**ÖZET**

Bilgisayarların ortaya çıkmasıyla yazılım geliştirme süreçleri başlamıştır. Geliştirilmeye başlanan yazılımlarda görülen iki büyük eksik; yazılım projelerinin zamanında teslim edilememesi ve istenilen kalitede (dokümantasyon, fonksiyonellik ve harcanan fazla iş gücü bakımından) olmamasıydı. Bunlara ek olarak teknoloji hızla gelişti, değişti. Bu sebeple birçok yazılım hız ve fonksiyonellik bakımından eksik kalmalarıyla tekrar projelendirilmek zorunda kalmıştır. İnternetin hızla gelişmesi ve yaygın bir şekilde kullanılmasıyla eksik kalmanın yanı sıra yazılımlar, içerdikleri güvenlik açıkları/zayıflıkları bakımından kötü amaçlı kullanıcılar sebebiyle günlük hayattaki işlemlerimizin (e-devlet, e-ticaret vb.) yazılımlar aracılığıyla sağlanmasını riskli hale getirmeye başlamıştır. [1]

Her ürünün geliştirme işlemi belirli aşamalar içermektedir. Yazılımların da birer ürün olduğu unutulmamalıdır. Nasıl bir bilgisayar üretilirken çeşitli aşamalardan bahsediyorsak yazılım için de benzer ve birçok aşama bulunmaktadır. Yazılımın hem üretim hem de kullanım süreci boyunca geçirdiği tüm aşamalar ‘’yazılım geliştirme yaşam döngüsü’’ olarak tanımlanır. Güvenlik, fonksiyonellik , zamanında teslim edilebilme ve hatalara geri dönebilme özellikleri ile yazılım yaşam döngüsü modelleri oldukça önemlidir. Yazılım işlevleri ve gereksinimleri sürekli olarak değiştiği ve genişlediği için söz konusu aşamalar sürekli bir döngü biçiminde ele alınmaktadır. Döngü içerisinde herhangi bir aşamada geriye dönebilmek ve tekrar ilerlemek mümkündür olabilmektedir. [2]

Ek olarak yazılım bir üründür ancak başka ürünler geliştirmek veya elde etmeye yaran bir araç görevi de görebilmektedir. Yazılımlar fiziksel ürünler olmadıkları için aşınmazlar fakat zamanla yetersiz hale gelebilirler. Büyük ve karmaşık sistemler geliştiren organizasyonların ürün geliştirme süreçlerini tanımlaması, sürekli iyileştirmesi gerekmektedir. Kaliteli ürün geliştiren organizasyonlar itibar, müşteri ve gelir kaybı riskleri ile karşılaşabilmektedir. Yazılım geliştirme modelleri sayesinde planlı çalışma ve yazılım projesini en iyi şekilde geliştirmek mümkün olmaktadır. [3]

Temel yazılım geliştirme aşamaları:

1. Gereksinim (Requirements)
2. Analiz (Analysis) - Çözümleme
3. Tasarım (Design)
4. Gerçekleştirme (Implementation)
5. Bakım(Maintenance) aşamalarından oluşmaktadır. Bu aşamalar açıklanacaktır.

Belli başlı yazılım geliştirme modelleri :

* Kodla ve Düzelt
* Şelale Modeli( Çağlayan / Geleneksel Yazılım Geliştirme Modeli)
* V Süreç Modeli
* Barok Modeli
* Evrimsel Geliştirme
* Prototipleme
* Spiral (Helezonik) Model
* Format Sistem Geliştirme
* Yeniden Kullanıma Yönelik Geliştirme
* Artımsal Geliştirme Süreç Modeli
* Birleşik Süreç
* Araştırma Tabanlı Model
* Çevik Modeller (Agile models: XP,SCRUM) modelleri açıklanacaktır.

Ek olarak aslında bir model olarak adlandırmanın doğru olmadığı Gelişigüzel Model’den de bahsedilecektir.

**Anahtar Kelimler:** *yazılım yaşam döngüsü, temel yazılım geliştirme aşamaları, yazılım geliştirme modelleri*

**YAZILIM YAŞAM DÖNGÜSÜ TEMEL ADIMLARI**

Yazılım yaşam döngüsü; yazılımın üretimi, geliştirilmesi, kullanımı ve bakım süreci boyunca geçirdiği tüm aşamalar olarak özetlenebilir. Yazılımın işlevleri ve ihtiyaçları sürekli değişmektedir bu sebeple bir döngü biçiminde düşünülmesi gerekmektedir. Tek yönlü ve doğrusal olarak kesinlikle düşünülmemesi gerekmektedir.

Temel yazılım geliştirme aşamaları : Gereksinim, analiz, tasarım, gerçekleştirme ve bakım adımlarından oluşmaktadır.

1. **Gereksinim (Requirements) :**

Müşteri ve donanım gereksinimleri çıkarılır, proje planı oluşturulur ve fizibilite (yapılabilirlik araştırması) çalışması yapılır.

1. **Analiz(Analysis)- Çözümleme :**

Sistem gereksinimleri ve işlevleri ayrıntılı olarak incelenir. Hali hazırda var olan işler incelenir ve temel sorunlar ortaya çıkartılır.

1. **Tasarım ( Design) :**

Belirlenen gereksinimler için yanıt verecek yazılım sistemlerinin temel yapısı bu aşamada oluşturulur.

-***Üst seviye ve mimari tasarım:*** Önerilen sistemin yapısı anlatılmaktadır. (Modüller, akış şemaları..)

-***Detaylı tasarım:*** Yazılımı içeren bileşenler ve bunların ayrıntıları. (Veri yapıları, ekran tasarımları..)

1. **Gerçekleştirme (Implementation) :**

Bu aşamada modüller kodlanır, birleştirilir, test edilir ve kurulum çalışmaları yapılır.

