Yazılım Geliştirme Yaşam Döngüsü(Software Development Lifecycle)

Geliştirilen bir yazılım projesinin planlamasından başlayarak teslimatına kadar geçirmiş olduğu bütün aşamalara ve bu aşamalardan oluşan döngüye, Yazılım Geliştirme Yaşam Döngüsü denir. SDLC, müşteri isteklerini karşılayacak şekilde süre ve maliyet tahminleri dahilinde tamamlanması sağlanan kaliteli yazılım üretmeyi hedefler. Yazılım geliştirme yaşam döngüsü genel olarak 5 aşamadan oluşmaktadır. Bu aşamalar; planlama, çözümleme, tasarım, gerçekleştirme ve bakım olarak sıralanır.

1-Planlama: "Ne istiyoruz?" SDLC'nin bu aşamasında ekip, analiz edilen gereksinimlerin uygulanması için gereken maliyeti ve kaynakları belirler. Ayrıca ilgili riskleri detaylandırır ve bu riskleri azaltmak için alt planlar sunar. Başka bir deyişle, ekip projenin fizibilitesini ve projeyi en düşük riski göz önünde bulundurarak nasıl başarılı bir şekilde uygulayabileceklerini belirlemelidir.

2-Çözümleme: Yazılım işlevleri ile gereksinimlerin ayrıntılı olarak çıkarıldığı aşamadır. Eğer varsa mevcut sistem incelenmesi gerçekleştirilir. Temel amaç, bir yazılım mühendisi gözüyle mevcut yapıdaki işlerin ortaya çıkarılması ve doğru olarak algılanıldığından emin olmaktır. Bu aşamada temel UML diyagramlarının çizilmeye başlanıldığı ilk adımdır. (Use Case, Activity, Class…)

3-Tasarım: Çözümleme aşamasından sonra belirlenen gereksinimlere çözüm olacak yazılım ilk ve temel halinin oluşturulması aşamasıdır. Kodlama olarak herhangi bir durum söz konusu değildir. Bu aşama 2'ye ayrılır. Bunlar:

- Mantıksal Tasarım: Mevcut sistem değil önerilen sistemin yapısı anlatılır. Olası örgütsel değişiklikler önerilir.
- Fiziksel Tasarım: Yazılımı içeren bileşenler ve bunların ayrıntılarını içerir.

4-Gerçekleştirme: Çözümlemesi ve tasarımı bitmiş olan sistemin kodlanması aşamasıdır. Kodlama işlemi planlama veya çözümleme aşamasında belirtilen programlama dili, geliştirme ortamı ve teknolojilerin yardımıyla yapılır. Projede ortaya çıkabilecek kusur ve eksik yanları bu aşamada test edilerek ulaşılmış olunur. Bu aşama kodlama, test etme, kurulum olarak 3'e ayrılır.

5-Bakım: Test etme ve kurulum aşamasından sonra gerçek ortama kurulan yazılımın, hata giderme, güncelleme gibi işlemlerin yapıldığı aşamadır. Bu aşama yazılım tüm yaşamı boyunca sürer. Bu yazılım yaşam döngüsünü daha aktif ve en iyi sonucu vermesi için yazılım geliştirme süreç modelleri kullanılabilir. Bu süreç modelleri ise en temelde 3'e ayrılırlar. Bunlar:

1-Yazılım Geliştirme Süreci Modelleri

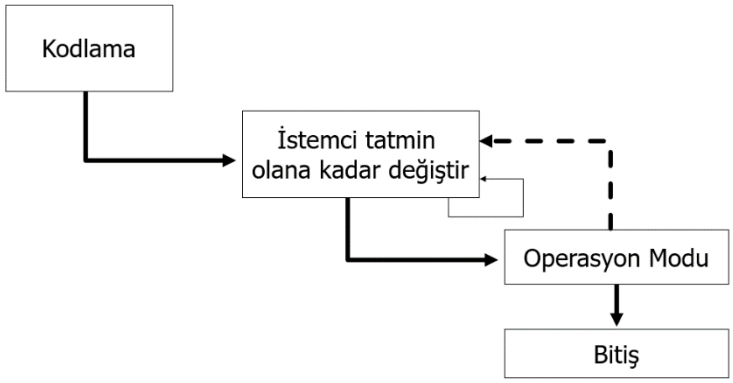
2-Birleşik Süreç

3-Çevik Yazılım Süreç Modelleri

Yazılım Geliştirme Süreci Modelleri

Yazılım geliştirme yaşam döngüsü (SDLC) modelleri, yazılım geliştirmenin karmaşık ve zorlu süreçlerini iyileştirmeyi hedefler. Bir yazılım projesinin kalitesi, zaman çerçeveleri, bütçesi ve müşterilerin beklentilerini karşılama hedefi yazılım geliştirme sürecinde seçilen modele bağlı olarak değişir.
Günümüzde kullanılan yaklaşık 50'den fazla SDLC modeli vardır. Bu modellerin her birinin belirli bir yazılım projesi veya ekipler için avantajları ve dezavantajlarını vardır.

1-Kodla ve Düzelt (Code and Fix) Model



Bu model genellikle resmi olmayan bir ürün fikriyle başlar ve program ürün "hazır" olana kadar ya da gerekli zaman bitene kadar kodlama yapılarak devam eder.

