**YAZILIM YAŞAM -DÖNGÜ MODELLERİ (SOFTWARE LİFE-CYCLE MODELS)**

Yazılım geliştirme yaşam döngüsü (SDLC), yüksek kaliteli, düşük maliyetli yazılımların mümkün olan en kısa sürede üretilmesini sağlayıp müşterideki işleyişini de takip eden sistematik bir süreçtir. SDLC’ nin amacı müşterinin tüm beklenti ve taleplerini karşılayan ve aşan üstün yazılımlar üretmektir. SDLC’ a bağlılık yazılımın geliştirme hızını arttırır ve diğer yönetim yöntemleriyle ilişkili proje risklerini ve maliyetini en aza indirir. Tabi ki sürekli değişim ve gelişme içinde bulunan Yazılım işlevlerinden dolayı tüm bu aşamalar bir döngü içinde ele alınıp adımlar arasında geriye dönülüp tekrar ilerlemek söz konusudur. Temelde 5 adım vardır:

**1.Planlama (Gereksinim /Requirements**): Planlama aşamasının (fizibilite aşaması olarak da adlandırılır) ve müşteriden gereksinimlerin toplandığı adımdır. Bu adımda yazılım geliştirme döngüsü için bir taslak geliştirilerek sorunlar gelişmeyi etkilemeden önce teorik olarak yakalanır bu da ihtiyaç duyulan finansman ve kaynakların güvence altına alınmasına yardımcı olur. Belki de en önemli olan bu adım belirli bir zamana kadar gönderilmesi gereken ticari bir ürün için kilit öneme sahip olabilecek proje takvimini belirler.

**2.Analiz (Analysis):** Bu adımda son kullanıcıların ihtiyaçlarını belirlemek için analiz ve araştırma yapılır (dokümante edilir), gereksinimler açığa kavuşturulur ve ne gibi risklerin ortaya çıkabileceği belirlenir.

**3.Tasarım (Design):** Tasarım aşamasında geliştiriciler ve teknik mimariler, her gereksinimi karşılayabilmek için yazılımın ve sistemin üst düzey tasarımına başlar. 2 tür tasarım vardır: **1) Yüksek Düzeyde ve Mimari Tasarım:** Yazılım modüllerinin etkileşimi ile ilgilenir. **2) Detaylı Tasarım:** Bu bölümde de Mimari Tasarımın dökümanları düzeltilir.

**4.Gerçekleştirim (Implementation):** Bu adımda geliştiriciler yazdıkları yeni kodu test etmek, birbirlerinin kodunu gözden geçirmek, derlemeler oluşturmak ve bir ortama yazılım dağıtmak adına her bileşen için birim testler yazarlar. Bu geliştirme döngüsü, gereksinimler karşılanana kadar tekrarlanır. (Kısaca Kodlama ve Test aşamasıdır.)

**5. Teslim ve Bakım (Delivery and Maintenance):** Son kullanıcıya teslimde ürün tek başına yeterli değildir yanında mutlaka kullanım kılavuzu ve versiyon fark dokümanı oluşturulmalıdır. Yani Yazılım sahaya teslim edildikten sonra SDLC sona ermez. Geliştiriciler artık bir bakım moduna geçmeli ve son kullanıcılar tarafından bildirilen sorunları ele almak için gereken tüm etkinlikleri uygulamaya başlamalıdır. Ayrıca, dağıtımdan sonra yazılımın ihtiyaç duyabileceği değişiklikleri uygulamaktan da geliştiriciler tabi ki sorumludur.

Yazılım yaşam döngüsünün temel adımları çekirdek süreçler (core process) olarak da adlandırılır. Bu süreçlerin gerçekleştirilmesi amacıyla Yazılım Belirtim Yöntemleri ve Yazılım Süreç Modelleri kullanılmaktadır.

**Yazılım Belirtim Yöntemleri (Specifications):** Çekirdek sürece ilişkin fonksiyonları yerine getirmek amacıyla kullanılır. Süreç akışı için kullanılan belirtim yöntemleri: Süreçler arası ilişkilerin ve iletişimin gösterildiği yöntemlerdir. (Veri akış şemaları, yapısal şemalar, nesne/sınıf şemaları). Süreç Tanımlama Yöntemleri: Süreçlerin iç işleyişini göstermek için kullanılan yöntemlerdir. (Düz metin, algoritma, karar tabloları, karar ağaçları, anlatım dili). Veri Tanımlama Yöntemleri: Süreçler tarafından kullanılan verilerin tanımlanması için kullanılan yöntemlerdir. (Nesne ilişki modeli, veri tabanı tabloları, veri sözlüğü).

**Yazılım Süreç Modelleri (Processes):** Yazılım Yaşam Döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi sırayla nasıl uygulanacağını tanımlar. Modellerin ortaya çıkmasında, ilgili dönemin donanım ve yazılım teknolojileri ile sektör ihtiyaçları önemli rol oynamıştır. Kısaca Yazılım üretim işinin genel yapılma düzenine ilişkin rehberler olarak kullanılırlar. Bu modeller aynı zamanda yönetici ve geliştiricilerin, yazılım geliştirme sürecinin karışıklılığı ile baş etmelerini saplarlar. Günümüzde 50’ den fazla model bulunmaktadır. Bu süreç modelleri 3 başlık altında incelenebilir: Düzenleyici Süreç Modelleri, Birleşik Süreç, Çevik Yazılım Süreci.

