**YAZILIMIN YAŞAM DÖNGÜSÜ**

**Beyza Nur KARAKOÇ**

**210601006**

Bakırçay Üniversitesi Mühendislik Ve Mimarlık Fakültesi /Bilgisayar Mühendisliği 1. Sınıf Öğrencisi

**ÖZET:**

Yazılım Geliştirme Yaşam Döngüsü , yazılımları tasarlamak, geliştirmek ve test etmek amaçlı kullanılan bir süreç olarak ifade edebiliriz. Yazılımın nasıl geliştirileceği, sürdürüleceği ve daha iyi hale nasıl getirileceğinin açıklayan bir plandan oluşmaktadır. Buradan da yazılımın aslında bir ürün olduğu ve o ürününde bir yaşam süreci olduğunu gözlemlemiş oluyoruz. 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. Bunlar; planlama, analiz, tasarım, test ve bakım alanlarından oluşmaktadır.

Planlama: Yaşam döngüsünün ilk aşamasıdır. Burada “Ne istiyoruz” sorusuna yanıt arıyoruz. Başka bir ifade ile ekip bu aşamada projenin fizibilitesini ve projeyi nasıl başarılı bir şekilde faaliyete geçirebileceğini belirlediği yerdir.

Analiz: Projenin ne kadar sürede tamamlanacağı ve ne gibi risk durumları olduğunun belirlendiği alandır. Bu aşama temel UML diyagramlarının başlandığı ilk alandır. Elde edilen bilgilerin belirli bir format halinde doküman haline getirilmesidir.

Tasarım: Analiz kısmının tamamlanması sonucunda tasarım aşamasına geçilir. Burada “İstediğimizi nasıl elde edeceğiz” sorusuna cevap aranır. Proje sürecinin nasıl devam edeceği konusunda bir tasarlama işlemi gerçekleşir. Tasarımda en önemli tekniklerden bir tanesi de Soyutlama(Abstraction) tekniğidir. Bu teknik ile problemi daha basit hale getirerek problemdeki önemli kısımlara dikkat edilmesine olanak tanır.

Test: Bu kısım projenin kodlandığı ve test edildiği alandır. Bu aşamada “İstediğimizi aldık mı?” sorusuna cevap arıyoruz. Projede ortaya çıkabilecek kusur ve eksik yanları bu aşamada test edilerek ulaşılmış olunuyor.

Bakım: Yazılım ürünü tüm aşamaları tamamlayıp sahaya çıkartılıp teslim edildikten sonra bu aşama başlar. Bu aşama ürün ile ilgili alınan geri bildirimler sonucunda üründe yapılan yeni güncellemeler ve bakım durumlarının yapıldığı alandır.

Yazılım Geliştirme Yaşam Döngüsü Modelleri (Software Process Models)

Yazılım geliştirme yaşam döngüsünde, süreçlerin geliştirme aşamasında, sürecin düzeni ve nasıl uygulanacağını ifade eden modellerdir. Karmaşıklık düzenini azaltır ve böylece oluşacak krizleri önler. Ürünlerin belirli bir kalitede olmasına olanak tanır.

Şelale Modeli (Waterfall Model)

V Modeli (V-Shaped Model)

Evrimsel Geliştirme (Evolutionary Development)

Prototipleme (Prototyping)

Kodla ve Düzelt (Code and Fix)

Artımlı Geliştirme (Incremental Development)

Spiral Model

Big Bang Model

RAD Model (Hızlı Uygulama Geliştirme)

Agile Model (Çevik Model)

**Yazılım Yaşam Döngüsü Nedir?**

Yazılımda aslında bir üründür ve ürünlerin bir yaşam süreci vardır. Yazılımın yaşam döngüsü tek yönlü ve doğrusal değildir. Geliştirdiğiniz bir yazılım projesinin planlanmasından başlayarak teslimatına kadar geçirmiş olduğu bütün aşamalara yazılım geliştirme yaşam döngüsü denir. Yazılımın yaşam döngüsü aynı zamanda il geliştirme aşamalarından başlayarak; mevcut sürümün hatalarının giderilmesi, iyileştirme çabasında yeni sürümler yayınlayarak yazılımın hatasız daha kullanılabilir sürüme ulaşmasını hedef bilir.

**Süreç:**

**GA(General Availability):**

Genel Kullanılabilirlik (GA), bir ürünün genel halka sunulmasıdır. Bir ürün GA'ya ulaştığında, öncelikle test ve kullanıcı geri bildirimi amacıyla kullanılan sınırlı bir sürüm veya beta sürümünün aksine, şirketin genel satış kanalı aracılığıyla kullanılabilir hale gelir.

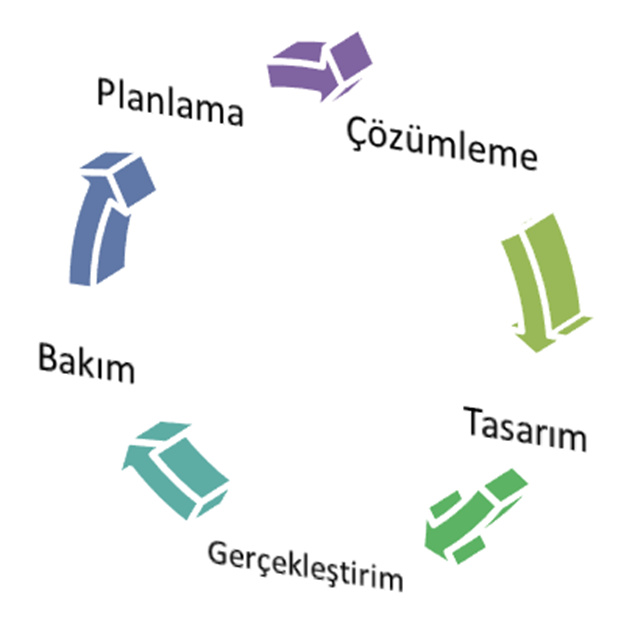
**RTM(Release To Manufacture):**

Üretime bırakmanın kısaltması olan RTM, ilk olarak donanım üreticilerine paketleme için piyasaya sürülen yazılım sürümünü tanımlamak için kullanılan bir terimdir. RTM süreci, geliştiricilerin yazılımın üretici iyileştirmesi yoluyla genel sürümden önce herhangi bir hatayı çözmesine yardımcı olması açısından faydalıdır. Örneğin, Microsoft bir işletim sistemi çıkardığında, diski RTM'ye çevirir ve yazılımın gelecekteki tüm kopyaları o diskten yapılır. Apple bu kopyaya GM olarak atıfta bulunur.

**RTV(Release To The Web):**

Dağıtımı internet üzerinden yapılan ürünler için geçerli olan bir terimdir.