1. **Bakım (Maintenance) :**

Bu aşamada hatalar giderilir ve yeni gereksinimler eklenir. (Teslim edildikten sonra)

Yazılım yaşam döngüsünün temel adımları çekirdek süreçleri (Core Processes) olarak da adlandırılmaktadır. Bu süreçlerin geliştirilmesi amacıyla: Belirtim(specification) Yöntemleri, Süreç(process) Modelleri, Süreç Akışı İçin Kullanılan Belirtim Yöntemleri, Süreç Tanımlama Yöntemleri, Veri Tanımlama Yöntemleri bulunmaktadır. Örneğin yazılım yaşam döngüsü süreçlerinde geliştirme aşamasının hangi düzen ve sırayla, nasıl uygulanacağı süreç modeli kapsamında incelenir. Veri akış şemaları, süreç akışı için kullanılan belirtim yöntemleri arasında yer alır. [4]

**YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ**

Yazılım üretiminin genel yapısı ve yapılma düzenine ilişkin rehber görevi görmektedirler. Süreçlere ilişkin ayrıntılarla ve süreçler arası ilişkilerle ilgilenmezler.

**Gelişigüzel Model**

metin, dizüstü, iç mekan, bilgisayar içeren bir resim

Açıklama otomatik olarak oluşturulduBu yöntemi bir model olarak adlandırabilmek doğru değildir. Çünkü gelişigüzel bir model, yöntem bulunamaz. Fakat 1960’lı yıllarda uygulanan bu yöntemi genellikle basit programlama içeren ve tek kişinin üretim yaptığı projelerde kullanılmıştır. Kişiye bağlı yazılım geliştirme şeklinde yapılır bu yüzden izlenebilirliği, bakımı ve yapılabilirliği çok zordur. Geliştiren kişiye bağlı olmasına rağmen belirli bir süre sonra o kişi dahi sistemi anlayamaz ve geliştirme güçlüğü yaşar.

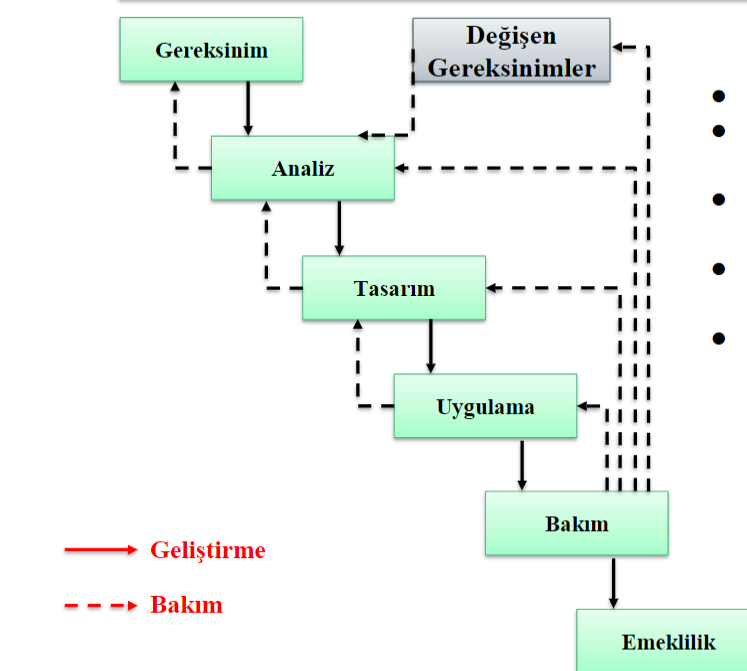
1. **Barok Modeli**

Yazılım yaşam döngüsü temel adımlarının doğrusal bir şekilde alınıp geliştirildiği bir modeldir. 1970’li yıllarda ortaya çıkmıştır. Aşamalar arasında nasıl gerekli geri dönüşlerin yapılacağı tanımlı değildir. Bu modelde, belgeleme(dokümantasyon) günümüzdeki modellerden farklı olarak ayrı bir süreçte ele alınmaktadır. Dokümantasyonun, yazılım geliştirme ve test faaliyetleri tanımlandıktan sonra yapılmasını öngörmektedir. Gerçekleştirim aşamasına ağırlık veren bir modeldir, günümüzde uygulanan bir model değildir.

İnceleme, analiz, tasarım, kodlama, modül testleri, alt sistem testleri, sistem testi, belgeleme, kurulum adımlarından oluşur.

1. **Şelale (Çağlayan/Geleneksel Yazılım Geliştirme) Modeli**

Geçmiş yıllarda en popüler yazılım geliştirme modeli olmuştur. En eski, tanınmış ve temel modeldir. Geleneksel yazılım geliştirme model olarak da bilinmektedir. Bu modelde, yazılım yaşam döngüsü temel adımları baştan sona kadar en az bir kez izlenerek gerçekleştirilir.

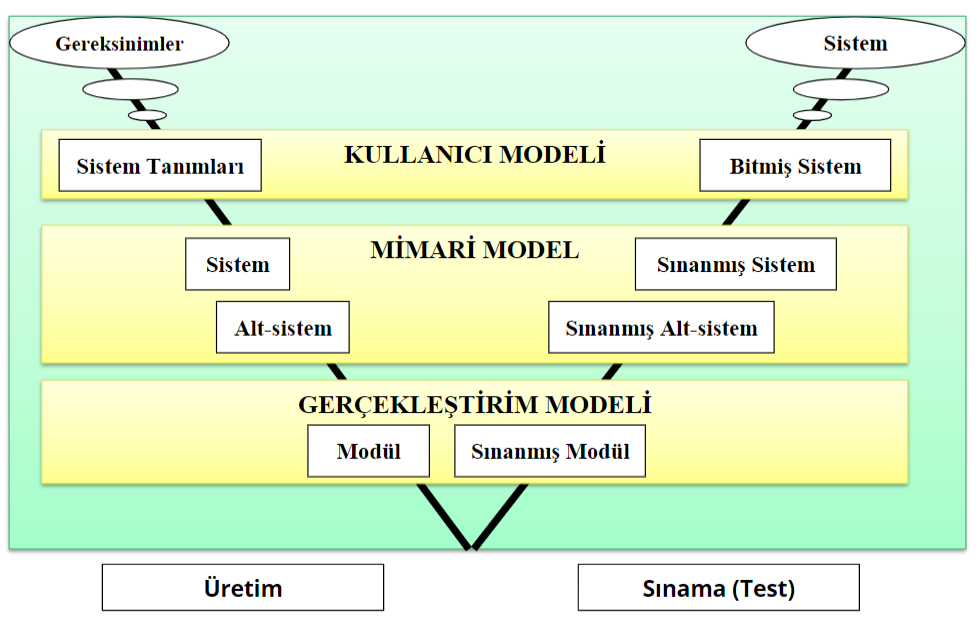
****Çok iyi tanımlanmış ve üretimi az zaman gerektiren projeler için uygun bir modeldir ve günümüzde kullanımı giderek azalmıştır. İşler aşamalar halinde yapılır. Bir aşama bitmeden diğer aşamaya geçilememektedir.