**1.Düzenleyici Süreç Modelleri:**

**-Kodla ve Düzelt (Code and Fix):** Küçük ve kısa projeler (birkaç yüz satırdan oluşan programlar) için uygun olabilir. Emeklilik (retirement) safhası vardır. Genellikle zaman tasarrufu olarak kabul edilir ve düşük bütçeli projeler için uygun bir modeldir. Code and Fix, ürün geliştirme süresini kısaltıyor gibi görünebilir. Ancak, gereksinimlerin somut bir şekilde anlaşılmaması nedeniyle (dokümantasyon yok) müşteri memnuniyeti ve hatta başarısızlık için büyük bir risk vardır. Ayrıca, büyük bir mühendislik ekibi olduğunda, sürekli iletişim sorunları olacağı için bu model güvenilir bir şekilde çalışmayacaktır. Yeni geliştirmeler ve özellik güncellemeleri, uygulamanın büyük bölümlerinin yeniden yazılması gerekebileceğinden, işlemin sonlarında önemli mimari sorunlara neden olabileceğinden, gelecekte değişiklik gerektiren uygulamalar için Kod ve Düzeltme kullanmak son derece risklidir.

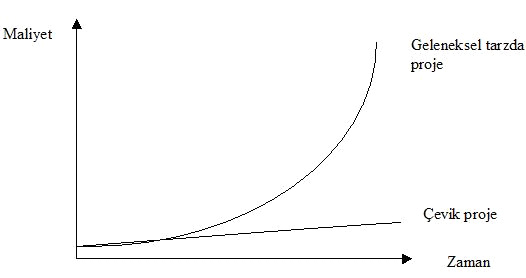
Orta ve büyük ölçekli yazılım projeleri için tasarlanmadığı için çoğu durumda en az etkili geliştirme metodolojisi olarak kabul edilebilir. Yazılım ekipleri, zaman baskısı nedeniyle genellikle Code and Fix'i kullanır.

Özetle bu modelde önce projenin ilk versiyonu çıkarılır bu versiyon müşterinin gereksinimleri karşılanana kadar düzenlenir teslim sonrası bakım aşamasından da geçilerek emeklilik adımıyla son bulur.

**-Gelişi Güzel Model (Random):** Bu süreç modelinde herhangi bir yöntem yoktur. Yazılım tamamen geliştiren kişiye bağlıdır. Bu yüzden bakımı çok zordur. Daha çok 1960’ lı yıllarda kullanılan bir yöntemdir. Geliştirilen yazılımın programlanması diğer metotlarla geliştirilen yazılımların programlamasına göre basittir. Herhangi bir kuraldan bağımsız, yalnızca geliştiren kişiye bağımlı hatta onun bile bir zaman sonra anlayamadığı ve değiştirme zorluğu çektiği modeldir. 60’ lı yıllarda tek kişinin ürettiği programlar böyledir. Basit öğrenci projeleri böyledir.

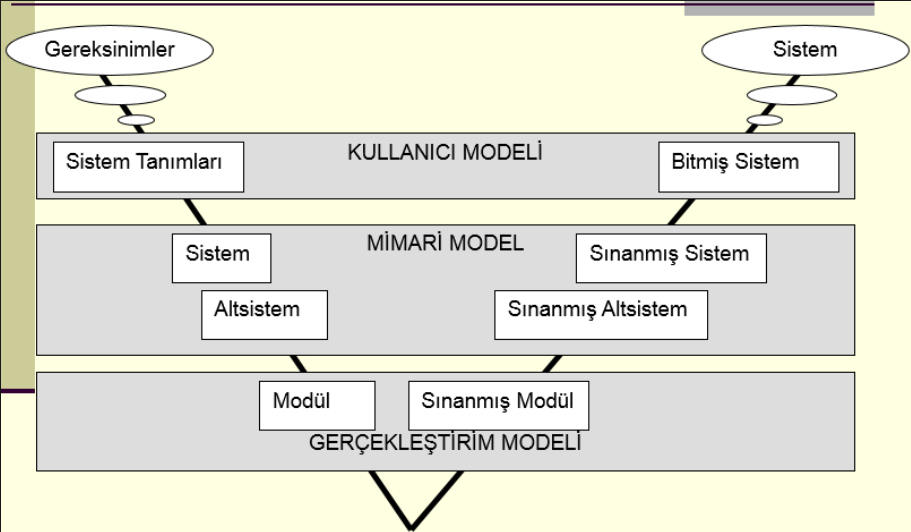
**-Barok Modeli:** Bu modelde yazılım yaşam döngüsünün temel adımları doğrusal bir biçimde takip edilir. 1970’li yıllarda ortaya çıkmıştır. Adımlar arası ilişkilerin tanımlı olmadığı bir yöntemdir. Günümüzde yazılım geliştirme projelerinde kullanılmayan bir yöntemdir. Bunun en büyük nedeni ise “dokümantasyon” adımının bu modelde ayrı bir adım gibi ele alınıp yazılımın geliştirilmesi ve testinin ardından yapılmasıdır. Günümüzde kullanılan modellere aykırı bir durumdur. Günümüzde tercih edilen modellerde “dokümantasyon” geliştirilen yazılımın ürünü olarak kabul edilmektedir. Ayrıca “dokümantasyon” adımına daha fazla ağırlık veren bir modeldir. Barok modelinin adımları: 1.İnceleme, 2.Analiz, 3.Tasarım, 4.Kodlama, 5.Modül testleri, 6.Alt sistem testleri, 7.Sistem testi, 8.Dokümantasyon, 9.Kurulum.