**Yazılım Yaşam Döngüsü Aşamaları:**

Basitçe bir proje geliştirilirken projenin **planlama**, **analiz**, **tasarım**, **üretim** ve **test**aşamaları yer almaktadır ve almalıdır. Bu aşamalar bir kere gerçekleştirildikten sonra proje tamamlanmayabilir. Bu aşamaların bir döngü halinde düşünülmesi gerekmektedir. Proje tamamlandıktan sonra gelecek istekler, hata düzeltmeleri, projeye eklenecek yeni modüller için bu süreç devam etmektedir. Bir yazılım projesinde tüm aşamalar tamamlandıktan sonra hiçbir hata ve yeni istek olmadığını varsayalım. Bu kısımda devreye bakım aşaması girmektedir. Geliştirilen bütün yazılım projelerinin ileride doğabilecek hataları ve yeni istek ve talepleri için bakım süreci de yazılımın yaşam döngüsüne eklenmesi gerekmektedir. Temel yazılım geliştirme aşamaları aşağıdaki gibidir:

**Planlama:**

Örneğin: Bir yazılım üretilecek; alt yapılar, istekler oluşturulacak ve bu yazılımı tasarlayacak bir ekip olacak. Yazılımın içinde hatalar, eksiklikler, uyuşmazlıklar olacak, işte bu döngünün nasıl kontrol edileceği sorusunun ilk adımı planlamadır. Diğer adımların başıdır, işin projelendirildiği, fikrin ortaya çıkarıldığı ve tartışıldığı kısımdır. Yazılım yaşam döngüsünün giriş kısmıdır. Temel ihtiyaçların belirlendiği proje için fizibilite çalışmalarının yapıldığı alandır. Proje planı oluşturulur. İlk olarak gereksinim analizi yapılır. Gereksinim analizi müşterinin istekleri doğrultusunda ortaya çıkacak olan yazılım hakkında ilk bilgilerin alındığı kısımdır. Bu adımda müşteri net bir şekilde dinlemelidir.

**Çözümleme(Analiz):**

Genelde istenilen fikrin ne olduğu ve temel analiz üzerinde konuşulacak kavramların tanımlandığı aşamadır. Çünkü aynı fikirden bahsedilmiyorsa farklı sonuçlara varılabilir. Bu analiz üzerine bir tasarım vardır. Bu analiz aşaması olarak da düşünülebilir. Problemin tanımlandığı veya yaşam döngüsünün tanımlandığı sistemin veyahut yazılımın tanımlandığı aşamadır. Analiz aşamasında projede nelerin istenildiği ile ilgili analiz çalışmaları da yapılabilir.

**Tasarım:**

Yazılımımızın veya sistemimizin tasarımları yapılır. Projeleri çizilir. Planlama ve tanımlaya göre bir tasarım çizilir. Kararlar verilir, seçimler yapılır, örneğin yazılımın ekranları, ekranlarda neler bulunacağı, hangi ekranlara nasıl geçileceği, fonksiyonel olarak hangi adımların oluşturulacağı, modülleri bu aşamada tasarlanır. Proje planının yansıra tasarım dokümanı da oluşturulmalıdır. Tasarım dokümanında proje bilgileri (amaç, kapsam vs.), sistem tasarım bilgileri, tasarım detayları, veri modeli, kullanıcı ara yüz tasarımları, UML Diyagramları. Tasarım dokümanının amacı, yazılım geliştiricinin yazılımını geliştirirken referans alacağı ve proje sürecinde/sonrasında projeye dahil olacak yeni yazılımcıların projeyi daha kolay anlayabilmesini sağlayacak teknik bir belgelendirmeye sahip olması gerekliliğidir. Bir sonraki adıma herhangi bir açık eksik olmadan geçilmesi beklenir. Geliştirme aşamasında herhangi bir soru veya karar bırakılmaz.

**Üretim(Gerçekleştirim):**

“Planlama, analiz ve tasarımı tamamlanıp yapılacak işlemleri detaylı olarak belirlenen projenin geliştirme aşamasıdır. Bu noktadan sonra herhangi bir analiz işlemi yapılmamalı. Üretim aşamasında yazılımcı alfa testlerini de gerçekleştirmektedir.” Tasarım aşamasında belirtilen detaylar, günler, başlangıç ve bitiş tarihleri kapsamında projenin, toplam başlangıç tarihinde başlanır ve bitiş tarihinde tamamlanacak şekilde kodlaması ve alfa testleri gerçekleştirilir. Bir anlamda daha önceden mimari projesi tamamlanmış inşaatın gerçekten yapılmaya başladığı aşamadır, yazılım projelerinin kodlandığı aşamadır.

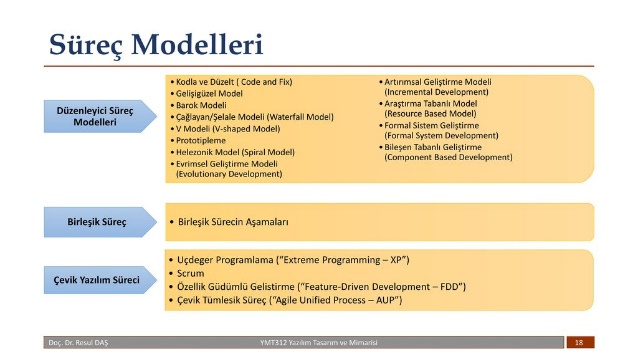
**Bakım:**

Proje yayına alındıktan sonra oluşabilecek hataların giderilmesi, yazılımın iyileştirilmesi ve yeni işlevlerin eklenmesi süreçleridir. Bu süreç zarfında kullanıcılardan gelen bilgiler doğrultusunda bu istekler gerçekleştirilmektedir. Yeni tanımlar ortaya çıkar, yeni tasarımlar yapılır ve bu şeklinde devam eden döngü hiçbir zaman bitmez. Yazılım tamamlandıktan ve yayına alındıktan (müşteri kullanmaya başladıktan) sonra yazılıma eklenmesi gereken yerler olduğu belirtildi: Uçak radardan çıkarsa son sinyalin yollanması gereken anlarda gecikme yaşandığı. Bu durumda gelen yeni talepler için yazılım geliştirme yaşam döngüsü bu istekler için başa döner ve yeni istekler için

Planlama> Analiz> Tasarım> Üretim> Test> Bakım; aşamaları baştan alınır.

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:



**Düzenleyici Süreç modelleri:**

Bunlar yazılım üretim faaliyetinin genel yapılma düzenine dair rehberler olarak adlandırılırlar. Bu modeller asla süreçlere ilişkin ayrıntılarla veyahut da süreçler arası ilişkilerle de ilgilenmezler.

•Kodla ve Düzelt (Code and Fix)

•Gelişigüzel Model

•Barok Modeli

•Çağlayan/Şelale Modeli (Waterfall Model)

•V Modeli (V-shaped Model)

•Prototipleme

•Helezonik Model (Spiral Model)

•Evrimsel Geliştirme Modeli (Evolutionary Development)

•Artırımsal Geliştirme Modeli (Incremental Development)

•Araştırma Tabanlı Model (Resource Based Model)

•Formal Sistem Geliştirme (Formal System Development)

•Bileşen Tabanlı Geliştirme (Component Based Development)

**Birleşik Süreç:**

Genellikle yazılım geliştirme süreçlerinin en iyi özelliklerinin bir araya getirilerek bütünleştirilmiş bir yazılım geliştirme sürecidir.

