****

**BİL 102**

**Yazılım Mühendisliği Temelleri**

**Doç. Dr. Deniz KILINÇ**

**Neslihan Kevser AKMAN**

**190601060**

**YAZILIM YAŞAM DÖNGÜ MODELLERİ**

**Yazılım Nedir?**

En yalın tanımıyla yazılım, belirli işlevleri yerine getirmesi için bilgisayarlara verilen komutların tümüdür. Yazılım; günümüzde sağlık sektöründen eğlenceye, ulaşımdan haberleşmeye, alışverişten savunma sistemlerine kadar hayatımızın her alanında vardır.

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

Bir yazılım ürünün üretilmeye başlandığı andan başlayıp kullanıldığı süreyi de içine alan ve bakım sürecini de kapsayan tüm aşamalara yazılım yaşam döngüsü denir. Yazılım yaşam döngüsü, belirli aşamalardan oluşur. Bu aşamaların herhangi birinde ilerlerken önceki aşamalara geri dönülebilir, hata veya eksikler varsa bunlar düzeltilebilir ve tekrar ilerlemeye devam edilebilir. Bu aşamalar bir döngü şeklinde ilerler. Aşamaların sayısı yazılıma ve kullanılan yaşam döngüsü modeline göre değişkenlik gösterebilir. Bu aşamalar temelde şu şekildedir:

**1)Gereksinimlerin toplanması (requirement phase):** Yazılım yaşam döngüsünün ilk aşamasıdır. Müşterinin istekleri ve temel ihtiyaçları belirlenmeye çalışılır. Bu aşamada projenin fizibilite çalışmaları yapılır. Tüm projenin planlaması ve maliyet hesabı da bu aşamada yapılır.

**2)Analiz aşaması (analysis phase):** Toplanılan gereksinimlerin değerlendirildiği aşamadır. Müşteriden alınan gereksinimlerin hangilerinin istek hangilerinin ihtiyaç olduğu kesin olarak belirlenmeye çalışılır. Burada “yazılımdan tam olarak ne bekliyoruz, yazılımın tam olarak ne yapmasını istiyoruz” sorularına cevap aranır. 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.

**3)Tasarım aşaması (design phase):** Gereksinimlerin tam olarak belirlenmesi ve dokümante edilmesinden sonra bu aşamaya geçilir. Bu aşama, gerçekleştirim ve test aşamalarının temeli olduğu için çok önemlidir. Tasarım öncelikle anlaşılabilir ve basit olmalıdır. Bu, yazılımcıya sonraki aşamalarda kolaylık sağlayacaktır. Tasarım aşamasında arayüzler ve sistemin özellikleri belirlenir. Bu aşamada iki tür tasarım yapılır. Bunlar, mimari tasarım (architectural design) ve detaylı tasarım (detailed design)’ dır. Mimari tasarım, yazılım modülleriyle ve sistemin genel yapısıyla ilgilenir. Detaylı tasarım ise yazılımın detaylarıyla, yazılımda kullanılacak algoritmalarla, veritabanıyla, etkileşim diyagramları ve sınıflarla ilgilenir. Burada en çok kullanılan yöntemlerden biri soyutlamadır. Soyutlama (abstraction), bir problemin çözümü için belirli bir nesnenin, olayın veya bir durumun bize gerekli olan kısmına yani bizim için en önemli olan yerine odaklanıp onun dışında kalan yerlerini veya özelliklerini görmezden gelmektir. Örneğin bir pencere nesne olarak ele alındığında pencerenin kulpu, çerçevesi, yapıldığı malzeme gibi detayları görmezden gelerek pencereyi bir ev mimarisi içinde düşünebiliz. Aksi halde diğer detaylara yoğunlaşan bir tasarımcı, oda düzeyinde bir görsel canlandırma yapamaz.

**4)Gerçekleştirim (kodlama ve test) (implementation phase):** Tasarım aşamasında belirli bir seviyeye gelindikten sonra gerçekleştirim aşamasına geçilir. Bu aşamada müşteriye sunulacak olan ürünün kodlaması ve kodlamadan sonra hatasız teslim edilebilmesi için testleri yapılır. Kodlama kısmında en önemli nokta kodun temiz, okunabilir ve anlaşılabilir olmasıdır. Bunun sağlanabilmesi için de yazılımcıların kodlama kalite standartları olmalıdır. Kodlama kalite standartlarına uymak hem yazılımı hazırlayan yazılımcıya hem de sonrasında yazılımı inceleyecek olan yazılımcılara kolaylık sağlayacaktır. Sisteme ait her modül için kodlar yazılır. Daha sonrasında her modül birbirinden bağımsız olacak şekilde testler yapılır (unit testing). Oluşturulan yazılımın çalışabilmesi için tüm modüller birleştirilir. Modüllerin birleştirilmesi sırasında ortaya bazı hatalar çıkabilir. Bu gibi durumlar test edilir (integration testing). Hem kodlama süresince hem de kodlama sonrasında yapılması gereken en önemli şeylerden biri testtir (early testing). 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. Gerçekleştirim aşamasında siste artık hayata geçer. Bu aşamada artık gerçek veri kullanılarak sistem test edilir. Bu test kabul testidir (acceptance testing) ve müşteri tarafından yapılır.