**-Çağlayan (Şelale) Yaşam Döngü Modeli (Waterfall):** Çağlayan modeli SDLC’ de kullanılan ilk modeldir. Diğer Yaşam Döngü Modellerine kıyasla Şelale Modeli çok daha durağandır. Örneğin bir köprü yapılacak. İstanbul’ da Boğaz Köprüsü yapıldı ve onlarca yıldır hiç değişmiyor ne beklentide farklılıklar var ne de sistemde. Kimse, köprüye her hafta yeni bir kat çıkalım, altından daha büyük gemilerin geçebilmesi için köprüyü biraz daha yükseltelim demiyor ve zaten bunu her gün talep etmek gibi bir dinamiklik de söz konusu değil. Ancak Yazılım projelerinde bütün bunları görebilirsiniz. Bazen projenin yeniden oluşturulmasına kadar giden taleplerin arka arkaya aynı gün içerisinde geldiği bile olur. Oysaki Boğaz Köprüsü yıllardır hiç değişmeyen bir projedir. Çağlayan Modeli bu projeler için uygun bir modeldir. Ancak Yazılım dünyası böyle değildir projelerde sürekli değişiklikler yenilikler söz konusudur. Çağlayan Modelinde SDLC temel adımları baştan sona en az bir kez izleyerek gerçekleştirilir. Temelde Analiz, Tasarım, Kodlama, Test, Sürüm ve Bakım safhalarından oluşur. İyi tanımlı ve üretimi az zaman gerektiren yazılım projeleri için bu modelin kullanımı uygundur. Geleneksel Model olarak da bilinen bu modelin günümüzde kullanımı baştaki örnekten de yola çıkıp anlayacağımız gibi giderek azalmaktadır. Barok Modelinin aksine “dokümantasyon” işlevini ayrı bir aşama olarak ele almaz ve üretimin doğal bir parçası olarak görür. Barok Modele göre geri dönüşler iyi tanımlanmıştır. Yazılım tanımlamada belirsizlik yok (ya da az) ise yazılım üretimi çok zaman almayacaksa uygun bir süreç modelidir. Bir sonraki aşama önceki aşama tamamlanmadan başlayamaz (lineerdir/doğrusaldır). Her aşamanın sonucu bir ya da birden fazla onaylanan belgedir. Gerektiğinde geliştirme aktivitelerinde tekrarlamalar olabilir. En önemlisi proje başlangıcında her detayı göz önünde bulundurmak mümkün olmadığı için, çağlayan modeliyle geliştirilen yazılım sistemlerinin müşteri gereksinimlerini tam tatmin etmez. Bunun önüne geçebilmek için projenin başlangıç safhasında Analiz için çok zaman harcanır ve müşteri gereksinimleri en ince detayına kadar tespit edilir. Kullanıcı katılımı başlangıç safhasında mümkündür. Kullanıcı gereksinimleri bu safhada tespit edilir ve detaylandırılır. Daha sonra gelen tasarım ve kodlama safhalarında müşteri ve kullanıcılar ile diyaloga girilmez. Bu model kullanıp, ilk sürümün teslimat safhasına gelindiğinde yeni bir yazılım gereğine cevap vermeniz mümkün olmayabilir çünkü bütçenin %90’ ına yakınını sarf etmiş olabilirsiniz.



Bu çerçeveden bakıldığında proje sonunda oluşan program müşterinin aktüel gereksinimlerini tatmin etmez durumdadır. Program daha çok müşterinin proje başlangıcında sahip olduğu gereksinimleri karşılayacak şekilde tasarlanmıştır. Projelerin birkaç sene boyunca sürebileceğini düşünürsek, aslında bu süreç sonunda oluşan program aktüel değildir.

**-V Süreç Modeli (V Process Model):** Yazılım geliştirme süreç modelleri açısından uygulanabilirliği en kolay olan ve gerektirdiği maliyetler açısından en uygun olan model V-Modelidir. V-Modeli yazılım geliştirme sürecini koordine eden bütünleşik ve birleştirici metotların toplamıdır. Ana aktivite alanlarının birbirleri arasında koordinasyonunu sağlayan, tüm aktiviteleri içerik bakımından tanımlayan, proje kapsamında üretilecek tüm yazılım hakkında dokümantasyon bilgileri içeren ve bu dokümantasyonlar hakkında detaylı açıklama getiren ve son olarak da her seviye arasında problemin tanımlandığı andan ürün sürümüne kadar geçen tüm aşamalarda içsel kontrollü bir metodolojiler toplamıdır. Bu model adından da anlaşılacağı gibi “V” yapısında bir yol izlenir ve adımlar bu şekilde gerçekleştirilir:



Sol taraf üretim, sağ taraf sınama işlemleridir. Bu modelde yer alan temel çıktıları “Kullanıcı Modeli”, “Mimari Model” ve “Gerçekleştirim Modeli” adı altında toplayabiliriz:

-Kullanıcı Modelinde geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine dair sınama belirtimleri ve planları ortaya çıkarılmaktadır.

-Mimari Modelde sistem tasarımı ve oluşacak alt sistem ile tüm sistemin sınama işlemlerine ilişkin işlevler ele alınmaktadır.