Her aşamada dokümantasyon yazılması gerekmektedir. Her şeyin dokümantasyonu olması gerekir. Eğer bir aşamada dokümantasyon ve test olmadıysa o aşamanın tamamlanmış olduğu kabul edilmez. Bu model, Barok modelinden farklı olarak dokümantasyonu ayrı bir süreç değil de üretimin doğal bir parçası olarak ele almaktadır. Aşamalar arasındaki geri dönüşlerin nasıl olacağı da tanımlıdır.

Bu modeli kullanırken aşamalar arasında geri dönüşler yapılabiliyor olsa da analiz aşamasında olabildiğince tüm detayların tasarıma yansıtılabilmesi adına müşteri ve sistemin gereksinimleri en ince ayrıntısına kadar belirlenmelidir ve tasarım aşaması da yazılımın tüm gereksinimlerini karşılayacak şekilde olmalıdır. Bu sebeple bu modelde analiz ve tasarım aşaması çok uzun bir zaman dilimine sahiptir. Ancak bu kadar uğraşmaya, detaylı çalışmaya ve vakit harcamaya rağmen projedeki gereksinimlerde değişmeler olması kaçınılmazdır. Üstelik kodlama ve test aşamalarında olabilecek değişimlerin sisteme/yazılıma yansıtılması da çok yüksek maliyete sahip olacaktır. Çünkü gerçek hayatta projeler genellikle yineleme gerektirir. Yazılımın kullanıcıya ulaşması uzun bir zaman alır. Gereksinimler net bir şekilde genellikle ifade edilemiyor olduğu için düzeltme, eksik giderme maliyeti oldukça yüksek olacaktır.

Kullanıcı bu modelde sürecin içinde yer almaz ve bu durumda da yazılım tamamlandıktan sonra geri dönüş sağlanacağı için yine eksik, hata ve düzeltmeler için oldukça yüksek maliyetler ortaya çıkacaktır. Barok modeline göre geri dönüşler iyi tanımlanmıştır. Yazılım tanımlamada belirsizlik yok veya yok denecek kadar az ise yazılımın üretimi çok zaman almayacak ise uygun bir modeldir.

1. **V Süreç Modeli**

Sol taraf üretim, sağ taraf ise sınama(test) işlemleridir. Temel çıktıları: Kullanıcı Modeli, Mimari Model ve Gerçekleştirim Modelidir.

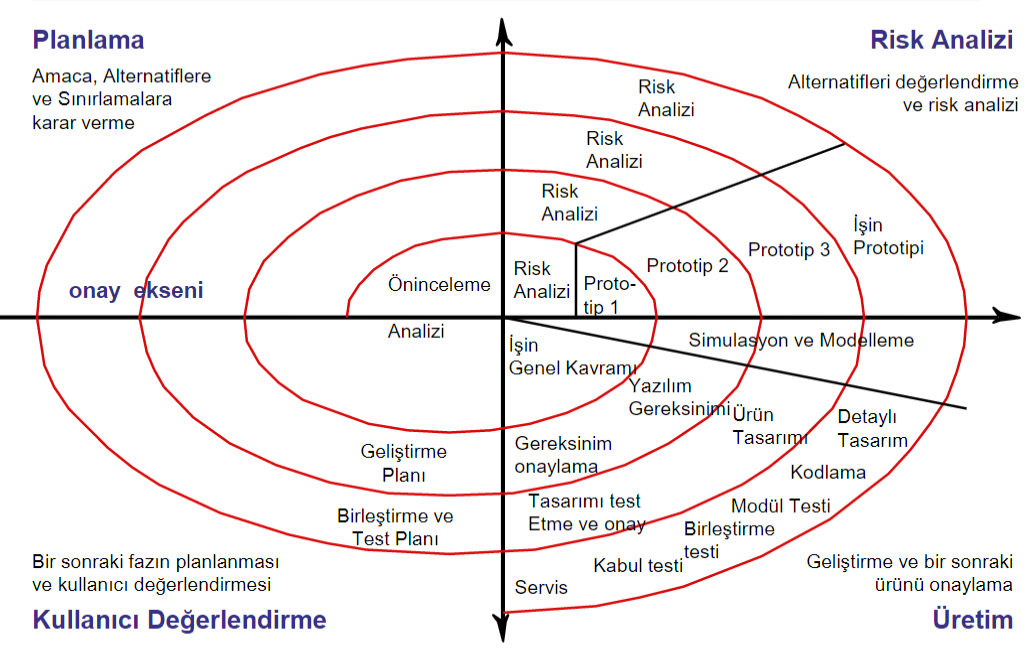
Kullanıcı Modeli : Gerçekleştirme sürecinin kullanıcı ile olan ilişkilerini tanımlamaktadır ve sistemin nasıl kabul edileceğine dair sınama belirtimleri ve planları ortaya çıkarılır.

Mimari Model : Sistem tasarımı ve oluşacak alt sistem ile tüm sistemin sınanması işlemlerine ilişkin işlevlerdir.

Gerçekleştirim Modeli: Yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlardır.

Belirsizliklerin az, iş tanımları belirgin olduğu BT(Bilgi Teknolojileri) projeler için uygun bir modeldir. Bu modelde, kullanıcının projeye olan katkısını artırmaktadır.

1. **Spiral(Helezonik) Model**

****Bu model; planlama, risk analizi, üretim ve kullanıcı değerlendirmesi temel aşamalarından oluşur.

Bu model; planlama, risk analizi, üretim ve kullanıcı değerlendirmesi temel aşamalarından oluşur.

Planlama: Üretilecek ara ürün için plan oluşturma, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirmenin yapıldığı adımdır.

Risk Analizi : Risk seçeneklerinin araştırılıp risklerin belirlendiği aşamadır.

Üretim : Ara ürünün üretildiği aşamadır.