**5)Teslim ve bakım(post-delivery maintenance phase):** Bütün test aşamaları tamamlandıktan sonra ürünün, müşteriye teslim edilecek versiyonu hazırlanmış olur. Teslim aşaması gerçekleştirilir. Teslim ürünü olarak yalnızca yazılım ürünü yeterli değildir. Bunun yanında ayrıca kullanıcılar için kullanım kılavuzu da olmalıdır. Ürün sahaya sunulduktan sonra da bu aşama devam eder. Burası bakım aşamasıdır. Ürün teslim edildikten sonra zaman içinde hatalarının giderilmesi veya güncellenmesi gerekebilir. Bu sürekli devam edecek olan bir aşamadır. Teslimat sonrası bakımın iki çeşidi vardır. Bunlardan ilki “Düzeltici bakım (corrective maintenance)” dır. Teslimat sonrasında, yazılım ürününde karşılaşılan hataları düzeltmek amacıyla yapılır. İkincisi ise “Özelliklerin arttırılması (enhancement)” dır. Bu da mükemmelleştirici bakım (perfective maintenance) ve uyarlanabilir bakım (adaptive maintenance) olarak ikiye ayrılır. Mükemmelleştirici bakım, kullanıcının isteği üzerine yazılım ürününün özelliklerini ve faydalarını arttırmak için yapılan bakımdır. Uyarlanabilir bakım ise yazılım ürününün kullanım sahasında meydana gelen değişikliklere uyum sağlaması için yapılan bakımdır.

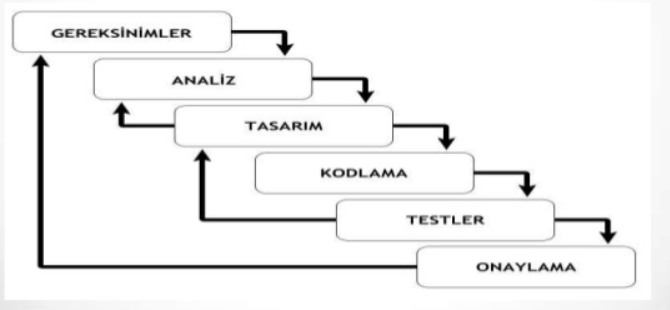
**Yazılım Yaşam Döngüsü Modelleri**

Yazılım yaşam döngüsünde bulunan aşamaların gerçekleştirilmesi sürecinde, hangi aşamaların hangi sırayla yerine getirilmesi gerektiğini ve bu aşamaların nasıl uygulanabileceğini gösteren bazı yazılım yaşam döngü modelleri vardır. Bu modeller sayesinde projeler yürütülürken karmaşıklık en aza indirgenir, müşterinin isteğine ve ihtiyaçlarına en yakın ürün ortaya çıkarılır, aşamaların belirli sıralarla takip edilmesiyle beraber hatalar en aza indirilerek hem maliyet düşürülür hem de prestij kazanılır.

**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 aslında model olarak adlandırmak doğru değildir. Bu modelde direkt olarak yazılım geliştirilmeye başlanır ve elde edilmek istenen ürün ortaya çıkana kadar yazılım geliştirilmeye devam edilir. Dokümantasyon olmamasından ve yazılımın anlaşılmasının zor olmasından kaynaklı olarak bu şekilde oluşturulan yazılım ürünlerinin bakımları çok zor yapılmaktadır veya yapılamamaktadır. Kodları sonradan değiştirilmeye elverişli yazılmadığından dolayı projeler esnek değildir ve kodların değiştirilmesi maliyetli olabilir. Yazılım geliştirmek için basit ve rahat ama masraflı bir yoldur. Bu tür projeler kontrollü değildir ve ürünün teslim tarihi net olarak belirlenemez.