-Gerçekleştirim Modelinde de yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar ele alınmaktadır. Bu model belirsizliklerin az iş tanımlamalarının belirgin olduğu bilişim teknolojileri (BT) projeleri için uygun bir modeldir. Ayrıca bu model kullanıcının proje katkısını arttırmaktadır. BT projeleri 2 aşamalı olarak ihale edilmesi çok uygundur:

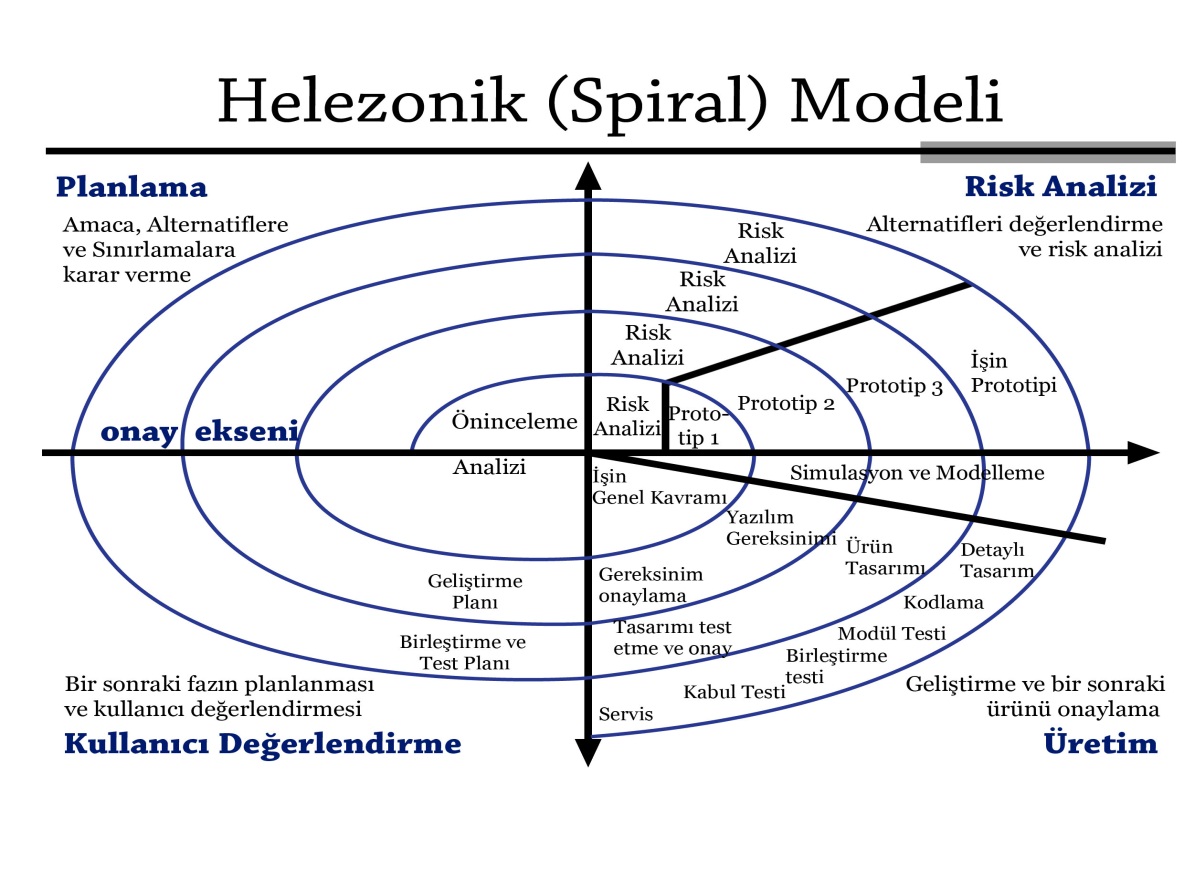
İlk ihalede kullanıcı modeli hedef alınarak, iş analizi ve kabul sınamalarının tanımları yapılmakta,

İkinci ihalede ise ilk ihale sonucunda elde edilmiş olan kullanıcı modeli tasarlanıp gerçekleştirilir.

Güncel olarak E-Devlet Uygulamalarında, Askeri ve Savunma Sanayi Yazılım Projelerinde, Finansal Yazılımlarda, Anahtar Teslim Yazılım Uygulama Projelerinde ve Avrupa Birliği Yazılım Geliştirme standartına uygun projelerde kullanılmaktadır.

**-Prototip Model (Prototyping Model):** Prototip geliştirme modelinde, gereksinimler hızlıca toplanarak işe başlanır. Geliştiriciler ve kullanıcılar aynı masa etrafında buluşarak yazılımdan elde edilecek bütün çıktılara, bu çıktılar için gerekli girdilerin nasıl sağlanacağına, nasıl korunacağına, hangi işlemlere uğrayacağına karar verirler. Daha sonra hızlıca yapılan bir tasarım ile yazılımın kullanıcıya yansıyacak yönünü aktaran bir prototip üretilir. Prototip kullanıcının kullanımına sunulur ve değerlendirme beklenir. Bu değerlendirmelerden yola çıkılarak prototip üzerinde gerekli değişiklikler yapılır ve prototipin yeni hali kullanıcı tarafından yeniden değerlendirilir. Böylece kullanıcının istediği yazılıma iyice yakınlaşılmış bir prototip üzerinde yazılımın neler yapacağı konusunda kullanıcı ile anlaşmaya varılır. Genelde prototipler yavaş, verimsiz ve hantal programlardır. Gerçek bir uygulamada kullanımları söz konusu değildir. Yazılımın gerçekleştirimine geçildiğinde prototip atılır. Ancak müşterinin bu aşamada geliştirilen prototipi gerçek yazılım gibi algılayıp etki beklemeleri tehlikesi vardır. Böylece kendince yapılıp bitirilmiş bir işin bir daha üstelik zor ve zahmetli bir şekilde yeniden yapılmaya çalışıldığını anlamakta zorlanır. Bu yüzden prototip geliştirmenin yalnızca gereksinimlerin doğru belirlenmesi için olduğunun ilk başta kullanıcıya iyice anlatılmış olması gerekir.

