**metin içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**BİL 102 YAZILIM MÜHENDİSLİĞİ TEMELLERİ**

**Doç. Dr. Deniz Kılınç**

**Mahmut Ersoy**

**210601059**

**Yazılım Yaşam Döngü Modelleri**

**Yazılım Nedir?**

Yazılım, bilgisayarı çalıştırmak ve belli birtakım görevleri yerine getirmesi için kullanılan veriler, programlar ve komutlar bütünüdür. Yazılım günümüzde aklınıza gelebilecek tüm sektörlerde kullanılmaktadır.

**Yazılım Yaşam Döngüsü (Software Development Life Cycle, SDLC)**

Yazılım aslında yazılım mühendislerinin ürettiği bir üründür ve her üründe olduğu gibi yazılımın da bir yaşam süreci vardır. Bu yazılım yaşam döngüsü, belirli aşamalardan oluşur. Bu aşamaların herhangi birinde ilerlerken eğer önceki aşamalara geri dönmemiz gerekirse dönülebilir. Hata veya eksikler varsa bunlar düzeltilebilir ve tekrar ilerlemeye devam edilebilir. Bu aşamalar isminden de anlaşılacağı gibi bir döngü şeklinde ilerler. Bu aşamalar temelde şu şekildedir:

**1)Planlama:** Her işte olduğu gibi ilk önce planlama yapmamız gerekir. Bu planlama da yazılım için temel ihtiyaçlar, toplantı tarihleri ve projenin genel anlamda planlanması ile bu basamak tamamlanmış olur. Bu aşama da yazılım ekibi ve müşteri ortak kararlar alarak ileri ki aşamaların daha kolay geçmesini sağlar.

**2)Gereksinimlerin Belirlenmesi:** Müşterinin istekleri ve temel ihtiyaçları belirlenmeye çalışılır. Sonrasında müşteriye gereksinim önerileri verilir.

**3)Analiz:** Bu aşamada müşteri ve yazılım ekibinin en fazla birlikte karar vermesi gereken aşamadır. Müşterinin istekleri çoktur ama zaman ve imkân kısıtlı olduğu için müşteriden toplanılan gereksinimler değerlendirilir. Sonrasında müşteriden alınan gereksinimlerin hangilerinin istek, hangilerinin ihtiyaç olduğu kesin olarak belirlenmeye çalışılır. Bu aşamada müşteri ile yazılımcı arasındaki ilişki çok önemlidir. Bu ilişkinin iyi olması sonraki aşamalarda eksik gereksinimlerin ortaya çıkmasını ve bazı olası hataların meydana gelmesini engeller. Bu aşamadaki önemli noktalardan biri de projenin dokümantasyonunun yapılmasıdır.

**4)Tasarım**: Gereksinimlerin tam olarak belirlenmesi ve dokümantere edilmesinden sonra bu aşamaya geçilir. Bu aşama, gerçekleştirime ve test aşamalarının temeli olduğu için çok önemlidir. Tasarım öncelikle anlaşılabilir ve basit olmalıdır. Bu, yazılımcıya sonraki aşamalarda kolaylık sağlayacaktır. Tasarım aşamasında ara yüzler ve sistemin özellikleri belirlenir. Bu aşamada iki tür tasarım yapılır. Bunlar, mimari tasarım ve detaylı tasarımdır. Mimari tasarım, modüllere, akış şemalarına ve use caselere rastlanır. Detaylı tasarım ise ekran tasarımları, veri yapıları, sınıflar ve yazılımın programlama dili belirlenir.

**5)Gerçekleştirime:** Bu aşamada önceki aşamalarda belirlenenlere göre müşteriye sunulacak olan ürünün kodlaması yapılır. Kod “Clean Code” felsefesine uygun olacak biçimde kısa, net ve anlaşılır biçimde yazılır. Temiz kod yazmamız hem yazılımı hazırlayan bize hem de sonrasında yazılımı inceleyecek olan yazılımcılara kolaylık sağlayacaktır.