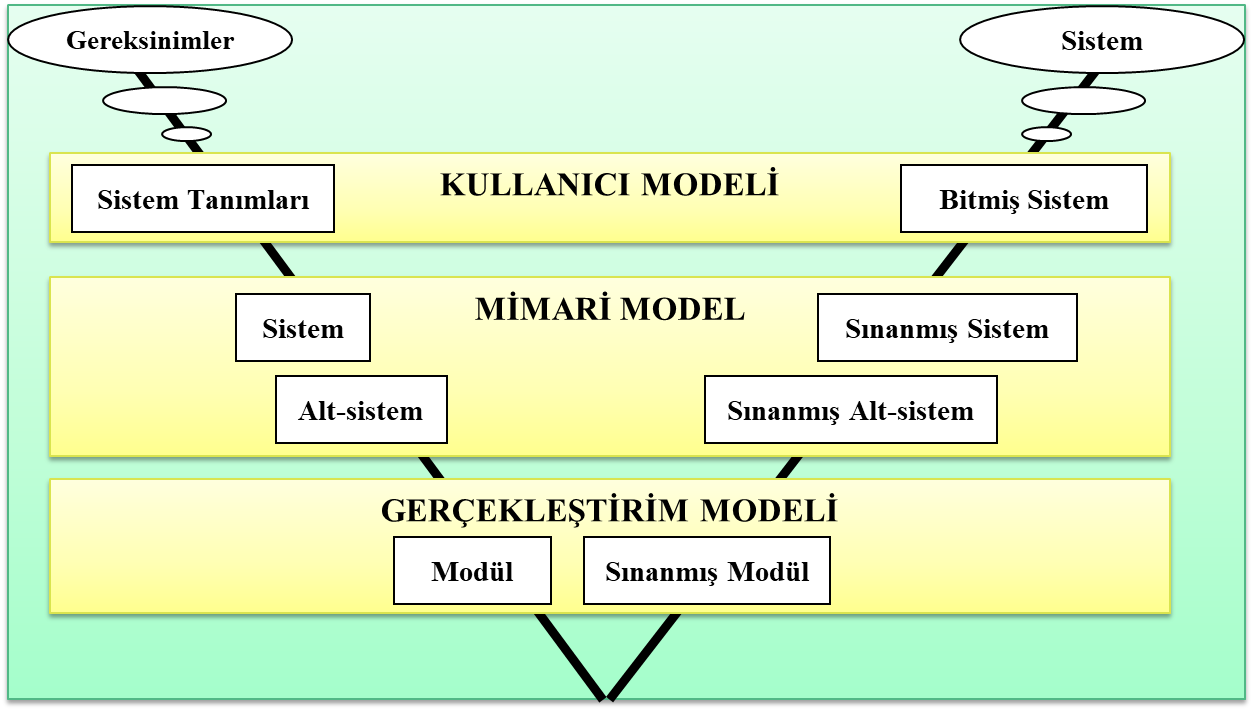
**2)Barok Model:** Bu model 1970’li yıllarda ortaya çıkmıştır. Yazılım yaşam döngüsündeki aşamaların doğrusal olarak uygulandığı bir modeldir. Sırayla gereksinimlerin toplanması, analiz, tasarım, gerçekleştirim, teslimat ve bakım aşamaları uygulanır. Herhangi bir aşamada hata veya eksik çıktığında geri dönüşlerin nasıl yapılacağı tanımlanmamıştır. Barok modelin en önemli özelliklerinden biri gerçekleştirim safhasına çok fazla ağırlık vermesidir. Ayrıca bu modelde diğer modellerden ve günümüzden farklı olarak ‘dokümantasyon’ ayrı bir süreç olarak değerlendirilir. Geliştirme ve test işlemleri bittikten sonra dokümantasyon yapılır. Ancak günümüzde dokümantasyon işlemi yazılım tasarım sürecinin bir parçası olarak görüp her aşamada belgeleme işlemleri yapılmaktadır. Belgeleme, yazılım tasarım sürecinin her aşamasında yazılımcıya yardım eder. Uzun süren yazılım tasarım süreçlerinde projenin başlangıcında neler yapıldığı unutulabilmektedir. Bu da yazılımcıların işini bir hayli zorlaştırmaktadır. Ayrıca bakım aşamasında yazılımcının eksik bilgilerle ürünün bakımını yapması oldukça zordur. Yazılımcıdaki eksik bilgiler, bakımı yaparken veya bir yeri düzeltirken başka yerlerde sorunlar ortaya çıkmasına sebep olabilmektedir. Bu yüzden yazılım tasarım sürecinin her aşamasında belgeleme yani dokümantasyon oldukça önemlidir. Barok modeli, günümüz projelerinin yazılım tasarım süreçlerinde kullanılan bir model olmaktan çıkmıştır.

**3)Çağlayan (şelale) Yaşam Döngü Modeli (waterfall):** Çağlayan yaşam döngü modeli geçmişte en çok kullanılmış olan ve en çok bilinen modeldir. Ayrıca en eski yazılım geliştirme modelidir ve geleneksel model olarak bilinir. Bu yaşam döngü modelinde aşamaların en az birer kez tekrar edilmesi gerekmektedir. Bu modelde bir aşama bitmeden diğerine geçilmez. Barok modelinden farklı olarak her aşamanın sonunda belgeleme işlemleri yapılır. Ayrıca yine barok modelden farklı olarak aşamalar arası geri dönüşler yapılabilir ve nasıl yapılacağı tanımlanmıştır. Çağlayan yaşam döngü modelinde müşteri sürecin içerisinde yer almaz. Bu modelin kullanımı basittir. Ancak projelerde oluşabilecek değişimlere karşı oldukça elverişsizdir. Her yapılan değişiklikte ciddi anlamda maliyet artmaktadır. Değişikliklere ve gelişime açık bir olmadığından dolayı müşteri memnuniyetinin sağlanamamasına neden olmaktadır. Gereksinimlerin belirlenmesi aşaması çok iyi yapılmaktadır. Sonrasında analiz ve tasarım aşamalarında tüm gereksinimler karşılanmaya çalışıldığından dolayı bu aşamalar oldukça uzun sürmektedir. Bu süreçte müşterinin istek ve ihtiyaçlarının değişmesi ve müşteriyle olan iletişimin iyi olmamasıyla beraber ciddi sorunlar doğmaktadır. Hem maliyet artmakta hem de yazılım ürününün teslim süresi uzamaktadır. Proje için çalışan yazılım ekipleri genellikle, bir an önce programı yazmak kodlama yapmak ve bir an önce ürün ortaya koymak istediklerinden dolayı bu modelle yürütülen projelerde çalışan yazılım ekipleri mutsuzlaşmaktadır ve kodlama dışındaki safhalara verdikleri önem azalmaktadır. Bu model küçük ve gereksinimleri çok iyi anlaşılmış olan projelerde işe yaramaktadır. Ancak daha karmaşık, uzun veya nesne yönelimli projeler için uygun değildir. Günümüzde şelale modelinin kullanımı gün geçtikçe azalmaktadır.