Kullanıcı Değerlendirmesi : Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmeleri içerir.

Bu modelde risk analizi ön plana çıkmaktadır. Her döngü bir fazı ifade eder yani doğrudan gereksinim, analiz vs gibi fazlar yoktur. Yinelemeli artımsal bir yaklaşıma ve protoip yaklaşımına sahiptir.

Bu modelin birçok avantajı vardır. Üretim süreci boyunca ara ürün üretmesi ve bu ara ürünün kullanıcı sınamasına imkan vermesi ayrıca yazılımı kullanacak kişilerin erkenden sürece katılması ile istenilmeyebilecek durumlar erkenden engellenmiş olur.

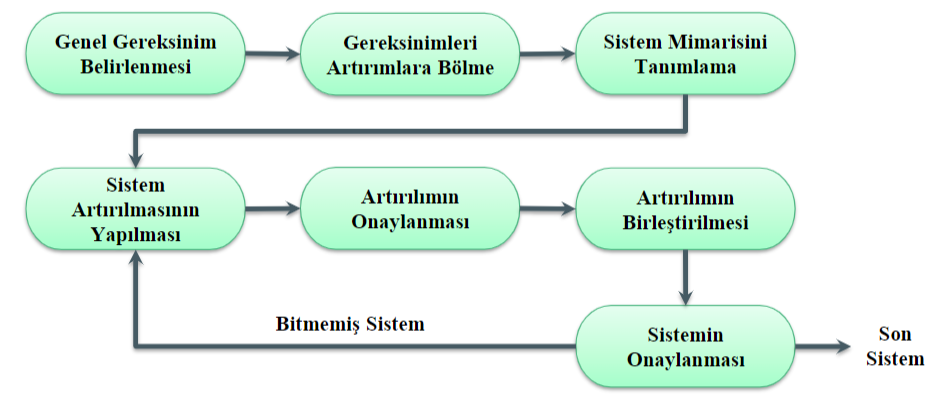
Bu modelde proje boyunca yöneticiler çalışan yazılımla karşılaştıkları için daha kolay izleme ve hak ediş planlamasını kolaylaştırır. Ayrıca yazılımın kodlanması ve sınanması da daha erken başlar.

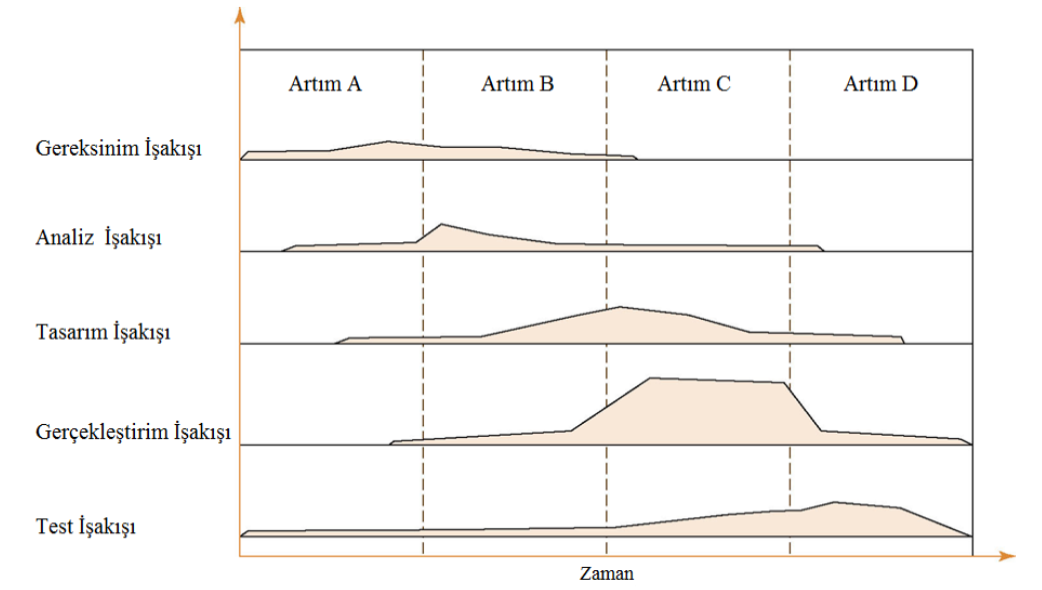
1. **Artımsal Geliştirme Süreç Modeli**

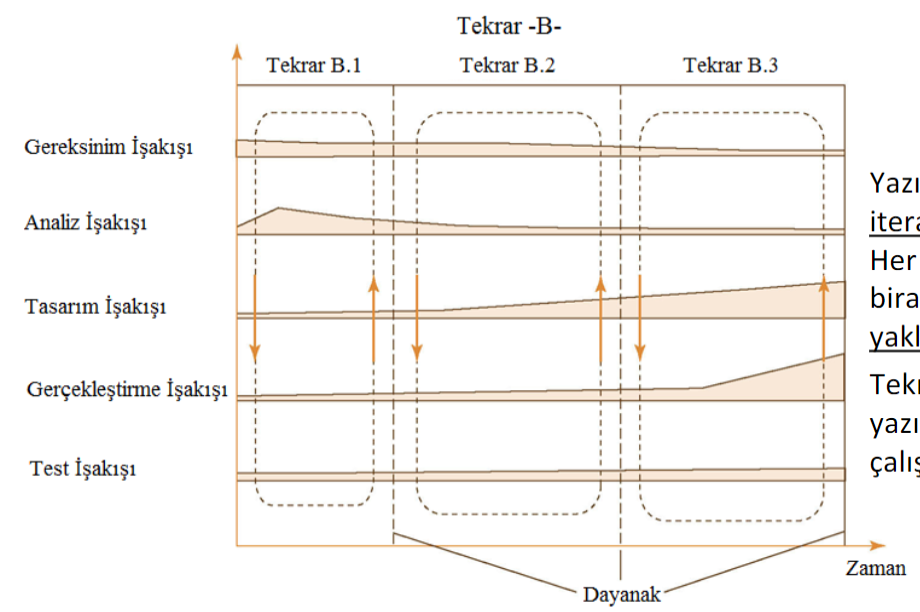
Bu modelde sistem tek seferde teslim edilmez. Geliştirme ve teslim parçalara bölünmektedir. Her teslim ise beklenen işlevselliğin bir parçasını karşılamaktadır.Kullanıcı gereksinimleri önceliklendirilir ve öncelikli gereksinimler ise erken teslimlere dahil edilmektedir. Gereksinimler önemleri, birbirine bağımlılıklarına göre sıralanır ve her yinelemede bunların bir kısmı tamamlanır.Bir parçanın geliştirilmesi başladığında ise gereksinimler dondurulur. Olası değişiklikler sonraki teslimlerde ele alınmaktadır.