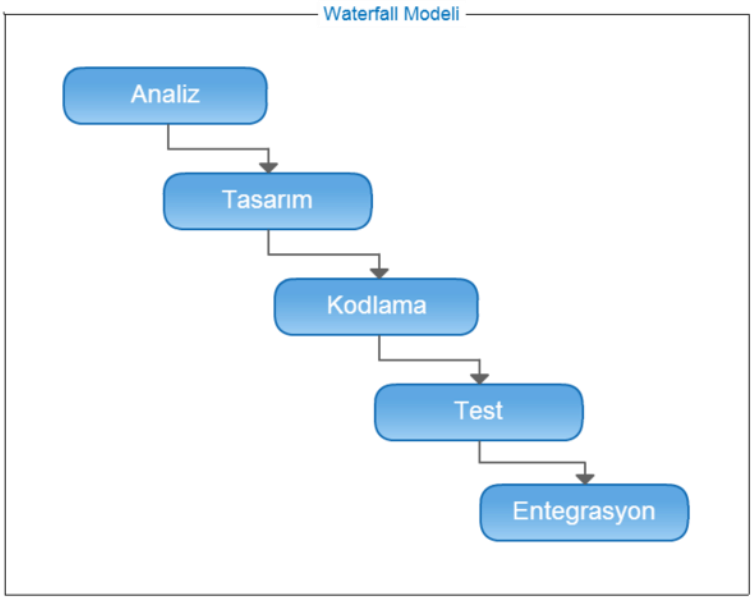
Avantajları

- Herhangi bir planlamaya ihtiyaç duyulmaz.
- Çok küçük projelerde ya da kısa ömürlü prototiplerde uygulanabilir.
- Program aşamaları çabuk geçilir.
- Uzman görüşüne ihtiyaç düşüktür herkes bu modeli kullanabilir.

Dezavantajları

- Kontrollü değildir.
- Kaynak planlaması yoktur.
- Bitiş süresi belli değildir.
- Hataların bulunması ve doğrulaması zordur.
- Kodları düzeltmek maliyetli olabilir.
- Kodlar kullanıcının ihtiyacını karşılamayabilir.
- Kodlar sonradan değiştirmek için planlanmadığından esnek değildir, değiştirilmesi zordur.

2-Çağlayan Modeli



Şelale yönteminde yazılım geliştirme süreci analiz, tasarım, kodlama, test, sürüm ve bakım gibi safhalardan oluşur. Geleneksel yazılım metotlarında bu safhalar şelale modelinde olduğu gibi lineer olarak işler. Her safha, başlangıç noktasında bir önceki safhanın ürettiklerini bulur. Kendi bünyesindeki değişikler doğrultusunda teslim aldıklarını bir sonraki safhanın kullanabileceği şekilde değiştirir.

Avantajları

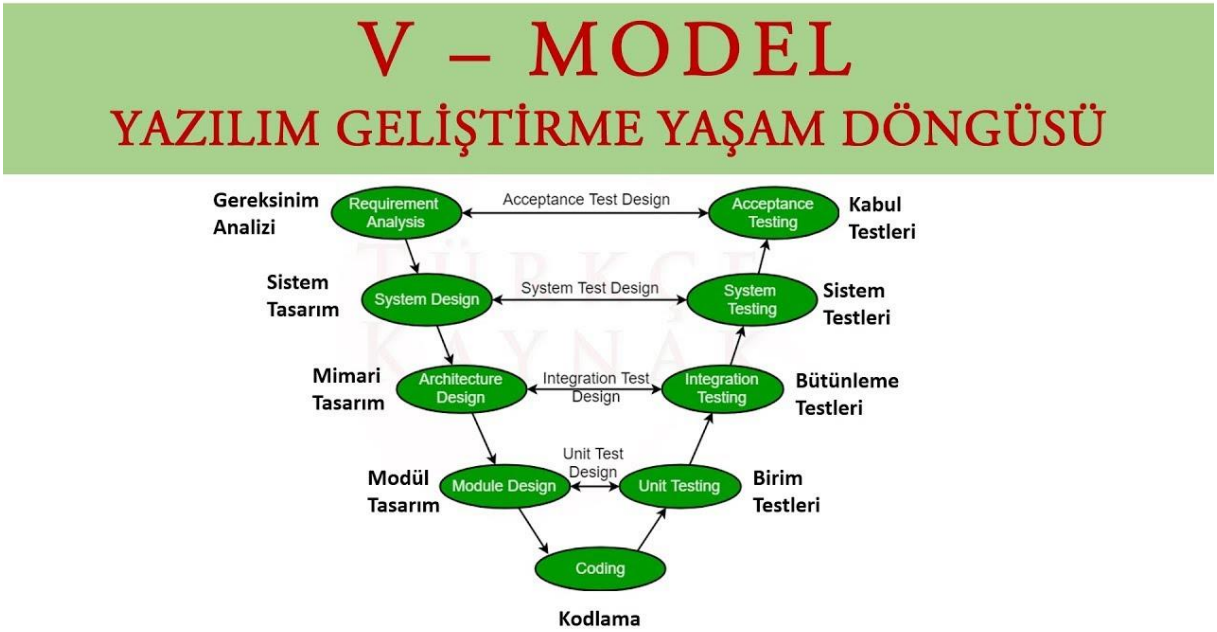
- Kullanımı ve anlaması basittir.
- Yönetimi kolaydır.
- Projenin safhaları ayrı olduğundan iş bölümü ve iş planı projenin en başında net bir şekilde bellidir. Bu durum projenin yönetimini de oldukça kolay hale getirir.
- Şelale Modeli çok küçük ve gereksinimleri çok iyi anlaşılmış projelerde iyi çalışır.