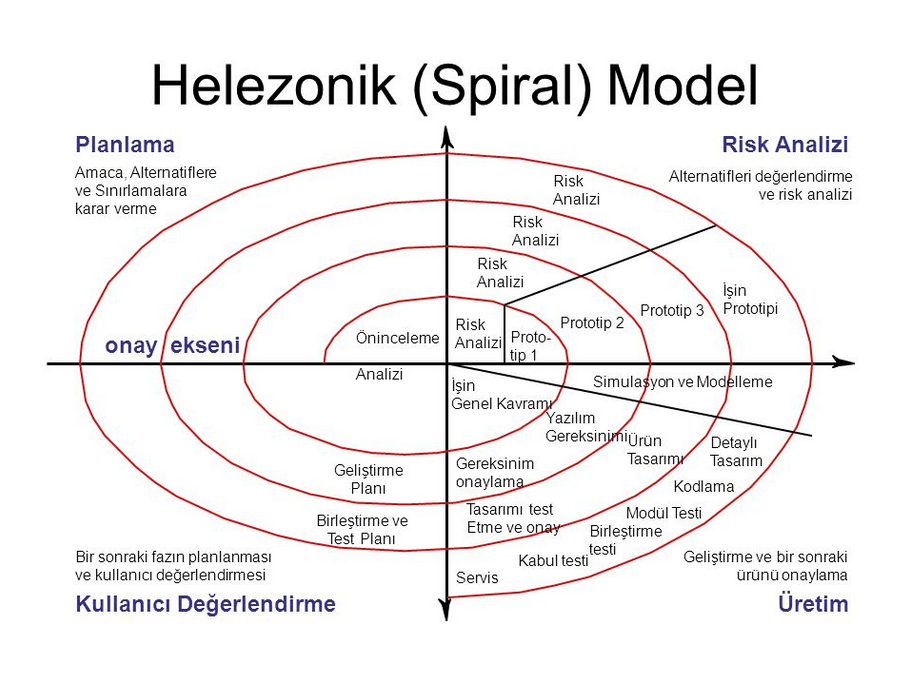
Şekil :Çağlayan/şelale modeli

**4)V Süreç Modeli (V-shaped model):** Bu model şelale modelinin bir uzantısıdır. Şelale modelinden daha gelişmiş bir modeldir. Bu modelde her aşamaya karşılık gelen bir test aşaması vardır. V süreç modelinin sol tarafı üretim, sağ tarafı sınama işlemlerini gösterir. V süreç modelinin üç temel çıktısı vardır. Bunlardan birincisi kullanıcı modelidir. Kullanıcı modelinde, geliştirme sürecinin kullanıcıyla olan ilişkileri tanımlanır. Sistemin nasıl onaylanacağına ilişkin sınama belirtimleri ve planlar ortaya konulur. İkincisi mimari modeldir. Mimari modelde sistemin tasarımı ve oluşacak olan alt sistemin ve tüm sistemin sınama işlemleri ortaya konulur. Üçüncüsü ise gerçekleştirim modelidir. Gerçekleştirim modelinde yazılım modülünün kodlarının yazılması ve sınama işlemleri ortaya konulur. Bu modelin avantajlarından biri kullanımının ve takibinin kolay olmasıdır. Sınama ve onaylama işlemleri erken aşamalarda vurgulanır. Bu model belirsizliklerin az olduğu aynı zamanda iş tanımlarının net bir şekilde belirlendiği bilgi teknolojileri projeler için uygundur. Bu model ile beraber kullanıcının sisteme katkısı artmaktadır. Risk çözümlemeleri ve tekrarların olmaması modelin kötü yönleridir.