Üretilen her yazılım sürümü birbirini kapsayacak ve giderek artan sayıda işlev içerecek şekilde geliştirilmektedir. Uzun zaman alabilecek ve sistemin eksik işlevsellikle çalışabileceği türden projelerde bu model kullanılabilir. Bir taraftan kullanım bir taraftan üretim yapılır. (Dönem projesi ödevi örnektir. Öğrenci geliştirmeye devam ederken, danışman gelişimi izlemektedir.)

Temel süreç şekilde verilmiştir.



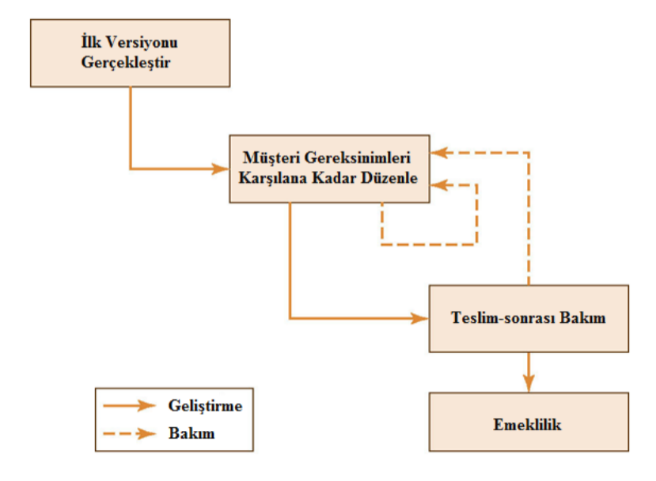
****

****



Yazılım geliştirme iteratif bir süreçtir ve her süreçte sonuca biraz daha yaklaşılır. Her teslimle müşteriye görünen bir değer döndüğü için sistemin işlevselliği erken aşamalarda ortaya çıkmaktadır. Erken teslimler sonrakiler için gereksinimleri çıkarmada prototip vazifesi görür. Projenin tamamen batması riski azaltılır. Öncelikli gereksinimleri karşılayan sistem işlevleri daha çok test edilmektedir.

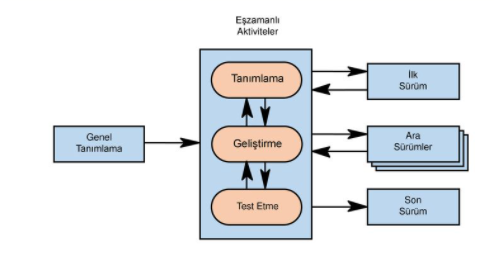
1. **Kodla ve Düzelt Yazılım Yaşam Döngü Modeli**

Yüzlerce satırdan oluşan oluşan programlar için kullanılabilir. İlk aşamada yazılımın ilk sürümü geliştirilir. Direkt olarak yazılım ürünü gerçekleştirilir. Sistem en son istenilen şekle gelene kadar devamlı olarak geliştirilir.

Bakım safhası vardır ancak çok zor bakım yapılmaktadır. Çünkü sisteme ait dokümantasyon bulunmamaktadır. Emeklilik safhası vardır. Bu model yazılım geliştirmenin en kolay yoludur ama aynı zamanda en pahalısıdır. Çünkü kodlamadan sonra sürekli değişiklik yapmak gerekiyor bu da maliyet olarak çok yüksek bir sayıyı ortaya çıkartır. Ayrıca pahalı olmasının yanı sıra dokümantasyon, şartname olmadığı için ürünün bakımını yapmak çok zordur. Yazılım geliştirmenin kolay olduğu , küçük( ya da tecrübesiz) firmalarda yazılım projelerinde kullanılır.

1. **Evrimsel Geliştirme**

İlk tam ölçekli modeldir. Daha çok banka uygulamaları gibi coğrafik olarak geniş alana yayılan, çok birimli organizasyonlar için önerilmektedir. Bu modelde modelin başarısı ilk evrimin başarısına bağlıdır. Her aşamada, üretilen ürünler üretildikleri alan için tam fonksiyonelliği/işlevselliği içerir. Pilot kullan, test et, güncelle ve diğer birimlere taşı şeklinde bir yol izlemektedir.

 Zayıf yanı, değişiklik denetimine sahip olmamasıdır. Modelin yapılandırılması, kalite, sürüm ve değişiklik yönetimi problemli olabilmektedir.

1. **Prototipleme**

Bu modelin temel amacı, ilgili alt modellerde ortaya çıkabilecek belirsizlikleri azaltmaktır. Prototip çalışması araştırma türünden bir çalışma olabilir veya sonradan atılacak bir yazılım parçası olabilir. Belirsizliği tanımlama, çözümleri tanımlama, prototip çalışması yapma ve belirsizliğin sonucunu elde etme adımlarından oluşur.

metin, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu modelin temel zayıf yönü , kaynak maliyetlerinin kestirilmesi zayıftır. Çünkü bir prototip çalışmasının ne kadar süreceği , ne kadar iş yükü getireceği kolayca kestirilemez ve bu sebeple yönetimi zorlaştırmaktadır.