Dezavantajları

- Karmaşık ve nesne yönelimli projeler için uygun değildir.
- Devam eden ve uzun projeler için zayıftır.
- Projede oluşabilecek her türlü değişime elverişsiz, katı bir modeldir. Yapılan her değişiklik maliyeti büyük oranda arttırır.
- Müşteri memnuniyetini sağlamak çok zordur çünkü gelişim ve değişime açık bir model değildir.

V Modeli

V-model(yazılım geliştirme) şelale(waterfall) modelinin gelişmiş hali olarak düşünülebilecek bir yazılım geliştirme süreci sunar. Doğrusal bir yönde ilerlemek yerine, süreç adımları kodlama evresinden sonra yukarıya doğru eğim alır ve tipik V şeklini oluşturur. V-Model geliştirme yaşam çevriminin her bir evresi arasındaki ilişkileri gösterir. Yatay ve dikey açılar zaman veya projenin tamamlanabilirliğini ve soyut seviyeyi gösterir.



Avantajları

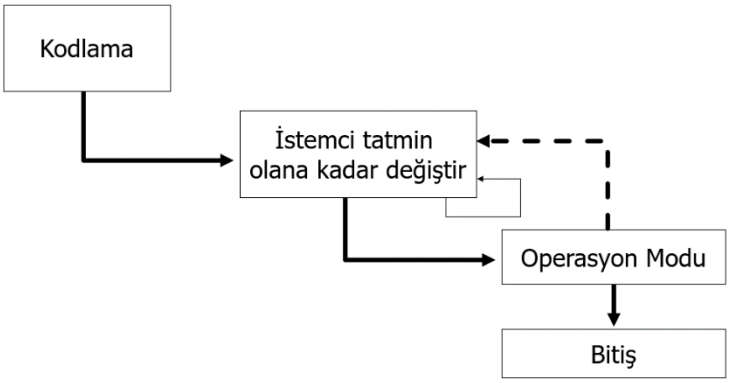
- Basit ve kullanımı kolaydır.
- Planlama ve test tasarımı gibi test faaliyetleri kodlamadan önce gerçekleştirildiği için proje içerisinde çok zaman kazandırır. Bu nedenle şelale modeline göre daha yüksek başarı şansı vardır.
- Hataların bulunması erken aşamada bulunur.
- Hataların bir sonraki aşamaya geçmesi önlenir.

Dezavantajları

- Aynı zamanda gerçekleştirilebilecek olaylara kolay imkan tanımaz.
- Fazlar arasında tekrarlamaları kullanmaz.
- Risk çözümleme ile ilgili aktiviteleri içermez.
- Yazılım da diğer sistemler gibi zamanla evrimleşir.
- Geliştirme devam ettikçe iş ve ürün gereksinimleri de değişkenlik gösterebilir.
- Son ürüne ulaşma düz bir çizgi ile ifade edilemez.

Evrimsel Geliştirme (Evolutionary Development)

İlk tam ölçekli modeldir. Coğrafik olarak geniş alana yayılmış, çok birimli organizasyonlar için önerilmektedir (banka uygulamaları). Diğer modeller ile kıyaslandığında ilerleyişi daha yavaştır. Modelin tamamının başarımı geçirdiği ilk evrimin başarısına bağlıdır. Modelin başarısı ilk evrimin başarısına bağımlıdır.



İki çeşit evrimsel geliştirme vardır:

- Keşifçi geliştirme (exploratorydevelopment)
    - Hedef: Müşterinin gereksinimlerini incelemek için müşteri ile çalışıp son sistemi teslim etmek
    - İyi anlaşılan gereksinimlerle başlanmalıdır
- Atılacak prototipleme(throw-awayprototyping)
    - Hedef: Sistem gereksinimlerini anlamak
    - Tam anlaşılmamış gereksinimlerle başlar

Avantajları

- Kullanıcıların kendi gereksinimlerini daha iyi anlamalarını sağlar.
- Sürekli değerlendirme erken aşamalardaki geliştirme risklerini azaltır.
- Hatalar azalır.