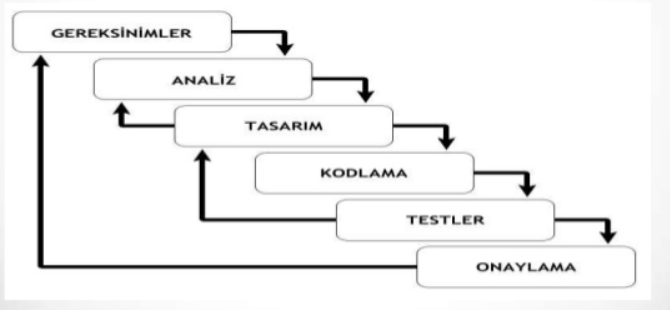
**6)Test:** Sisteme ait her modül için kodlar yazılır. Daha sonrasında her modül birbirinden bağımsız olacak şekilde testler yapılır. Oluşturulan yazılımın çalışabilmesi için tüm modüller birleştirilir. Modüllerin birleştirilmesi sırasında ortaya bazı hatalar çıkabilir. Bu gibi durumlar test edilir. Hem kodlama süresince hem de kodlama sonrasında yapılması gereken en önemli şeylerden biri testtir. Analiz aşamasından itibaren her aşamada testler yapılmalıdır. Hataların ortaya çıktığı aşama, teslim aşamasına ne kadar yakın olursa hataların düzeltilme maliyeti de düzeltilme süresi de o kadar artar. Bu yüzden erken test çok önemlidir. Yazılımcıya zaman, para ve prestij kazandırır. Gerçekleştirme aşamasında sistem artık hayata geçer. Bu aşamada artık gerçek veri kullanılarak sistem test edilir. Bu test kabul testidir ve müşteri tarafından yapılır.

**7)Teslim ve Bakım:** Yazılım oluşturulup müşterinin hizmetine sunulduğu aşamadır. Teslim ürünü olarak yalnızca yazılım ürünü yeterli değildir. Bunun yanında ayrıca kullanıcılar için kullanım kılavuzu da olmalıdır. Ürün teslim edildikten sonra zaman içinde hatalarının giderilmesi veya güncellenmesi gerekebilir. Bu sürekli devam edecek olan bir aşamadır. Bu noktada müşteri geri dönüşleri büyük önem taşımaktadır. Bu süreçte iki tarafın uzlaşması ve onayıyla yazılıma yeni özellikler eklenip çıkarılması da yapılabilir.

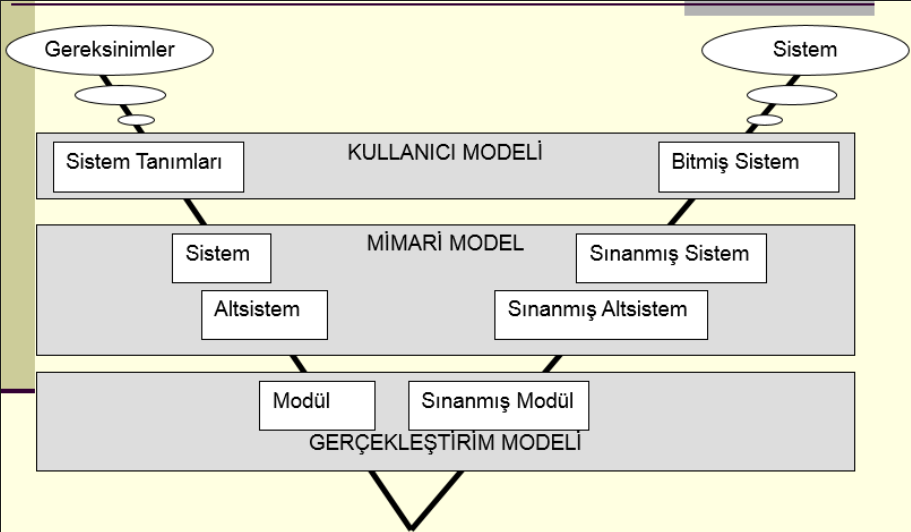
**Yaşam-Döngü Modelleri (Life-Cycle Models)**

**1)Gelişigüzel Model:** 1960’lı yıllarda kullanılan bu yöntem genellikle tek bir kişinin üretim yaptığı küçük ve basit projelerde kullanılır. Bu modelde herhangi bir sistem veya yöntem yoktur. Bu yüzden tam bir model olarak ele alınmaz. Kodlama tarzı ve süreç kişiden kişiye değişkenlik gösterir. Aynı zamanda dokümantasyonda yapılmaz. Bu sebeplerden dolayı yazılımın güncellenmesi ve bakımı çok zor hale gelir hatta bazen güncelleme ve bakım yapılamayabilir.