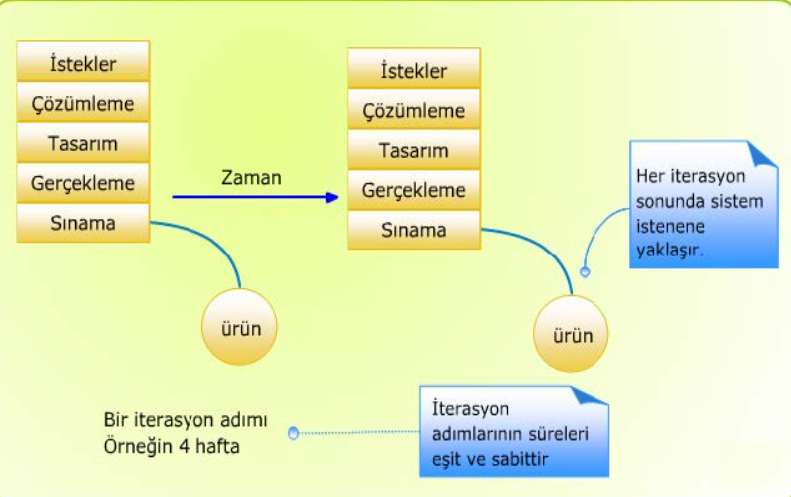
1. **Formal Sistem Geliştirme**

Bu model, matematiksel belirtinin farklı gösterim şekilleriyle çalıştırılabilecek bir programa dönüştürülmesine dayanır. Kullanıcı sistemi kullanmaya başladığı zaman karşısına çıkan belirtim hatalarını minimize eder. Çok zaman alan ve pahalı bir yöntem olduğu için bu model felsefesi pahalı olan hata ayıklama işlemini engellemek için kodu yazarken ilk başta doğru yazmak ve test aşamasında ise doğruluğunu sağlamak üzerinedir. Bu model hatasız yazılım imkanı sunmaktadır. [5]

1. **Yeniden Kullanıma Yönelik Geliştirme**

Organizasyon tarafından daha önce hazırlanmış veya dışarıdan alınmış bir yazılımın kullanılması ile geliştirme yapılması son yıllarda oldukça popüler hale gelmiştir. Organizasyonların olgunlukları arttıkça bu tür uygulamalar yapabilmek adına alt yapı kurulmuş olmaktadır. Kaynak kontrolü, maliyet denetimi, ve kısa sürede yazılım geliştirme imkanı sağlar. Basit ve anlaşılırdır. Daha önce oluşturulmuş sınıflar tekrar kullanılabilmektedir. Olumsuz yönleri ise gereksinimlerinin anlaşılmasının zorluğu, uzmanlık gerektirmesi, pahalı olması ve başarı garantisinin olmamasıdır. [6]

1. **Birleşik Süreç(Unified Process)**

Nesneye dayalı yazılım geliştirmek için olan yöntemlerin deneyimler sonucu kabul gören en iyi özellikleri bir araya getirilerek bir yazılım geliştirme süreci oluşturulmuştur. Yinelemeli, arttırmalı, evrimsel ve de aynı zamanda risk güdümlüdür.

Başlangıç(vizyon kararı, fizibilite çalışması, tamam ya da devam kararının verilmesi), Ayrıntılandırma (daha gerçekçi çözümleme, çekirdek yapı ve yüksek riskli kısımların yinelemeli oluşturulması), Tamamlama( daha az riskli ve düşük ölçekli kısımların yinelemeli olarak gerçekleşmesi) , Yayım (beta testleri , piyasaya sürme çalışmaları) aşamalarından oluşmaktadır. Değişen isteklere uygunluğu, erken geri besleme, risklerin erken giderilmesini sağlaması, erken ürün elde etme ve takım moralinin yükselmesi konularında avantaj sağlar ancak risk yönetimi zayıflığı, dokümantasyon yükünün ağır ve pahalı olabilmesi , karmaşıklığı bakımından dezavantajlıdır. [7]

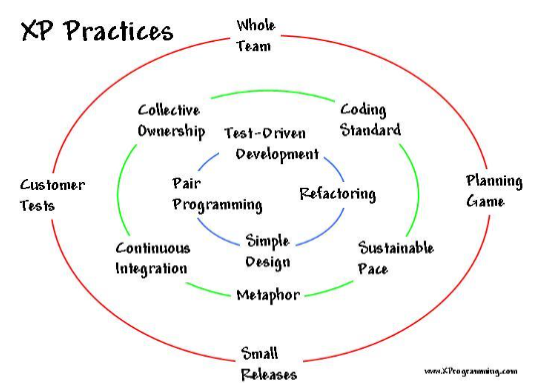
1. **Araştırma Tabanlı Model**

Bu model yap-at ilk örnek olarak da bilinmektedir. Araştırma ortamları tümüyle belirsizlik üzerine çalışan koşullardan oluşur. Bu modelde yapılan işlerden elde edilecek neticeler belirgin değildir.

1. **Çevik Modeller(Agile models: XP,SCRUM)**

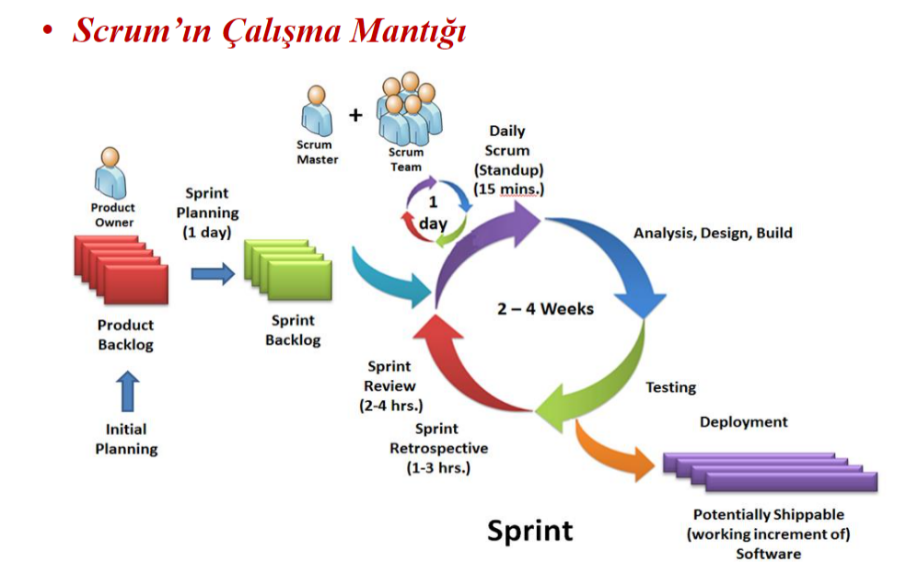
Bu modeller yazılım projelerinde başarı oranlarını artırmak, yazılım sürümlerinin zamanlama sorunlarını çözebilmek, değişik isteklere hızlı yanıt verebilmek, yazılım hatalarının erken safhada fark edilmesini sağlamak ve ekip içindeki iletişimi artırma amacıyla ortaya çıkmıştır. Bu metodlar hızlı ve ucuz çözüm sağlamaktadır. [8]