Dezavantajları

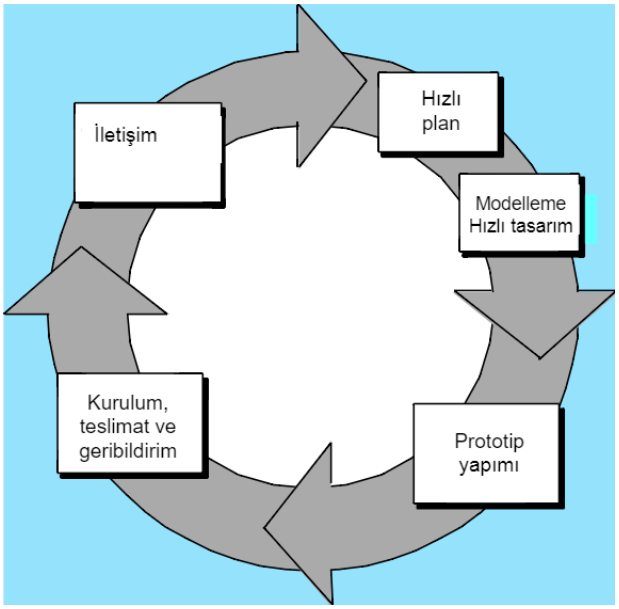
- Sürecin görünürlüğü azdır. (düzenli teslim edilebilir ürün yoktur)
- Sistemler sıklıkla iyi yapılandırılmaz. (sürekli değişiklik yazılımın yapısına zarar verir)
- Bakımı zordur.
- Yazılım gereksinimini yenilemek gerekebilir.

Prototipleme (Prototyping)

Gereksinimler hızlıca toplanarak işe başlanılır. Geliştiriciler ve kullanıcılar aynı masa etrafında buluşarak yazılımdan elde edilecek bütün çıktılara, bu çıktılar için gerekli girdilerin nasıl sağlanacağına, nasıl korunacağına, hangi işlemlere uğrayacağına karar verirler. Daha sonra hızlıca yapılan bir tasarım ile yazılımın kullanıcıya yansıyacak yönünü aktaran bir ilk örnek üretilir. Prototip kullanıcının kullanımına ve değerlendirilmesine sunulur.

Bu değerlendirmelere bakılarak ilk örnek üzerinde gerekli değişiklikler yapılır. Prototipin yeni hali kullanıcı tarafından yeniden değerlendirilir. Böylece kullanıcının istediği yazılıma iyice yaklaşılmış bir ilk örnek üzerinde yazılımın neler yapacağı konusunda kullanıcı ile anlaşmaya varılır. Doğrusal modelin döngüsel versiyonudur.

Bu modelde, gereksinim analizi ve prototipleme için tasarım yapıldıktan sonra, geliştirme süreci başlatılır. Prototipleme yaratıldıktan sonra, müşteriye değerlendirme için verilir. Müşteri paketi test eder ve düşüncelerini, ürünü müşterinin tam beklentilerine göre düzenleyen geliştiriciye iletir. Sınırlı sayıdaki yinelemelerden sonra, son yazılım paketi müşteriye verilir. Bu metodolojide, yazılım müşteri ve geliştirici arasında periyodik bilgi gidip gelmeleri sonucunda gelişir.



Avantajları

- Kullanıcı sistem gereksinimlerini görebilir.
- Karmaşa ve yanlış anlaşılmaları engeller.
- Yeni ve beklenmeyen gereksinimler netleştirilebilir.
- Risk kontrolü sağlanır.

Dezavantajları

- Belgelendirmesi olmayan hızlı ve kirli (quick and dirty) prototipler
- Prototip hedefleri net değilse kod hackleme ya da jenga başlar.

- Düzeltme aşaması atlanırsa, düşük performansa yol açar.
- Müşteri prototipten de son ürün gibi görünüm ve etki bekler.

Spiral Model

Sarmal modeli aynı safhalara geri dönülmesinin bir zorunluluk olduğunu vurgular, Sarmal modeli şelale modelinde yok sayılan riskleri göz önünde bulundurur, Proje çevrimlere ayrılır ve her bir çevrimin riskleri ayrı ayrı ele alınır. Çağdaş modellere son derece yakındır.

- Planlama

Üretilecek ara ürün için planlama, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirme

- Risk Analizi

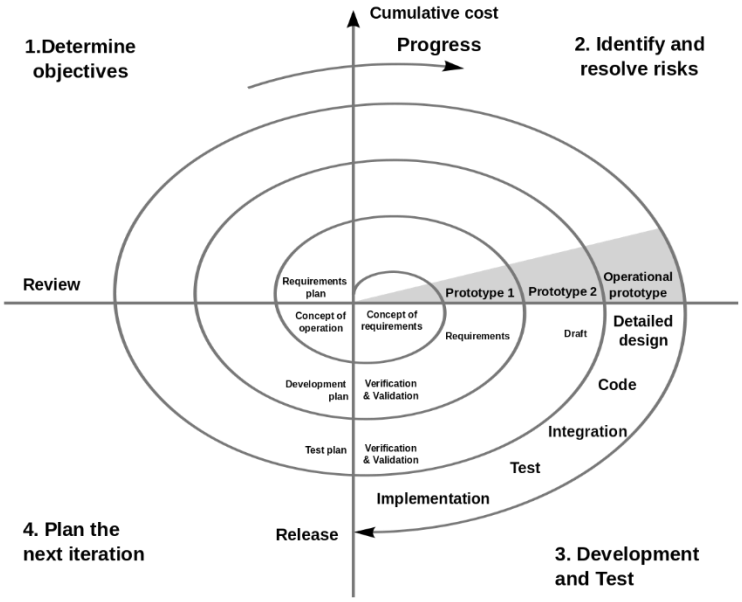
Risk seçeneklerinin araştırılması ve risklerin belirlenmesidir.