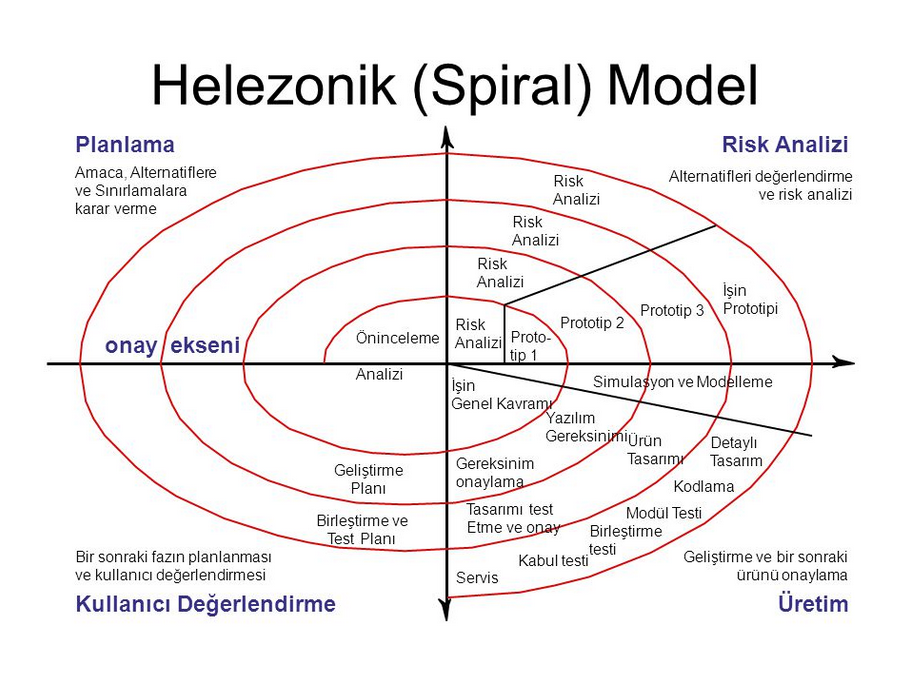
**2)Barok Model:** Bu model 1970’li yıllarda ortaya çıkmıştır.  Yazılım yaşam döngüsündeki aşamalar sırayla takip edilir. Herhangi bir aşamada hata veya eksik çıktığında geri dönüşlerin nasıl yapılacağı tanımlanmamıştır. Ayrıca bu modelde diğer modellerden ve günümüzden farklı olarak dokümantasyon ayrı bir süreç olarak değerlendirilir. Geliştirme ve test işlemleri bittikten sonra dokümantasyon yapılır. Bu yüzden süreç sırasında dokümantasyon açısından birçok veri kayba uğramakta ve dokümantasyon açısından büyük sıkıntılar açmaktadır. Saydığımız bu sebepler dolayı Barok model günümüzde geçerliliğini kaybetmiş ve kullanımdan çıkan bir model olmuştur.

**3)Çağlayan (Şelale) Yaşam Döngü Modeli (Waterfall):** Çağlayan yaşam döngü modeli geçmişte en fazla kullanılan modeldir. Aynı zamanda birçok modele öncülük etmiştir. Bu modelde yazılım yaşam döngüsündeki her aşamaya büyük önem ve özen gösterilir. Bir aşama tam anlamıyla bitmeden diğer bir aşamaya kesinlikle geçilmez. Aynı zaman da her aşamada dokümantasyon ve test yapılır. Bu modelin sorunlarına gelecek olursak, en önemli sorunu müşteriyi bu süreçlere hiç karıştırmamasıdır. Her aşamanın oldukça detaylı yapılması da süreci uzatmaktadır. Bu süre zarfında müşterinin istek ve ihtiyaçları büyük oranda değiştiği için yeniden yapım dolayısıyla hem süre tekrardan uzuyor hem de maliyet artmakta. Bu model küçük ve gereksinimleri çok iyi anlaşılmış olan projelerde işe yaramaktadır. Ancak daha karmaşık, uzun veya nesne yönelimli projeler için uygun değildir.