***-Extreme Programming (XP):***

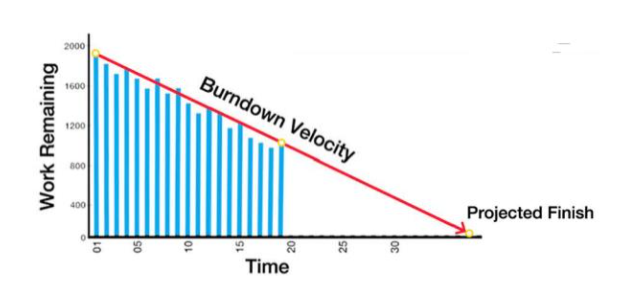
 Kent Beck ve arkadaşları 1996 yılında kurmuştur. 4 temel maddesi bulunur. Bunlar: basitlik, cesaret, geri dönüş ve iletişimdir. Basitlik; yazılan kodun sade , anlaşılır ve karmaşıklıktan uzak olması gerektiğini anlatır. Uzun dokümantasyondan uzak durulur. Cesaret ise projelerin üstünde korkmadan ilerlenmeyi ve kodun gerekirse silinip yeniden atılması eski kodun çöpe atılmasını ifade eder. Geri dönüş, oluşabilecek hataları azaltabilir veya ortadan kaldırabilir. Müşteri ve yazılım ekibi iletişim halindedir. İletişim ise projelerin en önemli sorunlarından birisidir XP bunu aşmaya çalışır. Ekip içi iletişime önem verir ve ekip içi iletişimin artırılması üzerine çalışır.

-***SCRUM***

Büyük projelerin parçalara bölünmesi ve her birisine “splint” adı verilmesi ve de bu splintlerin teker teker geliştirilmesi yapısına dayanır. SCRUM’da ekip içindeki iletişim oldukça önemlidir ve her gün “SCRUM meetings” adı verilen toplantılar düzenlenmektedir. SCRUM’da üç temel kavram vardır. Bunlar: Roller, Toplantılar ve Bileşenler/Araçlardır.

 Roller; ürünün sahibi, SCRUM yönetici ve SCRUM takımından oluşmaktadır ve ürünün sahibi proje için oldukça önemlidir hatta projenin beyni adı verilebilir. SCRUM yönetici ve takımı SCRUM etiklerine göre kontrolü sağlar. Takım 5-9 kişiden oluşan ve sürekli iletişim halinde olan bir yazılım geliştirme ekibidir.

Toplantılar; SCRUM için olmazsa olmazdır. Her gün toplantılar yapılır ve günlük olarak her gün yazılım geliştiricilerinin dün ne yaptıkları, karşılaştıkları sorunları ve bugün neler yapacaklarına dair bir konuşma geçmektedir. Her splint içinde bir de gözden geçirme toplantısı düzenlenir.

 Bileşenler/Araçlar: Ürün Gereksinim Dokümanı oluşturulur ve bu dokümanda proje boyunca neler yapılması gerekiyorsa basitçe yazılır. Splint Dokümanı oluşturulur. Bu doküman, ürün gereksinim dokümanına takiben oluşturulur ve amacı her splintin ona uygun bir biçimde ayarlanmasıdır ve bu dokümanı sadece takımdakiler değiştirebilir. Splint Kalan Zaman Grafiği, yapılan işin ne seviyede olduğu ve aslında planlanan zaman göre nerede olduğunun belirlenmesi amacıyla hazırlanır.

**SCRUM Günümüzde Neden Popüler?**

SCRUM’ın günümüzde en sık kullanılan yazılım geliştirme yöntemi olması ve aslında sadece yazılım geliştirmek için değil de birçok sistemin, projenin geliştirilmesinde de kullanılmasının bazı nedenleri vardır.

SCRUM; zamandan ve paradan büyük ölçüde tasarruf edilmesini sağlar, yüksek teknolojilere ve de son gelişmelere kolayca uyum sağlayabilir. Karmaşık görünen ve gereksinimleri tam belirlenemeyen projeler için idealdir. Ekip içi iletişim yüksektir ve bu da hataların erken fark edilmesi ve de daha kolay/hızlı düzeltilmesini sağlar. Kullanıcıdan sürekli geri bildirim alması ve bununla beraber sorunların azalışı, diğer yazılım metodolojileri gibi yinelemeli olması , değişen gereksinimlere hızlı tepki vermesi , böl ve fethet yaklaşımı içinde olması birçok avantaj sağlar ve bu sebeplerle SCRUM günümüzde sıkça kullanılır. Etrafımızda gördüğümüz birçok büyük marka SCRUM’ı kullanmaktadır.

**Yazılım Geliştirme Modellerinin Karşılaştırılması**

Maliyet konusunda Çevik Model ve Birleşik Süreç Modeli pahalı iken Artımlı Süreç daha uygun fiyatlı bir maliyet sağlar. Çevik Model, Artımlı Süreç ve Birleşik Modelden farklı olarak maliyet kontrolü yapmaya olanak sağlar. Zaman açısından Çevik Modeller, Birleşik Süreç ve Artımlı Sürece oranla oldukça tasarruf sağlar.

Spiral Model ve Formal Sisteme oranla Yeniden Kullanıma Yönelik ve Prototipleme modelleri daha düşük maliyete sahiptir. Yeniden Kullanıma Yönelik modelin başarı garantisi daha yüksektir. Yeniden Kullanıma Yönelik modelin bakımı diğerlerine oranla daha zordur.

V Model’in risk duyarlılığı yüksek değildir ancak Evrimsel Geliştirme, Çağlayan Model ve Kodla-Düzeltin risk duyarlılığı yüksektir. Değişiklik yapabilmek Kodla-Düzelt modelinde daha kolaydır.