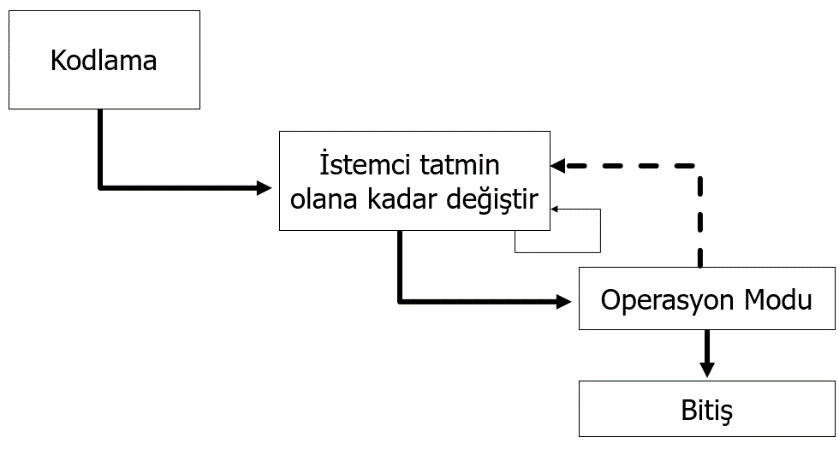
**Çevik Yazılım Süreci:**

•Uç değer Programlama (“Extreme Programming – XP”)

•Scrum

•Özellik Güdümlü Geliştirme (“Feature-Driven Development – FDD”)

•Çevik Tümleşik Süreç (“Agile Unified Process – AUP”)

**Kodla Ve Düzelt(Code And Fix):** 

Biri kodlamayı öğrendiğinde, ilk adımlar, kod yazma ve hata ayıklama ile çok sayıda deney yapmayı içerir. Hata ayıklamanın, program hatalarını bulma ve kaldırma rutin uygulaması olduğunu unutmayın. Öğrenme süreci genellikle programı yazarak ve ardından tüm yazılım hatalarını bulup düzelterek başlar. Bu, yazılım mühendisleri tarafından nispeten kolay ve kısa vadeli zorlukların üstesinden gelmek için ortaya çıkan yaygın bir yaklaşımdır. Ancak zamanla bu prosedür, teknolojik ürün geliştirmede de yaygın bir metodoloji haline geldi.Code and Fix (genellikle Kovboy kodlaması olarak bilinir) en basit ürün geliştirme süreci olarak kabul edilir ve yazılım odaklı teknoloji çözümlerinin prototipini oluşturmak için varsayılan yöntemdir. Kodlama ve Sabitleme olmak üzere iki ana adımdan oluşan döngüsel bir süreçtir. Bu modelde tipik olarak kapsamlı bir ürün planlaması (bazen çok basit olan), somut strateji veya iyi tanımlanmış tasarım düşüncesi yoktur.Sürecin kilit katılımcıları, iş, tasarım veya diğer ilgili disiplinlerden çok az veya hiç rehberlik almadan işten sorumlu olan yazılım mühendisleri olan kovboy kodlayıcılardır. Mühendisler, ürün ekibiyle bazı ilk görüşmelerin ardından Kodlama sürecine başlayacak. Düzeltme genellikle yol boyunca gerçekleşir, böylece ekip proje üzerinde çalışırken büyük hataları düzeltebilir. Code and Fix, araştırma ve geliştirmeyi (Ar-Ge) içeren mühendislik faaliyetleri gibi geliştirmenin ilk aşamalarında olan teknoloji odaklı çözümler için çok uygun olabilir. Örneğin, proje sonuçlarının belirsiz ve öngörülemez olabileceği makine öğrenimi ile ilgili araştırma programlarında çalışan kişiler, bu modeli ihtiyaçları için yeterli bulabilirler. Ek olarak, yazılım mühendisleri yeni teknolojileri ve geliştirme araçlarını öğrenirken bu modeli kullanmayı severler. Yaklaşan geliştirme faaliyetlerine katkıda bulunmak için yeterli uzmanlığı hızla kazanmalarını sağlar. Ayrıca model, mühendislerin düşünme kapsamlarını daraltmalarına ve yalnızca işleriyle en alakalı şeylere odaklanmalarına olanak tanır. Bu, teknoloji ve ürün engellerini geçici olarak unutmalarına ve o an için en önemli şeyleri oluşturmaya odaklanmalarına izin verebilir. Kodla ve Düzelt bazen zaman kazandırıcı olarak kabul edilir ve düşük bütçeli ürünler için uygun bir seçenektir. Kodlama becerilerini geliştirmek için deney, gözlem ve öğrenme ile ilgilenen birçok giriş seviyesi mühendisin bulunduğu yazılım gruplarında çok yaygındır. Bunun bir örneği, öğrencilerin derslerinin temel yönlerini uygulamak için ödevler aldığı, üniversite düzeyindeki Bilgisayar Bilimi programlarındaki bireysel veya grup projeleri olabilir. Bazı yazılım mühendisleri, yönetim kadrosundan herhangi bir resmi talimat olmaksızın geliştirme süreci üzerinde tam özerklik aldıkları için bu aktiviteden keyif alabilirler. Bu ortamda mühendisler, herhangi bir zorunlu yazılım teslim yapısı olmadan, genellikle projenin oluşturma akışını, kapsamını, uygulama ayrıntılarını, teknoloji yığınını ve geliştirme araçlarını kontrol eden kişilerdir. Bir anlamda, bu mühendislik yaratıcılığını artırabilir. Ayrıca, Kod ve Düzeltme modeli, yazılımın belirli mühendislik yönleri üzerinde bağımsız olarak çalışan deneyimli yazılım mühendisleri, özellikle de üstesinden gelinmesi zor ve çok fazla deneme yanılma gerektiren problemler üzerinde çalışan bireysel katkıda bulunanlar için uygun olabilir. İlk bakışta Code and Fix modeli, ürün geliştirme süresini kısaltıyor gibi görünebilir. Bununla birlikte, ürün gereksinimleri ve başarının tanımı (DOS) konusunda somut bir anlayış bulunmadığından, müşteri memnuniyetini elde edememe gibi büyük bir risk içerir. Ayrıca, zaman kısıtlamaları ve kalite güvence prosedürlerinin eksikliği nedeniyle, projenin kalite ile ilgili sorunları da olabilir. Ayrıca model, orta ölçekli ve büyük ürün ekipleri için iyi çalışacak şekilde tasarlanmamıştır Deneysel doğası nedeniyle, Kodla ve Düzelt modelini takip ederken teslimat işini tahmin etmek de zordur. Arzu edilen hedefleri gerçekleştirmek için tamamlanması gereken iş miktarını olduğundan fazla veya az tahmin edebilirler. Ürün ekibinin kendisini daha hızlı uygulamaya geçmeye çalışması durumunda, geliştirmenin sonraki aşamalarında okunması zor olan çirkin bir koda neden olabilir. Ayrıca, yeni geliştirmeler ve özellik güncellemeleri, ürünün büyük bölümlerinin yeniden yazılması gerekebileceğinden, daha sonra önemli mimari sorunlara neden olabileceğinden, gelecekte değişiklik gerektiren ürünler için Kod ve Düzeltme kullanmak son derece riskli olabilir. Bunun nedeni, teknik prototiplerin çoğu zaman yeterince kararlı olarak kabul edilmemesidir.