**-Helezonik Model (Spiral Model**): Yoğun müşteri katılımının görüldüğü bir modeldir ve çok kapsamlı risk değerlendirmesine odaklanır. Artırımsal (Yinelemeli) ve Şelale Modellerinin bir kombinasyonudur. Asıl amaç her bir artırımın güvenli ve hızlı yapılmasını sağlamaktır. Sarmal Model kullanıldığında her bir artırımda yazılımın yeni bir sürümü ortaya çıkar. İlk artırımda ortaya çıkarılan kesim adeta bir prototipmiş gibi düşünülebilir. Prototipten tek farkı uygulama ortamında kullanılan gerçek yazılım olmasıdır. Sonraki yinelemelerde üstüne yeni işlevler yine bu yaklaşımla mühendisçe eklenecektir. Kalite, verimlilik ve kapasite açısından uygulamanın gereklerine uygun özellikleri taşıması her sarmal döngüde (her yeni sürümde) sağlanarak geliştirilme süreci sürdürülür. Spiral Model 4 adımdan oluşur: Maliyet ve kaynak tahminleri kapsamlı bir **Planlama**, stratejik değerlendirme ve **Risk Analizi**, yazılım ve test aşamasında Prototip oluşturma, daha önce teslim edilen parçaların **Kullanıcı Değerlendirmesi** ve bir sonraki spirale geçmeden önce geliştirme ve bir sonraki ürünü onaylama **Üretim**. Bu tekrarlanan Spiral döngü, proje zaman çizelgesini uzatır. Maliyet bu modelin dezavantajıdır. Küçük projeler için kullanılmaz. Spiral Model kullanım alanları: Belirsiz iş ihtiyaçları olan projeler, inovatif (yenilikçi) ihtiyaçları olan projeler, büyük ve karmaşık projeler, Ar-ge faaliyeti vb.



**-Evrimsel Geliştirme Modeli (Evolutionary Development Model):** İlk tam ölçekli modeldir. Tam olarak “ne istediğimi bilmiyorum ama görsem tanırım” modeli olarak belirlenebilir. Diğer modeller ile kıyaslandığında ilerleyişi daha yavaştır. Modelin tamamının başarımı geçirdiği ilk evrimin başarısına bağlıdır. Coğrafik olarak geniş alana yayılmış, çok birimli organizasyonlar için önerilmektedir (banka uygulamaları). Her aşamada üretilen ürünler, üretildikleri alan için tam işlevselliği içermektedir. Pilot uygulama kullan, test et, güncelle diğer birimlere taşı. Modelin başarısı ilk evrimin başarısına bağlıdır. İki çeşit Evrimsel Geliştirme vardır:

**1)Keşifçi Geliştirme (Exploratory Development):** Hedef: Müşterinin gereksinimlerini incelemek için müşteri ile çalışıp son sistemi teslim etmek.

**2)Atılacak Prototipleme (Throw-awayprototyping):** Hedef: Sistem gereksinimlerini anlamak.

Örneğin yukarıda verilen örnekteki gibi çok birimli banka uygulamalarında;

Önce sistem geliştirilir ve Şube-1’e yüklenir. Daha sonra aksaklıklar giderilerek geliştirilen sistem Şube-2’ye yüklenir. Daha sonra geliştirilen sistem Şube-3’e yüklenir ve belirli aralıklarla eski şubelerdeki güncellemeler yapılır. Banka dışında küçük ve orta boyutlu etkileşimli sistemler (500.000 LOC dan az olan), büyük sistemlerin parçalarında (ör. Kullanıcı ara yüzlerinde), kısa süreli kullanılacak sistemlerde kullanılabilir.