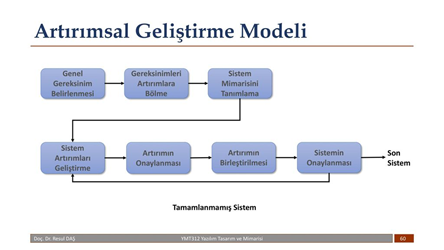
**4)V Süreç Modeli:** Bu modele şelale modelinin gelişmiş hali denilebilir. V süreç modelinin sol tarafı üretim, sağ tarafı test işlemlerini gösterir. Bu modelin aslında 3 temel çıktısı vardır. Bunlar; Kullanıcı Modeli, Mimari Model, Gerçekleştirm Modelidir. Kullanıcı Modelin de kullanıcının istekleri, proje için tanımları kullanılır ve planlar yapılır. Mimari Model de projenin tasarımı ve tüm sistemin test işlemleri yapılır. Gerçekleştirm Modeli ise yazılımın kodlarının yazılması ve test işlemleri yapılır. Bu modelin en büyük avantajı takibinin ve kullanımının kolay olmasıdır. Bu model belirsizliklerin az olduğu aynı zamanda iş tanımlarının net bir şekilde belirlendiği bilgi teknolojileri projeler için uygundur. Bu modelin en büyük sorunları aşamalarda tekrarların olmaması ve risk çözümleme olmamasıdır.



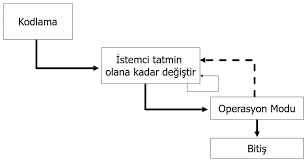
**5)Helezonik (spiral) Model:** Helezonik modeli diğer modellerden farklı kılan özellikleri risk analizinin ön planda olması ve prototip oluşturulmasıdır. Risk analizinin ön planda olması sayesinde hataları erken çözme olanağı olur. Prototip oluşturma her aşama da gerçekleştiği için kullanıcı da her aşamada yazılım projesinin bir parçasını görür ve bu da sorunların azalmasına yardımcı olur. 4 temel aşamadan oluşur. Planlama aşamasında her aşamada olan ara ürün için planlama ve amaç belirleme yapılır. Risk Analizinde ise riskler araştırılır, belirlenir ve çözülür. Üretim aşamasında ara ürün üretilir. Kullanıcı Değerlendirme aşamasında ise kullanıcıya sunulan ara ürünün kullanıcı tarafından sınandığı ve değerlendirildiği aşamadır. Bu modelin en büyük sorunları küçük ve düşük ölçekli projeler için maliyetli olmasıdır. Daha çok büyük ölçekteki projeler için kullanılmalıdır. Helezonik model karmaşıktır. Ara adımların çok fazla olması sebebiyle oldukça fazla dokümantasyon gerektirir.



**6)Artımsal Geliştirme Süreç Modeli:** Bu modelde proje parçalara bölünür ve kullanıcının ihtiyaç önceliğine göre bu parçalar sıralanarak sırasıyla birer ara ürün geliştirilmiş olur. Her ara üründe bir öncekinin üstüne geliştirilerek çıkartılır. Kısacası bu modelde bir yanda üretim kısmı devam ederken diğer tarafta ise kullanım kısmı devam eder. Bu model uzun zaman alacak ve ürünün eksik işlevsellik ile çalışabilecek yazılımlar için uygundur. Bu model sayesinde sistemin başarısız olma ihtimali azalır. Bu modelin en büyük sorunu ise her bir parçanın kendi içinde tekrar etmesine izin verilmediğinden dolayı bir ara ürünün bitip diğeri başlayana kadar değişiklik yapılamamasıdır.



**7)Kodla ve Düzelt Modeli:** Herhangi bir planlama yapılmaz. Çok küçük projelerde kullanılır. Program aşamaları hızlı geçilir. Diğer modellerden farklı olarak emeklilik safhası içerir. Bu modelin sorunlarına gelirsek; planlama yoktur, bitiş süresi net değildir, hataların bulunması zordur, kodları değiştirmek zor ve maliyetlidir.