**Gelişigüzel Model:**

Gelişigüzel Model aslında bir model değildir çünkü herhangi bir modeli veya yöntemi yoktur. Geliştiren kişinin izlediği yollardan oluşur ve kişiden kişiye göre değiştiği için bir standart yoktur.Geliştiren kişiye bağımlı (belli bir süre sonra o kişi bile sistemi anlayamaz ve geliştirme güçlüğü yaşar).

İzlenebilirliği ve bakımı oldukça zor.Bakımı zordur.Bu model, yazılıma yeni başlayanların kullandığı adımlara verilen model adıdır.Miladi olarak 60lı yıllarda ortaya çıkmıştır.

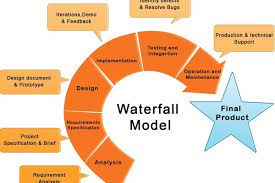
**Barok Model:**

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yaşam döngü adımları temel olarak doğrusal bir şekilde gözden geçirilir. Döngü yoktur ve belgeleme ayrı bir süreç olarak ele alınır. 1970’lerde kullanılmıştır.Bu iki model de günümüzde kullanılmamaktadır. Bu yüzden de aslında güncel bir yaşam döngü modeli olarak adlandırılmazlar.

**Çağlayan/Şelale Modeli (Waterfall Model):**

****

Şelale Modeli, yazılım geliştirme sürecini doğrusal bir sıralı akışta gösterir. Bu, geliştirme sürecindeki herhangi bir aşamanın ancak önceki aşama tamamlandığında başlayacağı anlamına gelir. Bu şelale modelinde fazlar örtüşmemektedir. Şelale yaklaşımı, projenin başarısını sağlamak için Yazılım Mühendisliğinde yaygın olarak kullanılan ilk SDLC Modelidir. "Şelale" yaklaşımında, yazılım geliştirme sürecinin tamamı ayrı aşamalara ayrılmıştır. Bu Şelale modelinde, tipik olarak, bir aşamanın sonucu, sıralı olarak bir sonraki aşama için girdi görevi görür. Şelale modelinde sıralı aşamalar şunlardır:

Gereksinim Toplama ve Analiz: Geliştirilecek sistemin tüm olası gereksinimleri bu aşamada yakalanır ve bir gereksinim belirtimi belgesinde belgelenir.

Sistem Tasarımı: Bu aşamada ilk aşamadaki gereksinim özellikleri incelenir ve sistem tasarımı hazırlanır. Bu sistem tasarımı, donanım ve sistem gereksinimlerinin belirlenmesine ve genel sistem mimarisinin tanımlanmasına yardımcı olur.

Uygulama:Sistem tasarımından gelen girdilerle, sistem ilk önce bir sonraki aşamada entegre edilen birimler adı verilen küçük programlarda geliştirilir. Her birim, Birim Testi olarak adlandırılan işlevselliği için geliştirilir ve test edilir.

Entegrasyon ve Test Etme:Uygulama aşamasında geliştirilen tüm birimler, her bir birimin test edilmesinden sonra bir sisteme entegre edilir. Entegrasyon sonrası tüm sistem herhangi bir hata ve arıza için test edilir.

Sistemin konuşlandırılması: İşlevsel ve işlevsel olmayan testler yapıldıktan sonra; ürün müşteri ortamında konuşlandırılır veya piyasaya sürülür.

Bakım:İstemci ortamında ortaya çıkan bazı sorunlar var. Bu sorunları gidermek için yamalar yayınlanır. Ayrıca ürünü geliştirmek için bazı daha iyi sürümler yayınlanmıştır. Müşteri ortamında bu değişiklikleri sağlamak için bakım yapılır.Tüm bu aşamalar, aşamalar boyunca sürekli olarak aşağıya doğru (bir şelale gibi) akıyor olarak görülen ilerlemenin görüldüğü birbiri ardına dizilmiştir. Bir sonraki aşama, ancak bir önceki aşama için belirlenen hedeflere ulaşıldıktan ve imzalandıktan sonra başlatılır, bu nedenle "Şelale Modeli" adı verilir. Bu modelde fazlar çakışmaz.

Şelale Modeli:Uygulama Geliştirilen her yazılım farklıdır ve iç ve dış faktörlere bağlı olarak izlenecek uygun bir SDLC yaklaşımı gerektirir. Şelale modelinin kullanımının en uygun olduğu bazı durumlar şunlardır:

Gereksinimler çok iyi belgelenmiştir, açık ve sabittir.

Ürün tanımı sabittir.

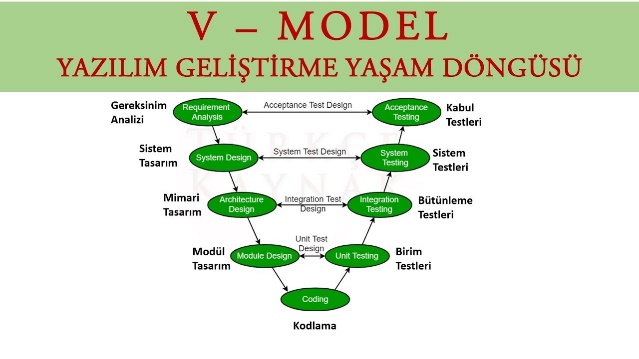
Teknoloji anlaşılır ve dinamik değildir.

Belirsiz gereklilikler yoktur.

Ürünü desteklemek için gerekli uzmanlığa sahip geniş kaynaklar mevcuttur.

Proje kısa.

**V-Shaped Model(V-Model):**



V modeli, işlemin V şeklinde sıralı bir şekilde yürütüldüğü bir tür SDLC modelidir. Doğrulama ve Doğrulama modeli olarak da bilinir. Karşılık gelen her bir geliştirme aşaması için bir test aşamasının ilişkilendirilmesine dayanır. Test aşamasıyla doğrudan ilişkili her adımın geliştirilmesi. Bir sonraki aşama ancak bir önceki aşamanın tamamlanmasından sonra başlar, yani her geliştirme faaliyeti için ona karşılık gelen bir test faaliyeti vardır.

V-Modeli

Doğrulama: Kod çalıştırmadan yapılan statik analiz tekniğini (inceleme) içerir. Belirtilen gereksinimlerin karşılanıp karşılanmadığını bulmak için ürün geliştirme aşamasının değerlendirilmesi sürecidir.

Gereksinim Analizi: Bu aşama, gereksinimlerini ve beklentilerini anlamak için müşteri ile ayrıntılı iletişimi içerir. Bu aşama Gereksinim Toplama olarak bilinir.

Sistem Tasarımı: Bu aşama, ürün geliştirmek için sistem tasarımını ve eksiksiz donanım ve iletişim kurulumunu içerir.