- Üretim

Ara ürünün üretilmesidir.

- Kullanıcı Değerlendirmesi

Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmelerdir.



Risk Analizi Olgusu ön plana çıkmıştır. Her döngü bir fazı ifade eder.

Avantajları

- Kullanıcılar sistemi erken görebilirler.
- Geliştirmeyi küçük parçalara böler. En riskli kısımlar önce gerçekleştirilir.
- Pek çok yazılım modelini içinde bulundurur.
- Riske duyarlı yaklaşımı potansiyel zorlukları engeller.
- Seçeneklere erken dikkate odaklanır.
- Hataları erken gidermeye odaklanır.
- Yazılım-donanım sistemi geliştirme için bir çerçeve sağlar.

Dezavantajları

- Küçük ve düşük riskli projeler için pahalı bir yöntemdir.
- Komplekstir.
- Spiral sonsuza gidebilir.
- Ara adımların fazlalılığı nedeniyle çok fazla dokümantasyon gerektirir.
- Büyük ölçekte projelerdir.
- Kontrat tabanlı yazılıma uymaz.
- Yazılımın içten geliştirileceğini varsayar.
- Kontrat tabanlı yazılımlar adım adım anlaşma esnekliğini sağlamaz.
- Öznel risk değerlendirme deneyimine dayanır.
- Yüksek riskli öğelere yoğunlaşmak, yüksek riskli öğelerin doğru belirlenmesini gerektirir.

Formal Sistem Geliştirme (Formal System Development)

Formal Sistem Geliştirme Modeli yazılım tasarım ve gerçekleştirmesiyle ilgili matematiksel bir tekniktir. Bu modelin temelinde karmaşık sistemleri geliştirme ve program geliştirmeye destek yatar. Formal Sistem Geliştirme Metodu kullanıcı sistemi kullanmaya başladığında karşısına çıkan belirtim hatalarını minimize eder.

Formal belirtim, tasarım ve geçerleme kullanarak yazılımda doğruluğun geliştirilmesini vurgular. Yazılım artımlarla geliştirilir. Sürekli tümleştirme vardır ve fonksiyonellik tümleştirilen yazılım artımları ile artar. Felsefesi pahalı hata ayıklama işlemini engellemek için kodu ilk yazarken doğru yazmak ve test aşamasından doğruluğunu sağlamaktır.

Avantajları

- Yazılımdaki belirsizlikleri, eksiklikleri ve uyumsuzlukları saptar.
- Hatasız yazılım geliştirme imkanları sunar.
- Her iterasyondan sonra aşamalı olarak artan efektif çözümler sunar.
- Karmaşık değildir.

Dezavantajları

- Çok zaman alan ve pahalı bir yöntemdir.
- Kullanımı esnasında teknik olmayan personelle iletişim mekanizması zor işler.
- Sadece birkaç geliştirici bu modelin uygulamasıyla ilgili temel bilgilere sahip olması için yaygın eğitim gerektirir.

Artımlı Geliştirme (Incremental Development)

Eğer bir müşterinin ürünlerinde değişikliğe ihtiyaçları varsa, artımlı model ihtiyaç olan bu değişikliğe ayak uydurur. Artırımsal model bir takvime bağlı olarak yazılımı kesim kesim geliştirip teslim etmeye dayanır. Her bir yeni kesim öncekinin üstüne bazı ek işlevlerin eklenmesini öngörür. Artırımsal model yazılım geliştirmenin kısıtlı sayıda çalışanla işin yapılmasını sağlama gibi bir üstünlüğü vardır. Ayrıca çalışanlar da her artırım geçildiğinde uygulama alanına ilişkin daha çok deneyim kazanmış olurlar. Bu modelde bir taraftan üretim bir taraftan da kullanım yapılır. Önceki modellerde ürünlerdeki değişiklikler göz önünde bulundurulmaz. Bu model doğal olarak yinelemelidir. Yeniden kullanılabilir bir ürün, fonksiyonellik sağlamış bir şekilde tüm döngülerin sonunda ortaya çıkar.

Avantajları

- Sistem için gerekli olan gereksinimler müşterilerle belirlenir.
- Gereksinimlerin önemine göre teslim edilecek artımlar belirlenir
- Öncelikle en önemli gereksinimleri karşılayan çekirdek bir sitem geliştirilir.
- Erken artımlar prototip gibi davranarak, gereksinimlerin daha iyi anlaşılmasını sağlar
- Tüm projenin başarısız olma riskini azaltır
- En önemli sistem özellikleri daha fazla sınanma (test edilme) imkanı bulmuş olur.
- Divide and Conquer (Böl ve Yönet) yaklaşımıdır.