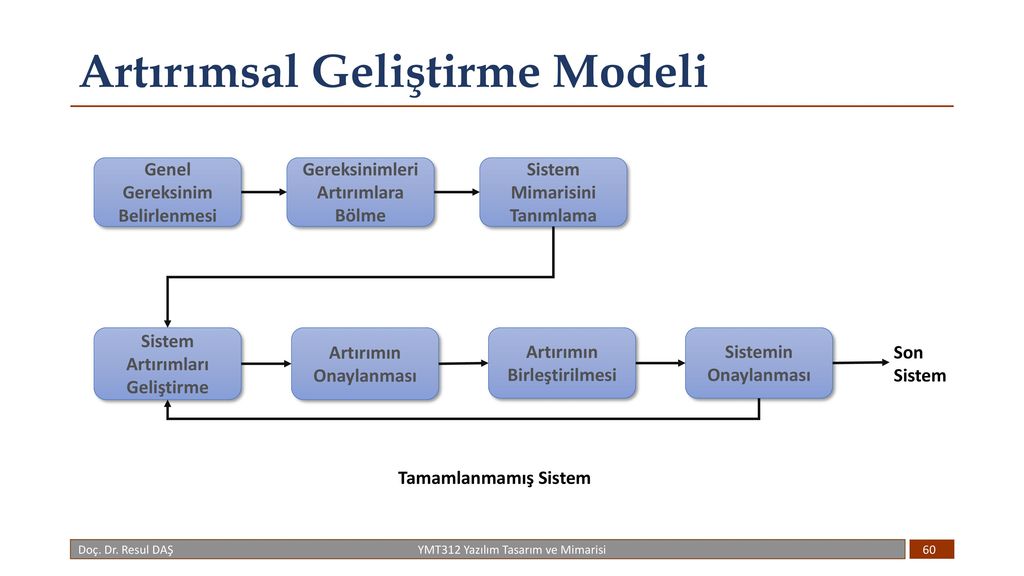
Şekil : V modeli

**5)Helezonik (spiral) Model:** Bu model aynı aşamalara geri dönmenin şart olduğunu vurgular. Spiral modelde, şelale modelinde dikkate alınmayan risk analizi öne çıkar. Yinelemeli artımsal bir yaklaşım vardır. Yazılım yaşam döngü süreci dört faza ayrılır. Bunlar planlama, risk analizi, üretim ve kullanıcı değerlendirmesidir. Planlama aşamasında, her döngüde üretilecek ara ürün için planlama yapılır. Bir önceki adımda üretilen ara ürünle birleştirme işlemi gerçekleştirilir. Risk analizi aşamasında, yazılımda ortaya çıkabilecek her türlü riskin araştırılması yapılır ve risk durumlarında neler yapılabileceği belirlenir. Üretim aşamasında, ara ürünler üretilir. Kullanıcı değerlendirmesi aşaması, kullanıcıya sunulan ara ürünlerin kullanıcı tarafından sınandığı ve değerlendirildiği aşamadır. Her fazda bir prototip üretilip risk analizi yapılır. Modelin riske duyarlı olması zorlukları engeller ve olası hataların erken aşamalarda giderilmesini sağlar. Bu modelde kullanıcı sistemi erken aşamalarda görme ve değerlendirme imkanı bulur. Bu model küçük ve düşük ölçekli projeler için maliyetlidir. Daha çok büyük ölçekteki projeler için kullanılmalıdır. Helezonik model karmaşıktır. Ara adımların çok fazla olması sebebiyle oldukça fazla dokümantasyon gerektirir.



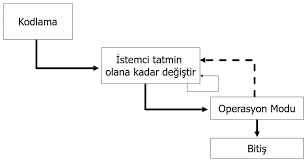
Şekil : Helezonik model

**6)Artımsal Geliştirme Süreç Modeli:** Bu modelde bir yandan üretim yapılırken diğer yandan kullanım yapılır. Bu model bir döngü halinde işler. Gereksinimlerin belirlenmesi müşterilerle beraber yapılır. Müşterinin en temel ve en önemli gereksinimlerine öncelik verilir. Analiz, tasarım ve gerçekleştirim aşamalarından sonra ürünün, müşterinin temel gereksinimlerini karşılayan bir versiyonu ortaya çıkarılır. Ürünün bu modelinde hatalar çıkabilir. Ancak bu hatalar gereksinimlerin daha iyi anlaşılmasını sağlayacaktır. Hataların da bulunduğu bu modelle döngüye devam edilir. Artımsal geliştirme süreç modelinde, müşteriye gereksinimlerini karşılayacak olan ürün aşama aşama geliştirilerek sunulur. Erken aşamalarda kullanıcıya sunulan ürünler sonraki aşamalar için prototip görevi görür. Gereksinimlerde oluşabilecek değişiklikler sonraki teslim ürünlerine aktarılır. Bu modelin en önemli özelliği fazlasıyla test imkanının olmasıdır. Bu da projelerin tamamen batma riskini azaltır. Yani bu modelin başarısızlık riski azdır. Uzun zaman alması öngörülen ve eksikleri zamanla eklenebilecek olan projeler için uygundur. Deneyimli çalışan gerektiren bir modeldir.