Mimari Tasarım: Sistem tasarımı, farklı işlevleri üstlenen modüllere ayrılmıştır. Dahili modüller ve dış dünya (diğer sistemler) arasındaki veri aktarımı ve iletişim açıkça anlaşılmaktadır.

Modül Tasarımı: Bu aşamada sistem küçük modüllere ayrılır. Modüllerin ayrıntılı tasarımı, Düşük Seviyeli Tasarım (LLD) olarak da bilinir.

Test Aşamaları:

Birim Testi: Modül tasarım aşamasında Birim Test Planları geliştirilir. Bu Birim Test Planları, kod veya birim düzeyindeki hataları ortadan kaldırmak için yürütülür.

Entegrasyon testi: Birim testi tamamlandıktan sonra Entegrasyon testi yapılır. Entegrasyon testinde modüller entegre edilir ve sistem test edilir. Entegrasyon testi, Mimari tasarım aşamasında gerçekleştirilir. Bu test, modüllerin kendi aralarındaki iletişimini doğrular.

Sistem Testi: Sistem testi, uygulamanın işlevselliği, karşılıklı bağımlılığı ve iletişimi ile birlikte test eder. Geliştirilen uygulamanın işlevsel ve işlevsel olmayan gereksinimlerini test eder.

Kullanıcı Kabul Testi (UAT): UAT, üretim ortamına benzeyen bir kullanıcı ortamında gerçekleştirilir. UAT, teslim edilen sistemin kullanıcının gereksinimlerini karşıladığını ve sistemin gerçek dünyada kullanıma hazır olduğunu doğrular.

Endüstriyel Zorluk: Endüstri geliştikçe, teknolojiler daha karmaşık hale geldi, giderek daha hızlı hale geldi ve sonsuza dek değişti, ancak BT'nin emekleme döneminde olduğu kadar bugün de geçerli olan bir dizi temel ilke ve kavram var.

Kullanıcı gereksinimlerini doğru bir şekilde tanımlayın ve iyileştirin.

Yetkili kullanıcı gereksinimlerine göre bir uygulama tasarlayın ve oluşturun.

Oluşturdukları uygulamanın yetkili iş gereksinimlerine uygun olduğunu doğrulayın.

V-Model Prensipleri:

Büyükten Küçüğe: V-Model'de testler hiyerarşik bir perspektifte yapılır, Örneğin proje ekibi tarafından belirlenen gereksinimler, projenin Üst Düzey Tasarım oluşturma ve Ayrıntılı Tasarım aşamaları. Bu aşamaların her biri tamamlandığında gereksinimler giderek daha rafine ve ayrıntılı hale geliyor.

Veri/Süreç Bütünlüğü: Bu ilke, herhangi bir projenin başarılı tasarımının hem verilerin hem de süreçlerin dahil edilmesini ve uyumunu gerektirdiğini belirtir.

Ölçeklenebilirlik: Bu ilke, V-Model konseptinin boyutu, karmaşıklığı veya süresi ne olursa olsun herhangi bir BT projesini barındırma esnekliğine sahip olduğunu belirtir.

Çapraz Referanslama: Gereksinimler ve ilgili test faaliyeti arasındaki doğrudan ilişki, çapraz referanslama olarak bilinir.

Somut Belgeleme: Bu ilke, her projenin bir belge oluşturması gerektiğini belirtir. Bu belgelendirme hem proje geliştirme ekibi hem de destek ekibi tarafından istenmekte ve uygulanmaktadır. Belgeler, bir üretim ortamında kullanıma sunulduktan sonra uygulamanın bakımı için kullanılır.

Neden tercih edilir?

Modelin sertliği nedeniyle yönetimi kolaydır. V-Model'in her aşamasının belirli çıktıları ve bir inceleme süreci vardır.

Proaktif kusur takibi – yani kusurlar erken aşamada bulunur.

Ne zaman kullanılır?

Gereksinimlerin açıkça tanımlandığı ve sabitlendiği yerlerde.

V-Model, teknik uzmanlığa sahip bol miktarda teknik kaynak mevcut olduğunda kullanılır.

Avantajlar:

Bu oldukça disiplinli bir modeldir ve Aşamalar birer birer tamamlanır.

V-Model, proje gereksinimlerinin net olduğu küçük projeler için kullanılır.

Basit ve anlaşılması ve kullanılması kolaydır.

**Prototipleme:**

Prototip oluşturma modeli, bir prototipin oluşturulduğu, test edildiği ve ardından tüm sistemin veya ürünün geliştirilebileceği kabul edilebilir bir sonuç elde edilene kadar gerektiği şekilde yeniden çalışıldığı bir sistem geliştirme yöntemidir. Bu model, tüm proje gereksinimlerinin önceden ayrıntılı olarak bilinmediği senaryolarda en iyi sonucu verir. Geliştiriciler ve kullanıcılar arasında gerçekleşen yinelemeli, deneme yanılma sürecidir. Prototipleme modelinin adımları

Yeni sistem gereksinimleri mümkün olduğunca ayrıntılı olarak tanımlanır. Bu genellikle, mevcut sistemin tüm departmanlarını veya yönlerini temsil eden bir dizi kullanıcıyla görüşmeyi içerir.

Yeni sistem için ön, basit bir tasarım oluşturulur.

Yeni sistemin ilk prototipi, ön tasarımdan yapılmıştır. Bu genellikle küçültülmüş bir sistemdir ve nihai ürünün özelliklerinin bir yaklaşıklığını temsil eder.Kullanıcılar ilk prototipi kapsamlı bir şekilde değerlendirir ve güçlü ve zayıf yönlerini, nelerin eklenmesi ve nelerin kaldırılması gerektiğini not eder. Geliştirici, kullanıcılardan gelen yorumları toplar ve analiz eder. İlk prototip, kullanıcılar tarafından sağlanan yorumlara göre değiştirilir ve yeni sistemin ikinci bir prototipi oluşturulur.

İkinci prototip, ilk prototiple aynı şekilde değerlendirilir.Kullanıcılar, prototipin istenen nihai ürünü temsil ettiğinden emin olana kadar, önceki adımlar gerektiği kadar yinelenir.Nihai sistem, nihai prototipe dayalı olarak inşa edilir.Nihai sistem kapsamlı bir şekilde değerlendirilir ve test edilir. Büyük ölçekli arızaları önlemek ve arıza süresini en aza indirmek için rutin bakımlar sürekli olarak gerçekleştirilir. Prototip model türleri

Hızlı atılan: Bu yöntem, daha sonra müşteri geri bildirimi ile revize edilen ön gereksinimlere dayalı bir prototipi hızla geliştirerek fikirleri keşfetmeyi içerir. Hızlı atma adı, her bir prototipin tamamen atıldığı ve nihai ürünün bir parçası olmayabileceği gerçeğine atıfta bulunur.

Evrimsel: Bu yaklaşım, müşteri geri bildiriminin her yinelemesinden sonra rafine edilen sürekli, çalışan bir prototip kullanır. Her prototip sıfırdan başlatılmadığından, bu yöntem zamandan ve emekten tasarruf sağlar.