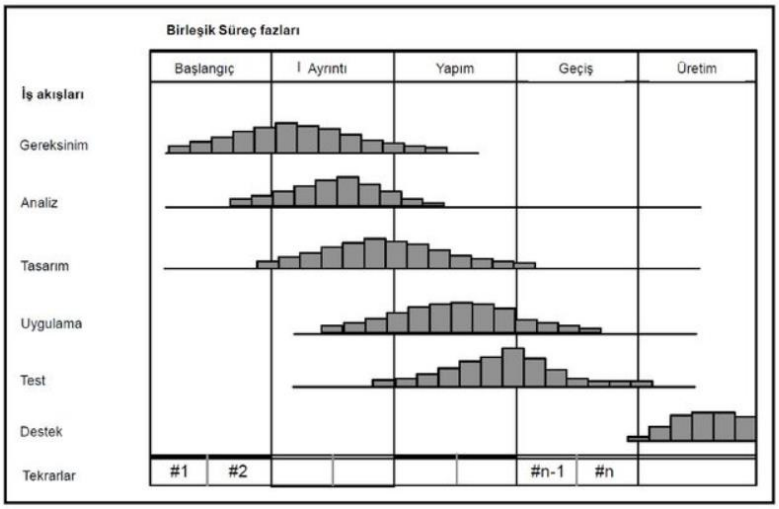
Dezavantajları

- Artımları tanımlamak için tüm sistemin tanımlanmasına ihtiyaç vardır.
- Gereksinimleri doğru boyuttaki artımlara atamak bazen zor olabilir.
- Artımların kendi içlerinde tekrarlamalara izin vermez.

Birleşik Süreç (Unified Process)

Nesne tabanlı (OOP) yazılım geliştirmek için var olan yöntemlerin kullanılmasıyla edinilen deneyimlerle bu süreçlerin en iyi özellikleri bir araya getirilerek bütünleştirilmiş bir yazılım geliştirme sürecidir.

Birleşik Süreç Fazları



Yinelemeli (Iterative): Problemdeki istekler bir bütün olarak görülmez. Problemdeki hedeflerin bir kısmı ele alınır ve sınanmış bir ürün ortaya konulur. Her ürün oluşumunun sonunda bir sonraki yinelemeye (iterasyon) geçilir ve yeni istekler ele alınarak geliştirilir. Her iterasyon sonunda hedeflenen ürüne yaklaşılır. Her bir iterasyona ayrı bir proje gibi bakılır ve tüm aşamalardan geçirilerek sınanmış ve çalışır bir ürün elde edilir.

Artırmalı ve Evrimsel (Incremental and Evolutionary): Her yinelemenin ardından farklı ve yeni istekler ele alındığı için elde edilen ürünlerin özellikleri artarak gelişir ve istenilen ürüne daha fazla yaklaşılmış olunur.

Risk Güdümlü (Risk-Driven): İlk iterasyonda en riskli kısımlar ele alınmalıdır. Böylece proje daha ilerlemeden oluşabilecek büyük problemler görülür ve gerekli önlemler alınabilir. Örneğin bütçe, personel miktarı veya zaman planı gibi değişkenler ayarlanabilir.

Avantajlarına gelecek olursak, proje adım adım geliştiği için değişen isteklere uygundur. Riskler erkenden belirlenip giderilebilir. Her iterasyonda deneyim kazanılır. Sürekli ürün elde edildiği için takımdaki moral ve şevk artar. Tüm bunların yanında karmaşık bir sitemdir. Dokümantasyonu çok iyi yapılmalıdır. Ve dokümantasyon yükü ağır olduğu için maliyet fazla olabilir.