Şekil : Artımsal geliştirme modeli

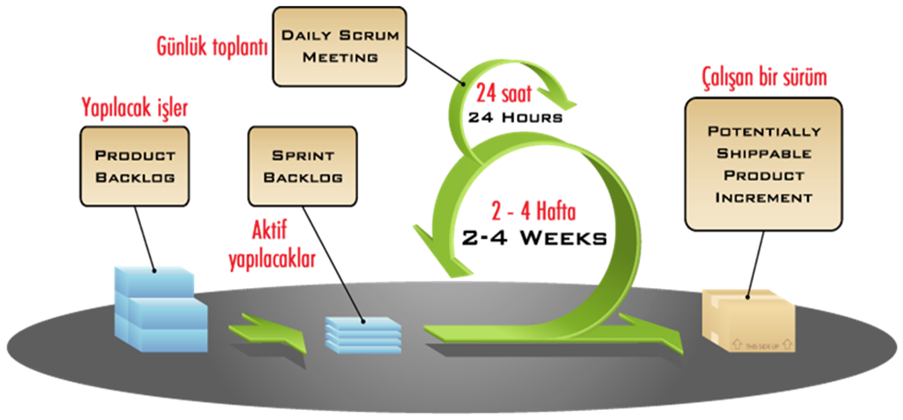
**7)Kodla ve Düzelt Modeli (cod and fix):** Kodla ve düzelt modelinde herhangi bir planlama yoktur. Direkt olarak yazılıma ve kod yazmaya başlanır. Müşterinin istediği ürün elde edilene kadar eksikler tamamlanmaya çalışılır. Gereksinimlerin belirlenmesi, analiz, tasarım aşamaları direkt olarak geçildiği için müşterinin ne istediği tam olarak bilenememektedir. Bazı zamanlarda müşteri de ne istediğini bilememektedir. Ve bu gibi durumlarda ürünün teslim tarihi gecikmektedir. Direkt olarak kod yazılmaya başlandığı için dokümantasyon yoktur. Kodlar sonradan değişime uygun olarak yazılmadığı için esnek değildirler. Dokümantasyonun olmaması da yazılım ürünlerinin bakım aşmasında büyük zorluklar çıkarmaktadır. Yazılım ürünü elde etmenin en kolay ama an çok masraflı ve maliyetli olan yoludur. Erken aşamalarda testler yapılmadığı için kodun yazılması bittikten sonra yapılmak istenen değişimler ciddi masraflara yol açmaktadır. Birkaç yüz satırdan oluşan kısa ve küçük çaplı programlar için kullanılır. Bu model tecrübesiz, yeni kurulmuş firmalarda veya öğrenci projelerinde kullanılan bir modeldir. Çoğunlukla ortaya spagetti kod çıkar. Kodlar esnek olmadığından dolayı çok uzun yıllar kullanılabilecek ürünler elde edilmez ve bir süre sonra yeni sistemlere ihtiyaç duyulur. Bu yüzden bu model diğer modellerden farklı olarak ‘emeklilik (retirement)’ aşaması içerir.



Şekil : Kodla düzelt modeli

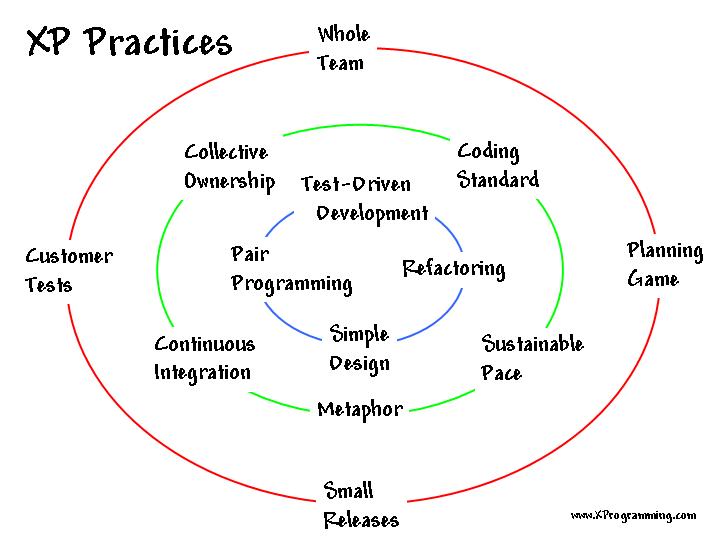
**8)Çevik Yazılım (Agile Model):** Yazılım dünyasında, yazılım ürünlerinin müşterilere zamanında teslim edilememesi ve teslimat süresinin çok uzun olması, müşterilerin gereksinimlerinde veya isteklerinde meydana gelen değişikliklere kolay adapte olunamaması, sistemdeki hataların geç fark edilmesi ve maliyetin artması gibi sorunlar ortaya çıkmıştır. Bu sorunlardan dolayı yazılım projelerinde tam anlamıyla başarı ve müşteri memnuniyeti elde edilememiştir. Tüm bu sorunların giderilmesi amacıyla 1990’lı yılların sonlarında çevik yazılım metotları geliştirilmiştir. Bu metotlar ürün teslimatını hızlı ve sık yaparak müşteri memnuniyeti sağlamaya çalışır. Yazılım ürünlerinin en kısa sürede müşteriye sunulmasını amaçlar. Bu metotlarda basitlik ilkesi önemlidir. Müşterilerin gereksinimlerinde veya isteklerinde meydana gelen değişiklikleri sisteme kolaylıkla adapte etmeyi ve bunları değişiklik yönetim planları üzerinden yapmayı öngörür. Dokümantasyona ciddi anlamda önem verir. Projelerin, motivasyonu yüksek ve kendinden örgütlü takımlar tarafından yürütülmesi gerektiğini savunur. Bu metotlar, yazılım yaşam döngü sürecinde en etkili ve en iyi iletişim yolunun yüz yüze görüşme olduğunu savunur. Bu metotlarla yürütülen projelerde çalışan ekipler üzerinde baskı vardır. Sürekli olarak değişen ihtiyaçlar, ekibin sürekli çalışmasını gerektirir. Bu metotların kurumsal yapılarda kullanılması zordur.