Artımlı:Bu teknik, nihai ürün konseptini daha küçük parçalara ayırır ve her biri için prototipler oluşturulur. Sonunda, bu prototipler nihai üründe birleştirilir.

Extreme: Bu prototip modeli, özellikle web geliştirme için kullanılır. Tüm web prototipleri, hizmetler katmanıyla birlikte bir HTML biçiminde oluşturulur ve ardından nihai ürüne entegre edilir.

Prototipleme modelinin avantajları

Bir prototip model kullanmak, aşağıdakiler dahil birçok avantaj sağlayabilir:

Müşteriler üründe erkenden söz sahibi olur ve müşteri memnuniyetini artırır.

Eksik işlevsellik ve hatalar kolayca tespit edilir.

Prototipler gelecekte daha karmaşık projelerde yeniden kullanılabilir.

Ekip iletişimini ve esnek tasarım uygulamalarını vurgular.

Kullanıcılar ürünün nasıl çalıştığını daha iyi anlıyor.

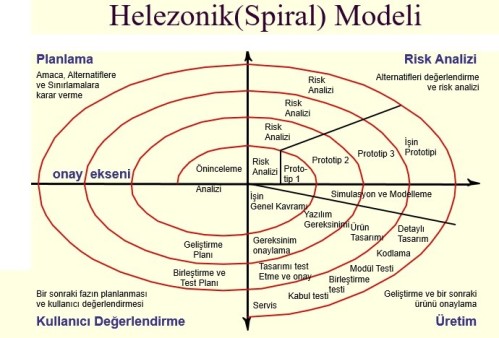
Daha hızlı müşteri geri bildirimi, müşteri ihtiyaçları hakkında daha iyi bir fikir sağlar.

Prototipleme modelinin dezavantajları

Bu metodolojinin ana dezavantajı, spiral veya Şelale modeli gibi alternatif geliştirme yöntemlerine kıyasla zaman ve para açısından daha maliyetli olmasıdır. Çoğu durumda prototip atıldığından, bazı şirketler bu yaklaşımı benimsemenin değerini görmeyebilir.

Ek olarak, geliştirme yaşam döngüsünün başlarında müşteri geri bildirimlerini davet etmek sorunlara neden olabilir. Sorunlardan biri, karşılanması zor olabilecek aşırı miktarda değişiklik talebinin olmasıdır. Müşteri prototipi gördükten sonra daha hızlı bir son sürüm talep ederse veya ürüne ilgi duymazsa başka bir sorun ortaya çıkabilir.

**Helezonik Model (Spiral Model):**



Spiral model, yinelemeli geliştirme fikrini şelale modelinin sistematik, kontrollü yönleriyle birleştirir.Ürünün kademeli olarak serbest bırakılmasına veya spiral etrafındaki her yinelemede kademeli iyileştirmeye izin verir.

Spiral modelin dört aşaması vardır. Bir yazılım projesi, Spiraller adı verilen yinelemelerde bu aşamalardan tekrar tekrar geçer.

Kimlik

Bu aşama, temel spiralde iş gereksinimlerinin toplanmasıyla başlar. Ürün olgunlaştıkça sonraki sarmallarda sistem gereksinimlerinin, alt sistem gereksinimlerinin ve birim gereksinimlerinin belirlenmesi bu aşamada yapılır.Bu aşama ayrıca müşteri ve sistem analisti arasında sürekli iletişim yoluyla sistem gereksinimlerinin anlaşılmasını da içerir. Spiralin sonunda, ürün belirlenen pazarda konuşlandırılır.

Tasarım

Tasarım aşaması, temel spiraldeki kavramsal tasarımla başlar ve mimari tasarım, modüllerin mantıksal tasarımı, fiziksel ürün tasarımı ve sonraki spirallerdeki nihai tasarımı içerir.

İnşa Et

Oluşturma aşaması, her sarmalda gerçek yazılım ürününün üretilmesini ifade eder. Temel sarmalda, ürün henüz düşünüldüğünde ve tasarım geliştirilirken, bu aşamada müşteri geri bildirimi almak için bir POC (Kavram Kanıtı) geliştirilir.

Ardından, gereksinimler ve tasarım ayrıntıları konusunda daha net olan sonraki sarmallarda, bir sürüm numarasıyla inşa adı verilen yazılımın çalışan bir modeli üretilir. Bu yapılar geri bildirim için müşteriye gönderilir.

Değerlendirme ve Risk Analizi

Risk Analizi, program kayması ve maliyet aşımı gibi teknik fizibilite ve yönetim risklerinin tanımlanmasını, tahmin edilmesini ve izlenmesini içerir. Yapıyı test ettikten sonra, ilk yinelemenin sonunda müşteri yazılımı değerlendirir ve geri bildirim sağlar.

SDLC Spiral Modeli

Müşteri değerlendirmesine dayalı olarak, yazılım geliştirme süreci bir sonraki yinelemeye girer ve ardından müşteri tarafından önerilen geri bildirimi uygulamak için doğrusal yaklaşımı takip eder. Spiral boyunca yineleme süreci, yazılımın ömrü boyunca devam eder.

Spiral Model Uygulaması

Spiral Model, herhangi bir ürünün doğal gelişim süreci, yani müşteri ve geliştirme firmaları için minimum risk içeren olgunlukla öğrenme ile senkronize olduğu için yazılım endüstrisinde yaygın olarak kullanılmaktadır.

Bir bütçe kısıtlaması olduğunda ve risk değerlendirmesi önemlidir.

Orta ve yüksek riskli projeler için.

Gereksinimler zamanla değiştikçe ekonomik önceliklerdeki olası değişiklikler nedeniyle uzun vadeli proje taahhüdü.Müşteri, genellikle böyle olan gereksinimlerinden emin değildir.

Gereksinimler karmaşıktır ve netlik elde etmek için değerlendirmeye ihtiyaç duyar.

Yeterli müşteri geri bildirimi almak için aşamalar halinde piyasaya sürülmesi gereken yeni ürün grubu.

Geliştirme döngüsü sırasında üründe önemli değişiklikler beklenmektedir.

Spiral Model - Artıları ve Eksileri

Spiral yaşam döngüsü modelinin avantajı, ürünün öğelerinin mevcut olduklarında veya bilinir olduklarında eklenmesine izin vermesidir. Bu, önceki gereksinimler ve tasarımla herhangi bir çelişki olmamasını sağlar.Bu yöntem, bir bakım faaliyetine düzenli bir geçiş yapılmasına izin veren birden çok yazılım derlemesi ve sürümüne sahip yaklaşımlarla tutarlıdır. Bu yöntemin bir başka olumlu yönü, spiral modelin sistem geliştirme çabalarına erken bir kullanıcı katılımını zorlamasıdır.Öte yandan, bu tür ürünleri tamamlamak çok sıkı bir yönetim gerektirir ve spirali belirsiz bir döngüde çalıştırma riski vardır. Bu nedenle, değişim disiplini ve değişiklik taleplerinin alınma derecesi, ürünü başarılı bir şekilde geliştirmek ve dağıtmak için çok önemlidir.