**-Artırımsal Geliştirme Modeli (Incremental Development Model):** Böl ve Yönet yaklaşımıdır. Eğer bir müşterinin ürünlerinde değişikliğe ihtiyaçları varsa, artırımlı model ihtiyaç olan bu değişikliğe ayak uydurur. Artırımsal model bir takvime bağlı olarak yazılımı kısım kısım geliştirip teslim etmeye dayanır. Her bir yeni kısım öncekinin üstüne bazı ek işlevlerin eklenmesini öngörür. Artırımsal model yazılım geliştirmenin kısıtlı sayıda çalışanla işin yapılmasını sağlama gibi bir üstünlüğe sahiptir. Bu modelde bir taraftan üretim bir taraftan da kullanım yapılır. Önceki modellerde ürünlerdeki değişiklikler göz önünde bulundurulmaz. Bu model doğal olarak yinelemelidir. Yeniden kullanılabilir bir ürün, fonksiyonellik sağlamış bir şekilde tüm döngülerin sonunda ortaya çıkar. Yani uzun zamanlı projeler için eksikler çıktıkça yama yapmak tipindeki projeler için uygundur. Aslında çağlayan modelinin örütşen şekilde uygulanmasıdır. Örneğin; tekstil firmasında bir yazılım yazılıyor olsun. Tekstil firmasında iç süreçler değişebilir. Web sayfası üzerinden satış yapılıyor olsun. Web sayfası ile ilgili yeni ihtiyaçlar gelmeye başlar. Bir yandan yazılımın çalışan hali ile ilgili problemler ortaya çıkar. Bir yandan da yeni istekler ortaya çıkar “şu modüle şu ekranı da ekleyelim gibi”. Diğer bir örnek olarak da bitirme tezleri verilebilir. Bu modelle bir dönem boyunca geliştirilmesi gereken bir programlama ödevinin haftada bir gelişiminin izlenmesi. İşte bütün bunlar artırımlı modelde her seferinde arttırıla arttırıla bir iterasyonda sürece dahil edilir.

**-Araştırma Tabanlı Model (Resource Based Model):** Model Yap-at ilk örnek olarak da bilinir. Bunun araştırma ortamları tümüyle belirsizlik üzerine çalışan koşullarıdır. Modelde yapılan işlerden elde edilecek neticeler belirgin değildir. Örneğin; en hızlı çalışan asal sayı test programı, en büyük asal sayıyı bulma programı, satranç programı.

**-Formal Sistem Geliştirme (Formal System Development):** Cleanroom yazılım geliştirme modelidir. Bu model yazılım tasarım ve gerçekleştirmesiyle ilgili matematiksel bir tekniktir. Temelinde karmaşık sistemleri geliştirme ve program geliştirmeye destek yatar. Bu model kullanıldığında kullanıcı sistemi kullanmaya başladığında karşısına çıkan belirtim hatalarını minimize eder. Sistem kullanıma konmadan emniyet ve güvenlik durumları sağlanması gereken kritik sistemlerde kullanılır.

-Bileşen Tabanlı Model (Component-Based Model): Sistemin COTS (“commercial-off-the-shelf”) adı verilen hazır bileşenler kullanılarak tümleştirilmesi esasına dayanır. Adımaları:1) gereksinim belirleme, 2) Bileşen Çözümleme, 3) Gereksinim Düzeltimi, 4) Yeniden Kullanımla Sistem Tasarımı, 5) Geliştirme ve Tümleştirme, 6) Sistem Geçerleme. Günümüzde kullanımı sınırlıdır.

**Metodoloji Nedir?** Metodoloji bizlerin doğru araştırmalar yapabilmemizi sağlayan, etkin bir metin yazabilmemiz için temellendirme ve değerlendirmeyi içeren bir bilim dalıdır. Özellikle bilimsel ve akademik bir çalışma yapmakla beraber, bu konuda makale yazmak için metodoloji yöntemi kullanılır. Günümüzdeki metodolojiler genelde Çağlayan ya da Helezonik modeli temel almaktadır.

Yazılım Süreç Modellerinde Süreç Tekrarı: Yazılım Süreç Modelleri tek bir defada uygulanmak yerine, birkaç tekrarda uygulanabilir. Örneğin: Geniş kapsamlı 6 alt sistemden oluşan bir sistemin; ilk alt sistemi için çağlayan modeli uygulandıktan sonra, geri kalanı için çağlayan modeli tekrar uygulanabilir. Böylece geliştirilme riskleri en az indirgenerek kazanılan deneyimlerden sistemin geri kalanı geliştirilirken faydalanılabilir. Hangi süreç modelinin, sistemin hangi bölümleri için ve kaç tekrarda uygulanacağına proje başında karar verilir. Süreç tekrarıyla yakından ilişkili iki geleneksel model vardır: Artırımsal ve spiral model.

**2.Birleşik Süreç (Unified Process):** Nesne tabanlı yazılım geliştirmek için var olan yöntemlerin kullanılmasıyla edinilen deneyimlerle bu süreçlerin en iyi özellikleri bir araya getirilerek bütünleştirilmiş bir yazılım geliştirme sürecidir. Yinelemeli, arttırmalı ve evrimsel aynı zamanda da risk güdümlüdür. Mimari merkezlidir. Birleşik süreçte yazılım geliştirme aşamaları şu şekildedir:

-Başlangıç: Vizyon kararı, fizibilite çalışması, tamam ya da devam kararı.

-Ayrıntılandırma: Daha gerçekçi çözümleme, çekirdek yapının ve yüksek riskli kısımların yinelemeli olarak oluşturulması.

-Tamamlama: Daha az riskli ve düşük öncelikli kısımların yinelemeli olarak gerçekleşmesi.

-Yayım: Beta testleri, piyasaya sürme çalışmaları.

Birleşik süreç modeli istikrarlı ve esnek çözümler üretir. Büyük ve riskli projelerde, yüksek kaliteli yazılımlarda vb kullanılabilir.