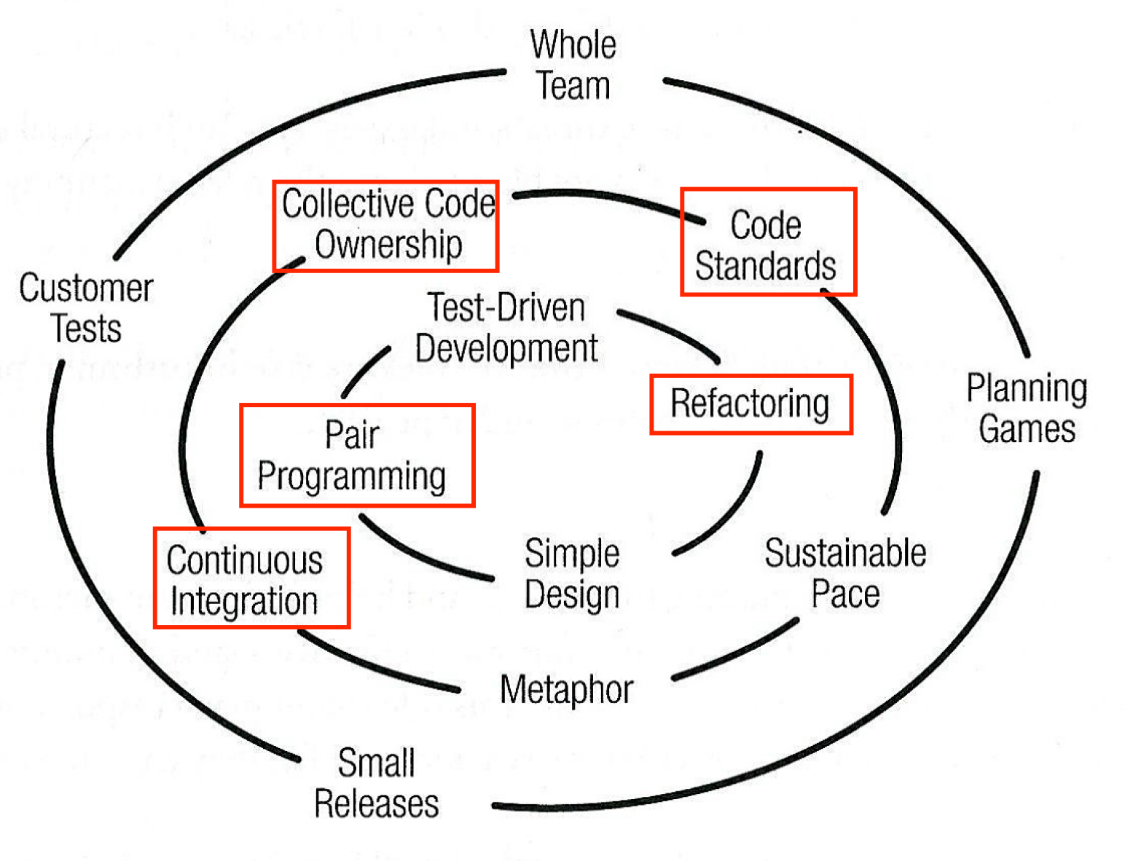


**8)Çevik Yazılım:** Yazılım ürünlerinin teslimat süresinin çok uzun olması, müşterilere zamanında ürünün teslim edilememesi, sistemdeki hataların büyük bir çoğunluğunun geç fark edilmesi, müşterinin ihtiyaç ve isteklerine kolay şekilde adapte olunamaması ve maliyetlerin artması gibi sorunlar ortaya çıktığından dolayı projelerde yüksek oranda başarı ve müşteri memnuniyeti elde edilememiştir. Tüm bu sorunlardan dolayı 1990’lı yılların sonuna doğru Çevik Yazılım metotları geliştirilmiştir. Bu metotlar ürünün en kısa zamanda müşteriye ulaşmasını sağlar. Dokümantasyona önem verilir. Eski modellerde sorun olan müşteri istek ve ihtiyaçlarına göre olan değişiklikleri kolay şekilde halletmeyi amaçlar.