Gelişigüzel modelde kişiye bağlı yazılım geliştirme yapıldığı için bakımı kodla düzelt, çağlayan model ve V modele göre oldukça zordur. Kodla-Düzelt, Çağlayan Model ,V Model, Evrimsel Geliştirme modellerinin zaman konusunda uzun bir dilimi alması sebebiyle Çevik Yazılımlar bu konuda oldukça tasarruf sağlar. [9]

Gelişigüzel ve Barok Modelleri artık gündemde değildir çünkü yinelemeli ve dokümantasyonu işin içerisinde barındırmamaları doğru değildir. Çağlayan modelin kullanılması, uygulanması basitti ve iş bölümü/planı detaylıdır. Eskiden en popüler modeldi artık kullanıcı iletişimi önemsendiği için ve büyük projelerde yetersiz kalıp değişimlere kapalı olması sorunları yüzünden kullanımı azalmıştır.

V model ise Çağlayan Model’in gelişmiş hali gibidir. Kullanımı kolay, onaylama, planlama erken aşamada bunlar avantajlı yönleridir fakat fazlar arası tekrar imkanı yoktur ve risk çözümleme basamakları olmaması gibi sorunlara sahiptir.

Spiral Model’de kullanıcı ara ürünleri gördüğü için sistemin içerisinde sayılabilir. Risk analizi çok önemlidir ve büyük projeler için idealdir. Hataların erken giderilmesi, planlamanın büyük bir parçayı oluşturması avantajlı fakat karmaşıklık, uzun sürmesi ve dokümantasyonun fazla olması gibi sebepler daha çok büyük projeler için uygun olduğunu gösterir.

Artımsal Geliştirme ise her teslimde müşteri geri bildirimi alınması sebebiyle işlevsellik erken safhada ortaya çıkmakta ve güvenli bir ortam oluşturmaktadır. Özellikler daha fazla test edilmektedir çünkü gereksinimler müşteri ile belirlenmektedir. Fakat her artımı tanımlamak için sistemin detaylı tanımlanması, deneyim gerekliliği unutulmamalıdır. Artımlar kendi içlerinde tekrarlanamaz.

Kodla-Düzelt Modeli küçük programlar için uygun fakat bakımı çok zor ve emeklilik safhası var. Büyük projeler için uygun olmama sebebi bakım masraflarının çok yüksek olması ve bireysel geliştirme ile sadece en son hali/istenilen hale kadar sadece kodlama yapılıp teslim edilmesi ve dokümantasyonun olmaması sebebiyle uygun değildir.

Çevik modeller müşteriyle olan iletişim ve ekip içi iletişimin yüksek olduğu , hataların az , alışmanın kolay, değişime ayak uydurabilen ve esnek olan , takım çalışmasına dayanan popüler bir modeldir. Dokümantasyonun detaylı olmaması , ihtiyaçların sürekli değişkenliği sebebiyle çalışma saatleri uzun olabilir.

Evrimsel Geliştirme Metodu’nda kullanıcılar gereksinimleri daha iyi anlar , hatalar azalır fakat sürecin görünürlüğü azdır ve bakım zordur.

**HANGİ PROJEDE HANGİ MODELİ KULLANMALIYIZ?**

Basit programlama içeren ve tek kişinin üretebileceği projelerde *Gelişigüzel Model* kullanılabilir ama bakımı oldukça zordur.

Çok iyi tanımlanmış ve üretimi az zaman gerektiren projelerde *Çağlayan Model* kullanılabilir ama günümüzde kullanımı oldukça azalmıştır. Çağlayan Model’de yazılım tanımlamada belirsizlik neredeyse yoktur ve yazılım üretimi çok zaman almayacaksa ise uygun bir modeldir. Küçük ve özellikleri iyi tanımlanmışsa *Çağlayan Modeli* kullanılabilir.

Büyük kitlelere ulaşacak bir projeler, daha çok banka uygulamaları gibi coğrafik olarak geniş alana yayılan çok birimli organizasyonlar için ise *Evrimsel Geliştirme Modeli* önerilir.

Belirsizliklerin az, iş tanımlarının belirlin olduğu BT projelerinde *V Model* de kullanılabilir. Bu modelde kullanıcının projeye katkısı artmıştır.

Yüzlerce satır koddan oluşuyorsa, zaman sorunu olmayan, bireysel projelerde *Kodla-Düzelt* kullanılabilir.

Büyük, maliyetli ve de uzun süren bir proje ise *Spiral veya Artımsal Geliştirme Modeli* uygundur.

Orta/küçük ölçekli ise, uzun sürmüyorsa *Çevik Modeller* kullanılabilir.

# **Kaynakça**

|  |  |
| --- | --- |
| [1] | «EMO,» [Çevrimiçi]. Available: https://www.emo.org.tr/ekler/0801491e6797bbd\_ek.pdf . |
| [2] | «Şadi Evren Şeker-Bilgisayar Kavramları,» [Çevrimiçi]. Available: http://bilgisayarkavramlari.sadievrenseker.com/2013/01/10/sdlc-yazilim-gelistirme-yasam-dairesi/. |
| [3] | *Doç. Dr. Deniz Kılınç, Bakırçay Üniversitesi Yazılım Mühendisliğine Giriş Dersi-BİL102- 2. ve 3. Hafta Sunumları.* |
| [4] | [Çevrimiçi]. Available: https://akademiksunum.com/index.jsp?modul=document&folder=d5d31bdd71631fa0332109b29cda3e2f97188f3c . |
| [5] | [Çevrimiçi]. Available: https://medium.com/@dilaraozturkmen/yazilim-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri%CC%87-597cd11a2a75. |
| [6] | [Çevrimiçi]. Available: https://medium.com/@brfn.kcr26/yazilim-geli%CC%87%C5%9Fti%CC%87rme-ve-s%C3%BCre%C3%A7-modelleri%CC%87-2131ea5f09b2#:~:text=8.Yeniden%20Kullan%C4%B1ma%20Y%C3%B6nelik%20Geli%C5%9Ftirme,i%C3%A7in%20alt%20yap%C4%B1%20kurulmu%C5%9F%20olmaktad%C4%B1r.. |
| [7] | [Çevrimiçi]. Available: http://w3.bilecik.edu.tr/bilgisayar/wp-content/uploads/sites/75/2019/02/ders-2.pdf. |
| [8] | [Çevrimiçi]. Available: https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742. |
| [9] | [Çevrimiçi]. Available: https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/. |