**3.Çevik Yazılım Geliştirme (Agile Software Development):** Çevik Model, her projenin farklı şekilde ele alınması gerektiğine ve mevcut yöntemlerin proje gereksinimlerine en uygun şekilde uyarlanması gerektiğine inanır. Çevik Yazılımda, bir sürüm için belirli özellikler sunmak için görevler küçük zaman dilimlerine bölünür. Yani yinelemeli bir yaklaşım benimsenir ve her yinelemeden sonra çalışan yazılım derlemesi teslim edilir. Her yapı, özellikler açısından artımlı; son yapı, müşterinin ihtiyaç duyduğu tüm özellikleri barındırır.

Çevik düşünce süreci, yazılım geliştirmede erken başladı. Esnekliği ve uygulanabilirliği nedeniyle zamanla popüler olmaya başladı. En popüler çevik yöntemler arasında Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (XP) (1996), Adaptive Software Development, Feature Driven Development ve Dynamic Systems Development Method (DSDM) (1995) yer alır. Bunlar, 2001 yılında Çevik Manifestosunun yayınlanmasından sonra Çevik Metodolojileri olarak adlandırıldılar. Çevik, uyarlanabilir yazılım geliştirme yöntemlerine dayanırken, şelale modeli gibi geleneksel SDLC modelleri, tahmine dayalı bir yaklaşıma dayanmaktadır. Geleneksel SDLC modellerindeki tahmine dayalı ekipler, genellikle ayrıntılı planlama ile çalışır ve önümüzdeki birkaç ay içinde veya ürün yaşam döngüsü sırasında teslim edilecek tam görev ve özellikler hakkında eksiksiz bir tahmine sahiptir. Tahmine dayalı yöntemler, tamamen döngünün başında yapılan gereksinim analizine ve planlamaya bağlıdır. Dahil edilecek herhangi bir değişiklik, sıkı bir değişiklik kontrol yönetimi ve önceliklendirme sürecinden geçer. Çevik, ayrıntılı bir planlamanın olmadığı ve yalnızca hangi özelliklerin geliştirilmesi gerektiğine ilişkin gelecekteki görevlerde netliğin olduğu durumlarda uyarlanabilir bir yaklaşım kullanır. Özellik odaklı geliştirme vardır ve ekip, değişen ürün gereksinimlerine dinamik olarak uyum sağlar. Ürün, sürüm yinelemeleri yoluyla çok sık test edilir ve gelecekte herhangi bir büyük arıza riskini en aza indirir. Müşteri Etkileşimi, bu Çevik metodolojinin belkemiğidir ve minimum dokümantasyonla açık iletişim, Çevik geliştirme ortamının tipik özellikleridir. Çevik ekipler birbirleriyle yakın iş birliği içinde çalışırlar ve çoğunlukla aynı coğrafi konumda bulunurlar. Bazı Çevik Metodolojilerinin geniş tanımı:

**-Extreme Programming (Uçdeğer Programlama) (XP):** En popüler çevik süreçlerinden biridir. XP ile oluşturulan çevik süreçte müşteri ve gereksinimlerin merkezde rol alırlar. Kökeni 1999’ da Çevik Manifestosunun yazarlarından biri olan Kent Beck’ e dayanır. XP’ nin temel değerleri: İletişim, Basitlik, Geri Bildirim ve Cesarettir.

**İletişim (Communication):** İletişim eksikliği, bilginin bir ekip içerisinde akmasını engeller. Çoğu zaman bir sorun olduğunda birileri onun nasıl çözüleceğini zaten bilir ancak iletişim eksikliğinden kaynaklanarak diğer grup üyelerinin sorunu öğrenmelerini veya çözüme katkıda bulunmalarını engeller. Böylece problem iki kez çözülerek atık üreterek sona erer.

**Basitlik (Simplicity):** Basitlik her zaman işe yarayan en basit şeyi yapmaya çalıştığınızı söyler. Basitlik zorunlu işlerin yapılmasıdır. Bir takım için basit olan, diğeri için tamamen takımın becerilerine, deneyimlerine ve bilgisine bağlı olarak karmaşıktır. Yani basitlik kişiden kişiye değişir.

**Geri Bildirim (Feedback):** Daha geleneksel yazılım geliştirme metodolojilerinde geri bildirim genellikle “çok az, çok geç olur”. Ancak XP değişimi benimser ve XP ekipleri erken ve sürekli geri bildirim almaya çalışırlar. Rotayı düzeltmeye ihtiyaç varsa, XP’ciler bunu mümkün olan en kısa sürede bilmek isterler. Bu süreçte müşteri proje grubunun bir üyesi haline gelir.

**Cesaret (Bravery):** Kent Beck, cesareti “korku karşısında etkili eylem” olarak tanımlar. Bir yazılım mühendisinin korkacak çok şeyi vardır dolayısıyla cesaret göstermek için birçok fırsat eline geçer. Gerçeği konuşmak ister. Geri bildirim vermek ve almak ister. XP başarısızlıktan korkmayı değil, başarısızlık durumunda onu en kısa zamanda telafi etmeyi önerir.

Yazılım geliştirmede kolaylığı ve esnekliği sağlamak için XP 12 farklı pratiği öngörür; Planlama Oyunu, Ekipte Müşteri, Önce Test, Basit Tasarım, Çiftli Programlama, Sürekli Entegrasyon, Kısa Aralıklı Sürümler, Yeniden Yapılandırma, Ortak Kod Sahiplenme, Metafor, Kodlama Standarttı, Haftada 40 Saat.