**Scrum:** Bir çevik yazılım yöntemidir. İsmini rugby sporundaki bir hücum yönteminden almaktadır. Scrum yöntemi sadece yazılım geliştirme süreçlerinde değil, her alanda uygulanabilecek bir yöntemdir. Scrum, 1990’ların ortalarında Jeff Sutjerland ve Ken Schawaber tarafından bulunmuştur. Günümüzün en popüler yöntemlerinden biridir. Artımsal bir yaklaşım sağlar. Büyük yazılım projelerini küçük alt birimlere (sprint) ayırarak ilerlemeyi öngörür. Yazılım ekibi günlük 15 dakikalık kısa toplantılarla projenin takibini yapar. Scrum 30 günden fazla olmamalıdır. Bu yöntem, karmaşık ortamlarda aşama aşama yazılım ürünü geliştiren ekipler için uygundur. Sprintler scrum ekbiyle gerçekleştirilir. Product backlog, müşteri ile daha önceden konulup temel ihtiyaçların, isteklerin öncelik sırasının yapılmış olduğu detaylı bir gereksinimler listesidir. Sprint backlog, çoğunlukla 2-4 hafta süren proje zaman dilimidir. Bir proje birden fazla sprintten meydana gelebilir (örn: 2 aylık veya 4 aylık projeler). Scrum daily meeting, scrum ekibinin her gün düzenli olarak 15 dakika (max 30 dk) boyunca ayakta yaptığı toplantılara verilen isimdir. Bu toplantılarda ekipteki herkes dün ne yaptığını veya ne yapamadığını, yapamadıysa neden yapamadığını, onu engelleyen şeyin ne olduğunu ve bu gün ne yapacağını açıklar. Burada ürün yöneticisi/sahibi (product owner), müşteri le gereksinimler hakkında konuşan kişidir. Scrum yöneticisi (scrum master), ekibin scrum’ın niteliklerine ve kurallarına bağlı kalmasını sağlamaktan ve scrum’ın sorunsuz şekilde ilerlemesini sağlamaktan sorumludur. Scrum ekibi ise 5-9 bireyden meydana gelir. Scrum ekibinde alt-üst ilişkileri bulunmaz, herkes aynı seviyededir. Sprint sonunda ürün sahibi (product owner), gereksinimlerin karşılanıp karşılanmadığını söyler. Gerekli görülürse, gereksinimler güncellenir. Scrum’da sprint kalan zaman grafiği (burndown chart) adında bir grafik vardır. Bu grafik, sprint boyunca projenin ne kadarının yapıldığını, ne kadarının yapılması gerektiğini gösteren grafiktir.