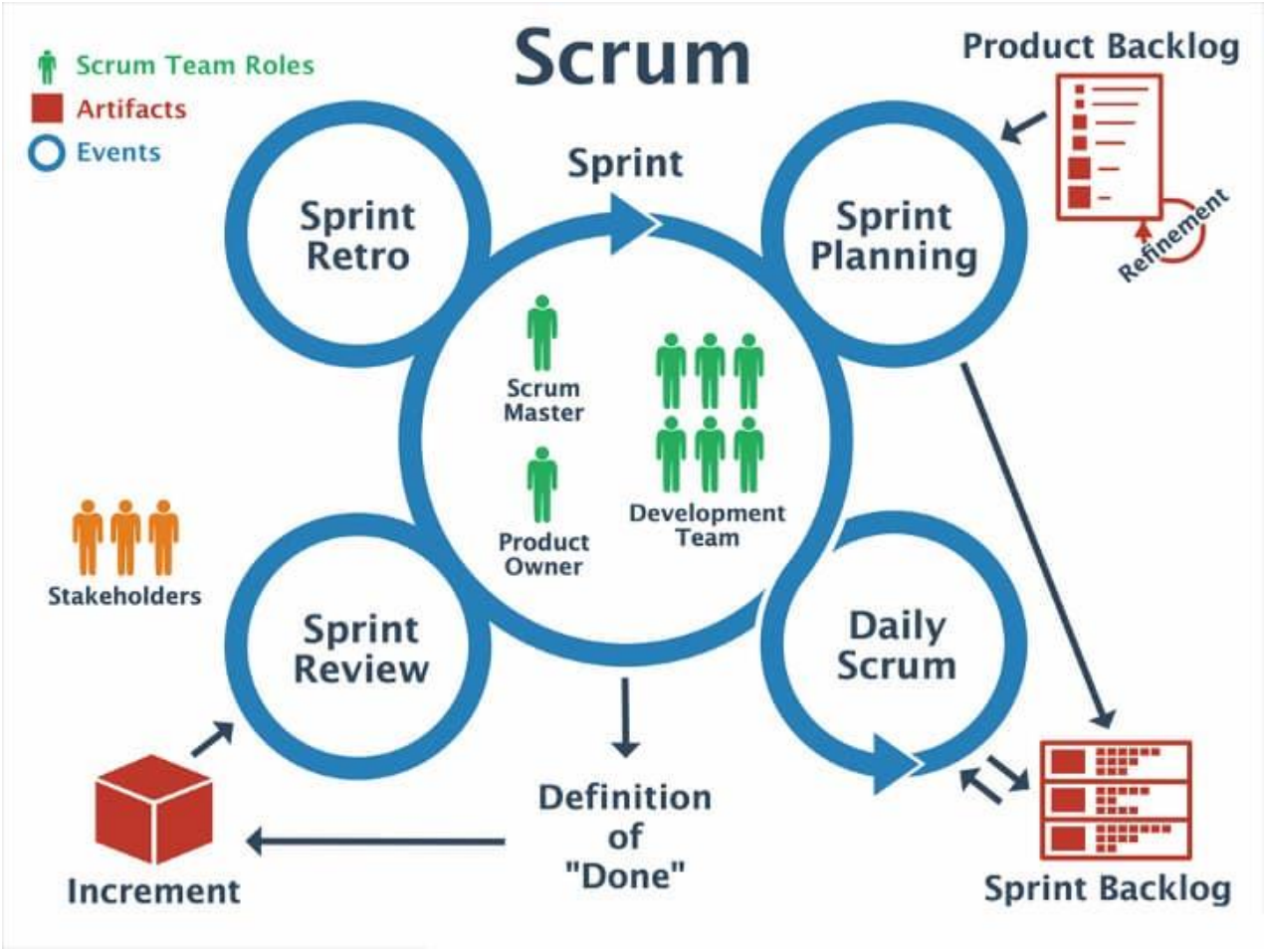
Çevik Yazılım Geliştirme Süreci (Agile Programming)

Yenilemeli (iterative) geliştirmeyi temel alan bazı yazılım geliştirme metodolojilerine dayanarak harmanlanmış, yenilikçi bir yazılım geliştirme süreçleri topluluğudur. “Heavyweight” olarak da bilinen eski, yavaş ve bürokratik sistemlere tepki olarak geliştirilmiştir. Agile metotları genellikle takım çalışmalarının, sık denetimlerin, düzenli ürün sunmanın ve müşteri ihtiyaçlarıyla firmanın yeteneklerinin uyumlu bir şekilde bir araya gelerek hızlı ve kaliteli bir biçimde ortaya ürün koyan süreçlerdir. “İlerlemenin en iyi göstergesi, çalışan yazılımdır” prensibiyle devamlı surette müşteriye bir ürün sunulur. Takımların kendi kendilerine organize olmaları önemlidir. Bunun için düzenli aralıklarla, takımlar kendilerini değerlendirerek nasıl daha etkili olabileceklerine bakar, çalışmalarını buna göre düzenler ve devam ederler.

Avantajlarına gelecek olursak, sık çıktı üretip beğeniye sunularak müşteri ihtiyaçlarına göre revize edilir. Kısa sürede müşteri memnuniyeti sağlanır. Doğal insan eğilimine yatkın olduğundan fazla eğitim gerektirmez, hızlı bir şekilde adapte olunabilir. Değişime açıklık ve esneklik seviyesi yüksektir. Projede planlama ve yürütme bir arada olduğundan plan aşamasında ayrıntıya kaçmak yerine iterasyonun planı yapmak yeterlidir.
Her süreçte olduğu gibi çevik yazılımın da dezavantajları vardır. Kurumsal bir yapıda uygulama zordur. Daha çok bağımsız şirketler tercih ederler. Sürekli değişen ihtiyaçlardan dolayı aşırı çalışma gereklidir. Ürünün başarısı ile proje başarısı ortaktır ve bu yüzden kariyer riski doğurur. Ve takımlar üzerinde hedefe ulaşmak için baskı oluşur.
Uç Değer Programlama (Extreme Programing — XP), Scrum, Çevik Tümleşik Süreç(Agile Unified Process -AUP) ve Özellik Güdümlü Geliştirme (Feature-Driven Development — FDD) günümüzde yaygın olarak kullanılan Çevik Yazılım Metotlarından bazılarıdır.