**-Scrum:** Scrum kelime olarak rugby oyununda oluşturulan küçük ekiplere verilen isimdir. Günümüzde çokça tercih edilen bir metottur. Jeff Sutjerland ve Ken Schawaber tarafından 1990’ ların ortalarında geliştirilen, proje yönetimi ve planlama ile ilgili yöntemlere odaklı olan ve mühendislik detayları içermeyen bir modeldir. Scrum’ ın sağladığı yararlardan biri de işleyiş sürecini şeffaf hale getirerek aksaklıkları ortaya çıkarır. Böylelikle proje ekibinin hataları fark ederek sürekli olarak iyileştirme yapmalarına olanak sağlar. Bu metodolojilerde bir yinelemenin tamamlanması 30 günden fazla sürmemekte ve günlük 15 dakikalık “dün ne yapıldı bugün ne yapılacak” toplantılarda sürekli iş takibi yapılmaktadır **(Scrum Daily Meeting).** Scrum’ ın genel olarak çalışma mantığına gelecek olursak, yapılacak iş **(Product Backlog**) parçalara bölünür. Daha sonra işin belirli bir parçası **(Sprint Backlog)** ele alınır ve 2-4 hafta süreyle bu işin tamamlanması planlanır. Genelde 4 hafta olan bu süreye **“Sprint”** denir. Scrum’ da 3 temel kavram vardır:

**1.Roller (Roles):**

-Product Owner (Ürün Sahibi)

-Scrum Master (Scrum Yöneticisi)

-Scrum Team (Scrum Takımı)

**2.Toplantılar (Meetings):**

-Sprint Planning (Sprint (Koşu) Planlama)

-Sprint Review (Sprint Gözden Geçirme)

- Daily Scrum (Günlük Scrum Toplantısı)

**3.Bileşenler/Araçlar (Artifacts):**

-Product Backlog (Ürün Gereksinim Dokümanı)

-Sprint Backlog (Sprint Dokümanı)

-Burndown Chart (Sprint Kalan Zaman Grafiği)

Scrum Örneği: Firma bir proje alır ve proje teslim tarihi belli değildir. Proje sahibinin beklentileri yüksek ancak istekleri yeterince net değil. Ayrıca süreçler içinde yeterli zaman bulunmamakta. Proje ekipleri arasında iletişim problemleri de var. Proje ise Kredi Kartı Taahhüt Programı. Sonunda iş birliği ile 8 ay olarak yer alan proje, 3 er haftalık 6 sprint ile 18 haftada tamamlanıyor. Günümüzde de Google, Microsoft, Nokia, IKEA hala Scrum ‘ı kullanmakta.

**Scrum günümüzde neden popüler?**

Scrum, her endüstride ve hatta genel olarak yaşamda proje yönetimini dönüştürme gücüne sahiptir. Scrum’ ı kullanarak, yolunuza çıkan kaçınılmaz değişime nasıl daha hızlı tepki vereceğinizi ve nasıl daha doğru tepki verileceğini keşfederek daha çevik oluruz. Odaklanmış kalarak, iş birliği yaparak ve iletişim kurarak gerçekten yapılması gereken işi başarıyla gerçekleştirilmiş olur. Uzun sözün kısası bu popülerliğin ve Scrum’ ın kullanılarak başarının sebebi ekibin sürekli iletişim içinde detaylı raporlaşarak düzenli organizasyonudur.

**KAYNAKÇA**

<https://medium.com/@omerharuncetin/yazılım-yaşam-döngü-modelleri-543c7879a742>

<https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>

<https://www.linkedin.com/pulse/yazılım-yaşam-döngüsü-nedir-veysel-ugur-kizmaz/>

<https://ybsansiklopedi.com/wp-content/uploads/2015/08/Yazılım-Geliştirme-Modelleri-Yazılım-Yaşam-DöngüsüSDLCYBS.pdf>

<https://medium.com/@batincetin44/yazılım-yaşam-döngü-modelleri-2342547d0840>

<https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696>

<https://www.zeolearn.com/magazine/why-scrum>

<https://www.clouddefense.ai/blog/system-development-life-cycle>

<https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/>

<https://devqa.io/software-development-life-cycle-sdlc-phases/>

<https://slideplayer.biz.tr/slide/12386328/>

<https://enprobilisim.com/yazilim-gelistirme-sureci-modelleri-sdmp/>

<https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>

<https://www.linkedin.com/pulse/code-fix-common-software-development-process-rafayel-mkrtchyan/>

<https://acikerisim.sakarya.edu.tr/bitstream/handle/20.500.12619/80049/T05405.pdf?sequence=1>

<https://www.techwell.com/2013/02/why-scrum-so-popular>

<https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm>

<http://www.bawiki.com/wiki/Unified-Process.html>

<https://slideplayer.biz.tr/slide/12237711/>

<https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm>

[https://www.digite.com/agile/extreme-programmhttps://www.quora.com/Why-is-the-Scrum-process-so-popular-in-the-software-industrying-xp/](https://www.digite.com/agile/extreme-programming-xp/)

**Doç. Dr. Deniz KILINÇ, Bakırçay Üniversitesi Yazılım Mühendisliği Temelleri Dersi 2. ve 3.hafta ders sunumları**

**İLETİŞİM BİLGİLERİM**

<https://www.linkedin.com/in/senanur-iriz-3b91b0223>

<https://github.com/iamsenanur>

<https://medium.com/@senairizz>

İZMİR BAKIRÇAY ÜNİVERSİTESİ

SENANUR İRİZ/ 210601011