişim halinde olan ve tek bir hedefe ulaşmak için çalışan kişilerden oluşur.

2 Toplantılar:

1) Sprint Planning; Product backlog ile belirtilen gereksinimler, bu toplantı ile geliştirme takımı tarafından küçük görevlere (task) ayrılır. Takımdaki her bir kişi kendi hızına göre bu taskleri kendilerine alır. Bu toplantıya product owner, geliştirme takımı ve scrum master katılır. Sprintler; her sprint sonunda product owner a sunulmak üzere yazılım geliştirmeyi hedefleyecek şekilde belirlenir.1–3 haftalık sprintler oluşturulur.

2) Daily Scrum; Her gün aynı yerde aynı saatte ayak üstü yapılan 15 dakikalık toplantılardır. Üyeler davet edilmeyi beklemezler. Bu toplantı gelecek 24 saati planlamak üzere yapılır. Takımdaki her üye dün ne yaptım,bugün ne yapacağım, işimi engelleyen herhangi bir sorun var mı sorularına cevap verir. Bu sayede herhangi bir sorunu var ise scrum master bu problemi ortadan kaldırır. Takım üyelerinden bu probleme yardımcı olabilecek biri var ise toplantı sonunda iletişime geçebilirler. Daily scrum her ne koşulda olursa olsun yapılır. Takımdaki birinin geç kalması veya gelmemesi toplantıyı etkilemez. Sadece takımdaki büyük çoğunluk yok ise toplantı yapılmaz.

3)Sprint Review; Her sprint sonunda yapılır. Yapılan sprint gözden geçirilir, ortaya çıkan ürün değerlendirilir. Amaç yazılımın ürün sahibinin gereksinimlerine uygun olarak geliştirildiğinden emin olmaktır. Eğer bir hata var ise farkedilir ve düzeltilir.

4)Sprint Retrospective; Sprint boyunca yapılan işlerin kalitesinin, doğruların ve yanlışların değerlendirildiği toplantıdır. Bu toplantı scrum takımının kendini geliştirebilmesi için bir fırsattır. "Neleri daha iyi yapabiliriz?", "Nasıl daha iyi yapabiliriz?" sorularına cevap aranır. Bu aşamadan sonra bir sonraki sprint planning toplantısı gerçekleştirilerek yazdıklarımızın hepsi tekrardan yaşanır.

3 Bileşenler ve Araçlar:

Ürün Gereksinim Dökümanı (Product Backlog):

Proje boyunca yapılması gerekenlerin basit bir listesidir. Statik değildir, sürekli değişir. Değişirken de en çok kullanıcı hikayelerinden ararlanız.

Sprint Dökümanı (Sprint Backlog):

Mevcut sprintteki ürün gereksinimlerinden elde edilen işleri kapsar. Hangi iş ne kadarı yapılacak, hangi gün ne kadar yapıldı gibi bilgileri içerir.

Sprint Kalan Zaman Grafiği (Burndown Chart):

Sprint'in toplam ne kadar süreceği hesaplanır ve her gün takım üyelerinin kaç saatlik iş yaptığını girmesi ile bu yapılan işler toplam saatten düşülür. İyi bir motivasyon aracıdır.

Evet geldik son cümlelere. Her bir model hakkında çok fazla kaynak var. Bu yazım ile genel hatları ile bilgi vermek istedim. Umarım yararlı olmuştur ve yine umarım akıcı ve kolay bir şekilde okumuşsunuzdur. Bu benim yayınladığım ilk yazım ve açıkçası ben yazmaktan çok keyif aldım. Diğer yazılarımda görüşmek üzere herkese iyi günler....

Bilal Arslan

210601078

Medium adresi:

https://medium.com/@arslanbilal2264/yazilim-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-sdlc-ve-modelleri%CC%87-5c0d2f735d38

Github adresi:

https://github.com/arsbilCENG

Yararlandığım kaynaklar:

- https://denizkilinc.com/yazilim-yasam-dongusu-temel-asamalari-software-development-life-cycle-core-processes/
- Yazılım Geliştirme

https://www.smartup.com.tr/what-we-do/#:~:text=Temel%20ihtiya%C3%A7lar%20belirlenir%2C%20proje%20i%C3%A7in,tan%C4%B1mlanmas%C4%B1)%20ve%20proje%20planlamas%C4%B1%20ger%C3%A7ekle%C5%9Ftirilir.Compare Text

- Yazılım Geliştirme - Erkamsoft

https://erkamsoft.com/yazilimgelistirme.php

- web.shgm.gov.tr › documents › sivilhavacilikT.C. ULAŞTIRMA VE ALTYAPI BAKANLIĞI SİVİL HAVACILIK GENEL

https://web.shgm.gov.tr/documents/sivilhavacilik/files/pdf/duyuru/2021/HGD/HGD-GYGR.pdf/

- 24 Saatte Yazılımcı - HARUN AYYILDIZ

https://www.harunayyildiz.com/24-saatte-yazilimci-olmak/#:~:text=Yaz%C4%B1l%C4%B1m%20tasar%C4%B1m%C4%B1nda%20kullan%C4%B1lan%20en%20%C3%B6nemli,oda%C4%9F%C4%B1n%C4%B1z%C4%B1n%20y%C3%B6n%C3%BC%20sadece%20Tasar%C4%B1mda%20olucakt%C4%B1r.

- www.orhanyener.net › 2015/03/05 › sdlc-softwareSDLC (Software Development Lifecycle) – Orhan YENER'in ...

https://www.orhanyener.net/2015/03/05/sdlc-software-development-lifecycle//

- www.testeryou.com › tr › sdlc-dongusunde-testSDLC Döngüsünde Test Kariyerinin Önemi Ve Geleceği - TesterYou

https://www.testeryou.com/tr/sdlc-dongusunde-test-kariyerinin-onemi-ve-gelecegi//

- http://blog.emrahkahraman.com.tr/xp-extreme-programming-pratikleri/
- https://medium.com/@secilcor/scrum-nedİr-6a4326951dd8
- http://blog.emrahkahraman.com.tr/xp-extreme-programming-pratikleri/
- https://medium.com/@tunaytoksoz/yazilim-yaşam-döngüsü-sdlc-ve-modelleri-c3fe40f6e4e8
- https://ybsansiklopedi.com/wp-content/uploads/2015/08/Yazılım-Geliştirme-Modelleri-Yazılım-Yaşam-DöngüsüSDLCYBS.pdf