Spiral SDLC Modelinin avantajları aşağıdaki gibidir -

Değişen gereksinimler karşılanabilir.

Prototiplerin kapsamlı kullanımına izin verir.

Gereksinimler daha doğru bir şekilde yakalanabilir.

Kullanıcılar sistemi erken görür.Geliştirme daha küçük parçalara bölünebilir ve riskli bölümler daha erken geliştirilebilir, bu da daha iyi risk yönetimine yardımcı olur.

Spiral SDLC Modelinin dezavantajları şunlardır:

Yönetim daha karmaşıktır.

Projenin sonu erkenden belli olmayabilir.

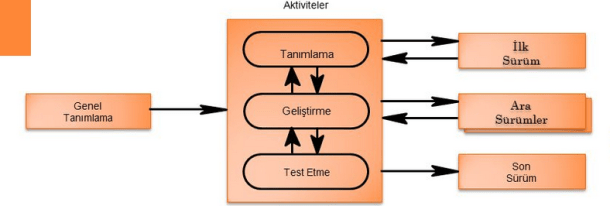
Küçük veya düşük riskli projeler için uygun değildir ve küçük projeler için pahalı olabilir.

Süreç karmaşık

Spiral süresiz olarak devam edebilir.

Çok sayıda ara aşama, aşırı belgelendirme gerektirir.

**Evrimsel Geliştirme Modeli (Evolutionary Development):**



Bu model ilk tam ölçekli olan modeldir. Projenin başarıya ulaşması ilk evrimin başarısına bağlıdır. “Ne istediğimi bilmiyorum ama görsem tanırım.” mantığına dayanır. Diğer modeller ile kıyaslandığında ilerleyişi daha yavaştır. Modelin tamamının başarımı geçirdiği ilk evrimin başarısına bağlıdır.

Avantajları

•Gereksinimi anlamayı kolaylaştırır.

•Risk az.

•Hata az.

Dezavantajları

•Bakım zor.

•Düzenli ürün oluşumu yok.

•Gereksinimi sürekli yenilemek gerekebilir.

**Artırımsal Geliştirme Modeli (Incremental Development):**

Artımlı geliştirme, sistemin çeşitli bölümlerinin farklı zamanlarda veya geliştirme hızlarında geliştirildiği ve tamamlanma zamanına bağlı olarak birleştirildiği bir sistem geliştiren bir yöntemdir. Yinelemeli geliştirmede (adım adım büyüme), geliştirme ekipleri, işlevselliklerini geliştirmek için sistemin parçalarını yeniden ziyaret eder.

Artımlı geliştirme, Başlangıç ​​Aşaması, Detaylandırma Aşaması, İnşaat Aşaması ve Geçiş Aşaması olmak üzere dört aşamaya ayrılmıştır. Çoğu insan aşamalı geliştirmeyi modern bir uygulama olarak görse de, 1950'lerin ortalarından beri kullanılmaktadır. Önde gelen yazılım programcıları ve mühendisleri, birbirini izleyen her on yılın liderlerini IID uygulamalarını destekledi ve birçok büyük proje bunları başarıyla kullandı. Bu uygulamalar ayrıntılarında gelişmiş olabilir, ancak hepsinin tek geçiş ardışık, arşiv odaklı, kapılı adım yaklaşımından uzak durmaya yönelik tipik bir konusu vardı.Artımlı Geliştirmenin Ortak Kullanımları

Artımlı geliştirme, farklı bölümlere ayrılmış aşamaları tekrar gözden geçirmeyi kolaylaştıran aşamalı büyümesi nedeniyle çok avantajlıdır.

Artımlı geliştirmede, aynı projenin farklı bölümleri için farklı tamamlanma süreleri vardır.

Tüm projenin genel bir bölümü olduğundan, aşamalı geliştirmede gelişim aşamalarını gruplamak kolaydır.

Artımlı Geliştirmenin Yaygın Yanlış Kullanımları

Artımlı geliştirmede, tüm bölümler aynı tamamlanma süresine ve hızına sahiptir.

**Araştırma Tabanlı Model (Resource Based Model):** Bu model yap-at prototipi olarak da bilinir. Araştırma ortamları bütünüyle belirsizlik üzerine çalışan ortamlardır. Yapılacak işlerden edinilecek sonuçlar belirgin değildir.

**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.

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.

**Bileşen Tabanlı Geliştirme (Component Based Development):** Bileşen tabanlı geliştirme (CBD), yeniden kullanılabilir yazılım bileşenlerinin yardımıyla bilgisayar tabanlı sistemlerin tasarımını ve geliştirilmesini vurgulayan bir prosedürdür. CBD ile odak noktası, yazılım programlamasından yazılım sistemi oluşturmaya geçer.Bileşen tabanlı geliştirme teknikleri, ideal hazır bileşenleri seçerek ve sonra bunları iyi tanımlanmış bir yazılım mimarisi kullanarak birleştirerek yazılım sistemlerinin geliştirilmesi için prosedürler içerir. Kaba taneli bileşenlerin sistematik olarak yeniden kullanılmasıyla CBD, daha iyi kalite ve çıktı sağlamayı amaçlamaktadır.Bileşen tabanlı geliştirme, bileşen tabanlı yazılım mühendisliği (CBSE) olarak da bilinir.

CBD'nin bazı avantajları şunları içerir:

Minimum teslimat:

Bileşen kataloglarında arama

Prefabrik bileşenlerin geri dönüşümü

Verimliliği arttırmak:

Geliştiriciler uygulama geliştirmeye odaklanıyor

Geliştirilmiş kalite:

Bileşen geliştiricileri kaliteyi sağlamak için ek süre tanıyabilir

Minimum harcamalar

**Çevik Yazılım Süreci:**

**Uç değer Programlama (“Extreme Programming – XP”):**

Aşırı programlama veya kısaca XP, programlama yapmadan önce testler yazarak kaliteyi vurgular. Yazılım yazıldığı gibi, kod günde birçok kez gözden geçirilir ve test edilir. Tüm yazılım bir araya getirilir ve günde bir veya birkaç kez test edilir. Bu, yazılımın hiçbir bölümünün sistemi çökertmemesini sağlamak için yapılır.Aşırı programlamanın bilinen yönlerinden biri, iki programcının bir bilgisayarda birlikte çalıştığı çift programlamadır. Bu yaklaşım sayesinde yazılımın yaratılmasına giren düşünce süreci, tasarlama ve kodlama anında kontrol edilir. Herhangi bir yazılım geliştirme sorumluluğu paylaşılmaktadır.Bu geliştirme yöntemiyle, yazılım basit bir tasarımla başlar ve ek iş gereksinimlerini karşılamak için gelişir. Özelliklerin oluşturulması, gerçekten yüksek öncelik haline gelinceye kadar önlenir. Vurgu tam sistemler oluşturmak için ekip çalışmasına odaklanır, bu nedenle her ekip üyesi her gün analist, tasarımcı, programcı ve testçi rolünü oynar. Programlama ekibi içinde ve programcılar ile yazılımın teslim edildiği kullanıcılar arasında yoğun bir iletişim vardır.