**Scrum:** Bir Çevik Yazılım yöntemidir. Scrum, 1990’lı yıllarda Jeff Sutjerland ve Ken Schawaber tarafından bulunmuştur. Büyük yazılım projelerini küçük parçalara(sprint) böler ve her bir sprinti teker teker geliştirir. Scrum da ekip içi iletişim ve bilgi alışverişi çok önemli olduğu için her gün “Scrum Meetings” adı verilen toplantılar yapılır. Scrum da Roller, Toplantılar ve Araçlar olmak üzere 3 temel kavram vardır. Roller; ürün sahibi, scrum yöneticisi ve scrum ekibinden oluşur. Scrum yöneticisi ekibini scrum kurallarına göre kontrol eder. Scrum ekibi ise 5-9 kişiden oluşan yazılım geliştirme ekibidir. Ürün sahibi projenin beyni görevindedir. Toplantılar; Her gün scrum toplantıları yapılır. Bu toplantılarda ekip dün ne yaptık, karşılaştıkları sorunlar ve çözüm yöntemleri, bugün ne yapacakları hakkında konuşurlar. Araçlar; ürün gereksinim dokümanı ve sprint dokümanı oluşturulur. Sprint dokümanı her sprintin ona uygun hazırlanması için oluşturulur ve bu dokümanı sadece ekiptekiler değiştirebilir. Sprint Kalan Zaman Grafiği ise yapılan işin hangi seviyede ve planlanan zamana göre nerede olduğunu bilmek için hazırlanır. Günümüzde en fazla Scrum kullanılır. Scrum sadece yazılım geliştirmede değil birçok sistemin geliştirilmesinde de kullanılır. Çünkü; karmaşık ve ihtiyaçları tam belirlenmemiş projeler için ideal olması, ekip içi iletişimin yüksek ve buna bağlı olarak hataların erken fark edilmesi, kullanıcıdan dönütler alarak sorunların azaltılması, zaman ve paradan tasarruf ettirmesi, son gelişme ve teknolojilere uyum sağlaması sayılabilir.



**Extreme Programming (XP): Kent** Beck ve arkadaşları tarafından 1990’lı yılların sonuna doğru kurulmuştur. Extreme Programming’de müşteri her aşamada vardır. İletişim yüz yüzedir. Yazılımcılar ile müşteriler arasındaki ilişkininin çok iyi ve sağlıklı olması gerektiğini savunur. Asıl amacı sadelik ve basitliktir. Karmaşık projeler bu modele uygun değildir. Bu modelde geri bildirimler önemlidir çünkü bu sayede çıkabilecek hatalar erkenden tespit edilir ve çözülür. XP 4 temel maddeden oluşur. Bunlar; Basitlik, Cesaret, Geri Dönüş, İletişimdir. Basitlikte yazılan kod sade ve anlaşılır olmalıdır. Cesarette projenin üstüne korkmadan gidilmelidir hatta gerekirse kodu tamamen silip yeniden yazılması gerekir. Geri Dönüşte yapılan geri dönüşler ile yapılacak hatalar azaltılır. İletişim projelerdeki en büyük sorunlardan biri olduğu için XP bunu çözmek için ekip içi iletişime önem verir.



**KAYNAKÇA**

* Doç. Dr. Deniz Kılınç, Bakırçay Üniversitesi Yazılım Mühendisliği Temelleri Dersi 2. Ve 3. Hafta Sunumları
* <https://toptalent.co/agile-yazilim-gelistirme-alaninda-bir-metod-scrum-framework-nedir>
* <https://clarusway.com/what-is-software-development-life-cycle/>
* <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>
* <https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>
* <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://medium.com/@furkaneren_81038/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-203fe380b5e7>
* <https://mustafaserdarkonca.medium.com/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modellerini-a%C3%A7%C4%B1klamadan-%C3%B6nce-yaz%C4%B1l%C4%B1ma-ve-yaz%C4%B1l%C4%B1m%C4%B1n-nas%C4%B1l-bir-geli%C5%9Fim-ge%C3%A7irerek-2f724cd8c868>