Scrum Nedir?
Scrum, başarılı çevik proje yönetim tekniklerinden biridir. Karmaşık yazılım süreçleri için kullanılır. Bütünü parçalara bölerek, projenin sağlam adımlarla ve değişimlere ayak uydurarak ilerlemesini sağlar. İşler parçalara bölündüğü için zamandan tasarruf edilir. Bunun sebebi, Scrum'da iletişim ve takım çalışmasının yüksek olmasıdır. Sürekli iletişim halinde olunduğu için hata olasılığı düşer ve zamandan kazanılır.
Scrum 3 temel ilkeye dayanır;

1. **Şeffaflık:** Projenin gidişatı, çözülmeye çalışılan sorunlar ve gelişmeler herkese açık olmalıdır.
2. **Denetleme:** Sık kontrollerle proje süreci kontrol edilerek, beklenmedik bir hatayla karşılaşma olasılığı en aza indirilir.
3. **Uyarlama:** Proje sürecinde çözülemeyen sorunlar veya herhangi bir yenilik olduğu zaman süreç değiştirilebilir.
   Scrum'ın işleyişi aşağıdaki gibidir;



Scrum'da Yer Alan Roller

Product Owner: Müşterinin isteklerini aktaran kişidir. Bazı durumlarda bu kişi müşterinin kendisi olabilir. Bu istekleri yazılım ekibine iletir.
Scrum Master: Scrum Master, takımın Scrum'ı anlamasına ve Scrum prensiplerini yerine getirmesine rehberlik eden ve takım için iletişimin sağlanmasında rol oynayan kişidir.
Development Team: Müşteriden gelen isteklerin oluşturulması ve geliştirilmesinden sorumlu ekiptir.

Scrum Terimleri

Planlama Toplantısı (Sprint Planning): Product Owner, Sprint'ten önce bir hedef belirler. Development Team ile birlikte çalışarak hangi işlerin gelecek Sprint'e alınacağına karar verirler. Eğer iş yükü geliştirme ekibine fazla gelirse, iş birden fazla Sprint'e bölünür.

Günlük Sprint Toplantıları (Daily Scrum Meeting): Scrum takımı günlük olarak en fazla 15 dakika süren toplantılar gerçekleştirir. Bu toplantılarda, yapılan ve yapılacak olan işler ve eğer bir problem yaşandıysa bunun hakkında konuşulur. Böylece sık iletişim kurularak sorunlar hemen giderilir.

Demo Toplantısı (Sprint Review): Geliştirme ekibi, Scrum Master, Product Owner ve müşterinin katıldığı bu toplantı, her Sprint sonrasında gerçekleştirilir. Geliştirme ekibi, Sprint sürecinde hangi işleri yaptığını ve ortaya çıkan sonucu sunar. İşler sunulduktan sonra müşteri yorumlarını aktarır ve düzenlenmesi gereken bir yer varsa bildirir.

Sprint Retrospective: Scrum takımının kendini değerlendirdiği, Sprint sonunda yapılan toplantıdır. Takımlardaki gelişimi sürdürmek adına önemli toplantılardır. Doğru ve yanlış yapılan işler değerlendirilir ve "Ne, nasıl daha iyi yapılabilir?" soruları üzerine düşünülür.

Scrum günümüzde neden popüler?

Scrum, günümüzde popüler bir proje yönetim yaklaşımı haline gelmiştir çünkü esnek, hızlı ve müşteri odaklı bir yaklaşım sunar. Bu yaklaşım, birçok şirketin, özellikle yazılım geliştirme ve teknoloji odaklı şirketlerin hızlı bir şekilde değişen iş ihtiyaçlarına uyum sağlamasına yardımcı olur.

Scrum, çevik bir proje yönetimi yaklaşımıdır ve projenin her aşamasında müşteriye yönelik geri bildirimler toplamak ve projeyi sürekli olarak geliştirmek için düzenli olarak yapılan incelemelerle desteklenir. Bu, proje ekibinin, müşteri ihtiyaçlarını daha iyi anlaması ve bunlara uygun çözümler sunması için daha fazla fırsat sunar.

Scrum ayrıca, proje ekiplerinin daha özerk olmasına ve karar alma sürecinde daha fazla dahil olmasına olanak tanır. Bu, ekiplerin daha yaratıcı ve yenilikçi çözümler bulmasına yardımcı olur ve sonuç olarak, proje başarısını artırır.

Sonuç olarak, Scrum, birçok şirket için etkili bir proje yönetimi yaklaşımıdır çünkü müşteri ihtiyaçlarına odaklanır, hızlı bir şekilde yanıt verir ve proje ekiplerine daha fazla özerklik ve karar alma gücü verir. Bu nedenle, birçok şirket Scrum'u tercih ederek, daha verimli ve başarılı projeler yapmayı hedeflemektedir.

Duygu Aslan

220601018

Referanslar:

- https://medium.com/@denizkilinc/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-temel-a%C5%9Famalar%C4%B1-software-development-life-cycle-core-processes-197a4b503696
- https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/
- https://talentgrid.io/tr/yazilim-gelistirme-modelleri/
- https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/
- https://hayririzacimen.medium.com/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-s%C3%BCre%C3%A7-modelleri-70fdfb2f8f77#:~:text=Birle%C5%9Fik%20S%C3%BCre%C3%A7%20(Unified%20Process),b%C3%BCt%C3%BCnle%C5%9Ftirilmi%C5%9F%20bir%20yaz%C4%B1l%C4%B1m%20geli%C5%9Ftirme%20s%C3%BCrecidir.

- https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/
- https://caglartelef.com/yazilim-yasam-dongusu/
- https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm#:~:text=The%20Waterfall%20model%20is%20the,the%20previous%20phase%20is%20complete

Linkedin: https://www.linkedin.com/in/duygu-aslan-862910246/

GitHub: https://github.com/duyguaslann

Medium: https://medium.com/@duygucanaslan