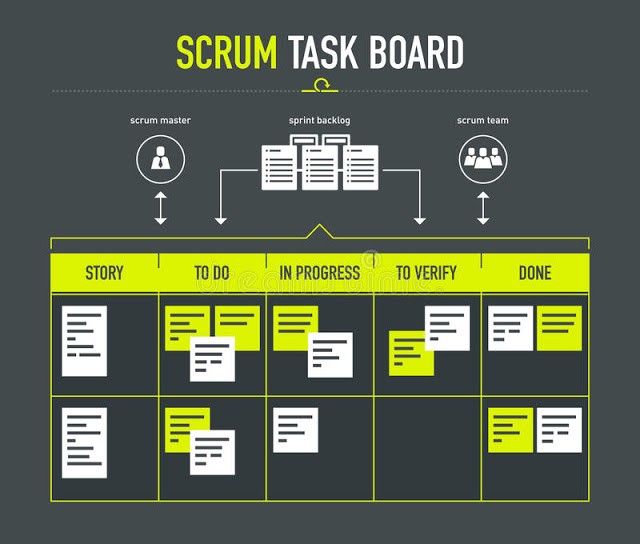
**Scrum:**

Süreçten ziyade bireylerin daha etkileşimli olmalarına, kapsamlı dökümantasyonlardan ziyade çalışan yazılıma, sözleşme pazarlığından ziyade müşteri ile işbirliğine, bir plana bağlı kalmaktan ziyade değişime karşılık vermeye dayanır. Kompleks yazılım süreçlerinin yönetilmesi için kullanılır. Bunu yaparken bütünü parçalayan; tekrara dayalı bir yöntem izler. Düzenli geri bildirim ve planlamalarla hedefe ulaşmayı sağlar. Bu anlamda ihtiyaca yönelik ve esnek bir yapısı vardır. Müşteri ihtiyacına göre şekillendiği için müşterinin geri bildirimine göre yapılanmayı sağlar. İletişim ve takım çalışması çok önemlidir

Şeffaflık

Denetleme

Uyarlama.



Scrum Yapısı

1)Product Backlog; Proje için gerekli olan gereksinimler listesidir. Proje sonunda “Ne üretilmek isteniyor?” sorusuna cevap aranır. Product owner tarafından müşteriden gereksinimler alınır, öncelik sırasına göre sıralanır. Product owner, değişen ihtiyaçlara göre product backlog’a ekleme veya çıkarma yapabilir.

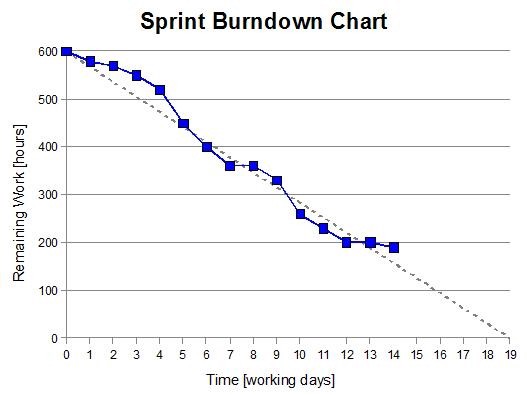
2)Product Backlog Item; Product backlog içindeki her bir gereksinime verilen isimdir

3)Sprint(Koşma); Scrum içerisindeki tüm aktiviteler sprint içerisinde gerçekleşir.

4)Sprint Backlog; Bir sprint boyunca yapılacak itemların listesini oluşturur. İşlerin detaylı olarak zaman çizelgesi çıkarılır.

5)Scrum board; Yapılacak olan tasklar “TO DO” bölümüne alınır. Takım üyesi bu işe başladığında “IN PROGRESS” bölümüne getirilir. Bir iş, test için hazırsa “TO VERIFY” durumuna getirilir. İş, kontrol edildikten sonra “DONE” bölümüne getirilir.

6)Burndown Chart; Yatay ekseninde sprintin günlerini, dikey ekseninde sprintte kalan işi gösteren grafiktir.



Rollere gelirsek ilk olarak bir ürün Sahibi (Product Owner) olmalıdır. Bu kişi illa müşterinin kendisi olmak zorunda değildir. Bir müşteri temsilcisi veya Scrum ekibinin bir parçası da olabilir.

Scrum yöneticisi takımlar ve ürün sahibiyle olan iletişime yardımcı olan, takımın ve organizasyonun Scrum’a adapte olmasını sağlayarak üretkenliğin artmasına katkıda bulunan kişidir.

Scrum Takımı (Scrum Team) , birbirleriyle iletişim halinde olan ve programlama işini yapan gruptur.

Toplantılar:

Sprint Planning de sprintlerde neler yapılacağı, gereksinimlerin listesi, takımların belirlenmesi, risk analizleri ve maliyet hesaplamaları yapılır.

Tüm sprint boyunca her gün takım üyeleri bir araya gelerek 15 dakika boyunca ayaküstü toplantılar yaparlar. Bu toplantılara “Daily Scrum Meeting” denir. Bu toplantılarda önceki gün neler yapıldığı, ne gibi sorunlarla karşılaşıldığı, o gün neler yapılacağı ve nasıl yapılacağı hakkında özet şeklinde konuşmalar yapılır.

Sprint Listesi (Sprint Backlog), bir sprint turunda takımın yapması gereken işlerin listesidir.

Sprint Kalan Zaman Grafiği (Burndown Chart), bir sprint turunda görevler yerine getirildikçe yapılan iş ile kalan iş arasındaki bağıntıyı ortaya çıkaran grafiktir.

**Scrum günümüzde neden popüler?**Scrum, uzmanlık gerektiren ve maliyeti yüksek bir modeldir. Ancak kısa sürmesi, kolay uygulanması ve başarı garantisinin yüksek olması nedeniyle günümüzde kullanımı oldukça artmaktadır. Google, Microsoft, IBM ve Yahoo gibi büyük şirketler tarafından tercih edilmektedir.

**KAYNAKLAR:**

1. <https://www.computerhope.com/jargon/r/rtm.htm>
2. <https://www.productplan.com/glossary/general-availability/>
3. <https://caglartelef.com/yazilim-yasam-dongusu/>
4. <https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/>
5. <https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm#:~:text=The%20Waterfall%20model%20is%20the,the%20previous%20phase%20is%20complete>.
6. <https://www.codex.com.tr/yazilim-gelistirme-modelleri>
7. <https://productcoalition.com/the-code-and-fix-model-2cabd4c48166>
8. <https://talentgrid.io/tr/yazilim-gelistirme-modelleri/>
9. <https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>

**HESAPLAR:**

1. [**https://www.linkedin.com/in/beyza-nur-karako%C3%A7-386797229**](https://www.linkedin.com/in/beyza-nur-karako%C3%A7-386797229)
2. [**https://github.com/Beyzanurkarakoc**](https://github.com/Beyzanurkarakoc)
3. [**https://medium.com/@beyzanurkarakoc\_ceng**](https://medium.com/@beyzanurkarakoc_ceng)