Şekil : Scrum modeli

**Extremme Programming (XP):** XP, Kent Beck tarafından 1999 yılında ortaya konmuştur. Başta bir yazılım geliştirme disiplini ortaya çıkmıştır. XP, kendisine kadar olan modellerde başarısızlığa neden olan noktaları barındırmamayı öngörür. Örneğin bir projede başarı elde etmek için müşterinin gereksinimleri çok iyi anlaşılmalıdır. XP’de müşteri her aşamada vardır. İletişim yüzyüzedir. Yazılımcılarla müşteriler arasındaki ilişkininin çok iyi ve sağlıklı olması gerektiğini savunur. XP, karmaşık problemlere basit çözümler üretmeyi hedefler. Komplex çözümler bu modele uygun değildir. Günün öncelikli, zorunlu ihtiyaçlarını hedef alarak basit çözüm yolları arar. Esnek bir sistem oluşturmayı hedefler. Bu modelde 2-4 hafta aralıklarla yazılım ürünün geliştirilen sürümleri ortaya konulur. Müşteri, yazılım ekibi ve yönetici, ürünün bu sürümüyle ilgili değerlendirmelerde bulunurlar. Burada müşteri proje grubunun bir parçası olarak ele alınır. Bu değerlendirmeler sayesinde geliştirilen sistem hakkında geri bildirimler alınır. Böylece ilerde ortaya çıkabilecek sorunlar en aza indirgenir. XP’nin savunduğu önemli bir diğer nokta ise başarısızlıklardan kaçmamak gerektiğidir. Başarısızlığın üzerine gidilerek hataların kısa sürede telafi edilmesi gerektiğini savunur. Buna göre başarısız olmaktan korkmak yazılımcıyı geriye götürür ve projenin ilerleme hızını düşürür. Yani XP’nin savunduğu en temel 4 değer; iletişim, basitlik, geri bildirim ve cesarettir. XP, yazılım geliştirme sürecinde kolaylık ve esneklik sağlamak amacıyla 12 temel pratiği öngörmüştür. Bunlar; planlama oyunu, ekipte müşteri, önce test, basit tasarım, çiftli programlama, sürekli entegrasyon, kısa aralıklı sürümler, yeniden yapılandırma, ortak kod sahiplenme, metafor, kodlama standardı, haftada 40 saattir. Planlama oyununda, müşterinin sonraki sürümler için istediği yenilikler öğrenilir ve yazılım ekibi o işin ne kadar sürede yapılacağıyla ilgili tahminlerde bulunur. Yazılım ekibi belirlenen süre içerisinde müşterinin isteklerini yerine getirmeye çalışır. Ekipte müşteride, müşteri tüm süreçlerde ekibin bir parçası olarak görülür. Böylelikle yazılımcının ihtiyacı olan tüm bilgiler kısa sürede elde edilir. Bu da ürünün en az hatayla ve en kısa sürede oluşturulmasına katkı sağlar. Önce testte, kodlama yapılmadan önce bir test programı yazılır. Böylece sonradan ortaya çıkabilecek olası hatalar erken aşamada tespit edilmiş olur. Basit tasarımda, müşterinin üründen beklentisine ve isteğine göre en basit tasarım yapılır. Böylece üzerinde çalışılabilecek ve değişiklikler yapılabilecek bir yazılım oluşmuş olur. Çiftli programlamada (pair programming), usta çırak ilişkisi vardır. Özellikle işe yeni başlayan yazılımcılar için faydalıdır. Hem iki yazılımcının aynı anda bir probleme odaklanması problemin daha hızlı çözülmesini sağlayacaktır hem de ileride görülebilecek olası hata ve sorunlar daha erken aşamada fark edilecektir. Sürekli entegrasyonda; sistemde yapılan değişiklikler anında sisteme entegre edilir. Böylece oluşabilecek olası hatalar için erken tespit yapılmış olur. Kısa aralıklı sürümlerde, proje 2-4 haftalık zamanlara bölünür. Her bölümde belirlenen işin teslim süresi vardır. Böylece her toplantıda, özellikleri biraz daha gelişmiş ve yaşayan bir ürün ortaya konulmuş olur. Yeniden yapılandırmada, yazılımcı müşteriyi memnun etmek amacıyla kod ve tasarımı sürekli olarak gözden geçirir. Ortak kod sahiplenmede, üretilen proje tüm ekibin ortak malıdır. Ekipteki herkes yazılıma ulaşabilir ve yazılımı geliştirebilir. Benzetimde (metaforda), parçalara ayrılan projede her birim geliştirilmeye çalışırken aynı zamanda birbirine benzetilmeye de çalışılır . böylece parçalar birleştirildiğinde kaliteli bir ürün elde edilmiş olur. Kodlama standartında, ekip üyeleri önceden belirlenmiş olan kodlama standartlarına göre kod yazar. Böylece karmaşıklık en aza indirilirken yazılımın da tüm ekip tarafından anlaşılması sağlanır. Haftada 40 saat uygulamasında, hafta içi her gün 8 saat çalışılması öngörülür. Haftada sadece bir kez mesaiye kalınabilir. Bu uygulama, haftada kırk saatten fazla çalışmanın verimi arttırmadığını savunur.



Şekil : XP modeli

Sonuç olarak bu yazıda; bir yazılım ürünü geliştirilirken hangi yazılım yaşam döngü modellerinden faydalanılabileceğini anlattık. Hangi projeler için hangi yaşam döngü modelinin kullanılmasının daha uygun olabileceğinden bahsettik. Projeleri hayata geçirirken müşteriyi her aşamada sürecin içinde tutmamız gerekmektedir. Projeler için uygun yazılım yaşam döngü modelleri seçildiğinde hem ürünün teslimat süresi kısalmış hem maliyeti düşmüş hem de nitelikli ve hata oranı az ürünler edilmiş olmaktadır.

**REFERANSLAR**

* <https://medium.com/@oemine075/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-cb3670054591>
* <https://medium.com/@halilyesil3506/yazilim-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri%CC%87-efc28dea3bf0>
* <https://medium.com/@haliltoksz/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-d2e8696ab50e>
* <https://medium.com/@furkaneren_81038/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-203fe380b5e7>
* <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>
* <https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>
* <http://bilgisayarkavramlari.sadievrenseker.com/2013/01/10/sdlc-yazilim-gelistirme-yasam-dairesi/>
* <https://yazilimtestikariyerrehberi.blogspot.com/2018/02/v-model-nedir.html>
* <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>