**Yazılım Yaşam Döngüsü Nedir?**

**Yazılım yaşam döngüsü (Software Development Life Cycle) yazılımın hem oluşturulurken geçirilen aşamaların hem de yazılımı satın alan müşterinin deneyimi sürecinde geçen tüm olayların bütününe denir. Yazılım ürünü oluşturulurken bazı olaylar genel hatlarıyla ele alınırken bazıları da genel hatlarıyla incelenerek sistemleştirilir. Bu olaylar yaşanırken her zaman tek doğrultuda başarıyla ulaşılmaz bu sebeple bu süreçte sürekli olarak yazılım-üretici-müşteri arasında geri dönüşler (Feed Back) yaşanır. Bu dönüşler yazılımın üretimine olumlu yönde katkı sağlayarak ürünümüzün kusursuz olarak ortaya çıkmasına yol açar. Bu dönüşler yazılım oluşturma sürecinin döngüler halinde olmasını sağlar. Döngüler temel adımlardan oluşur. Bu döngüler sayesinde yazılımı oluşturmanın daha rahat olmasının yanı sıra üretim sürecindeki müşteri-yazılımcı arasındaki ekonomik işbirliğini sağlar. Döngülerin en önemli yararı ise proje oluşturma sürecindeki zaman yönetiminin en optimum seviyede yani gereksiz zaman israflarından uzak şekilde yapılmasıdır.**

**Yazılım yaşam döngüsünün temel adımlarını 5 maddede sıralayabiliriz:**

**1. Planlama Aşaması (Planning): İlk adımda oluşturulacak proje için müşteriden beklentileri ve gereksinimleri istenir ve gerekli zamanlama kurguları yapılır.**

**2. Analiz Aşaması (Analysis): Bu adımdaki temel amaç yazılımcı tarafından bir önceki aşamada oluşturulan plandaki çözümün incelenip hangi ölçüde yarar sağlayıp sağlamadığını ve müşteri gereksinimlerini karşılayıp karşılamadığını araştırmaktır. Projede yapılacak çözümün ince ayrıntılarıyla ele alınılmasıdır. Detaylı çözümleme yapılır. Her şey artılarıyla eksileriyle ele alınır ve buna göre dokümantasyon çalışmaları yapılır. Müşteri-yazılımcı arasında istenilen ürün belirginleşir.**

**3. Tasarım Aşaması (Design): İhtiyaçlar net bir şekilde belirlendikten sonra başlaması gereken adımlardır. Bu aşamada projeyle ilgili tasarımlar yapılır ve tasarımlar genel hatlarıyla ikiye ayrılır.**

**3.1 Düşünsel Tasarım: Yapılacak sistemin tasarımı anlatılır.**

**3.2 Fiziksel Tasarım: Yazılımı içeren parçalara ve öğelerin ayrıntılarına değinilir.**

**4. Üretim (Production): Planlama, analiz ve tasarım aşamaları bittikten sonra yapılacak işlemler detaylı olarak belirlenip projenin geliştirme aşamasıdır. Bu noktadan sonra herhangi bir plan veya analiz işlemi yapılmamalı ve plan dahili içerisinde ilerlenmelidir.**

**5.** **Gerçekleştirme Aşaması (Implementation): Kod oluşum süreci ve test edilişi aşamalarından sonra yazılımın kurulumunun yapıp işlevsel hale getirildiği aşamadır.**

**6. Bakım Aşaması(Maintenance): Bu aşama proje bittikten sonraki aşamadır. Yazılım işlevsel hale geldikten sonra yazılımın ömrü boyunca yazılımı oluşturan mühendis tarafından yazılıma eklenen her türlü iyileştirme, hataları çözme ve yazılımın eksikliklerini giderme sürecidir.**

**Yazılım Yaşam Döngüsü Modelleri**

**Yazılımın yaşam döngüsü sürecinde yazılımın oluşturulma safhasında geçirilen tüm aşamalar ana mantık dizilimi aynı olmak suretiyle bazı modellere işlenmiştir. Ortaya çıkan modeller kod geliştirme sürecinde hangi sırayla ilerlenmesi gerektiğini hangi düzende sürecin uygulanacağı konusunda etkin rol oynar. Modeller zaman içerisinde yazılım mühendisleri tarafından işlevsellik mantığıyla ortaya çıkarılmıştır. Çıktıkları dönemin donanım teknolojileriyle birlikte ihtiyaçlar çerçevesinde evrimleşmişlerdir. Yani bu modeller yazılım ürünü oluşturulurken yazılımcıya en hızlı en ekonomik ve en rahat şekilde yazılım oluşturması yönünde rehberlik sağlar. Yazılım oluşturma tecrübelerine göre bu modellerin ortaya koyduğu somut deliller gösterir ki bu aşamaları doğru takip etmek daha kaliteli ürün çıkartma konusunda etkilidir.**

**Bu modeller iki temel kısma ayrılmıştır:**

1. **Geleneksel Yazılım Süreç Modelleri:**

**Gelişigüzel Model:**

**Eski ve kişisel bir yöntemdir. Belirli bir yöntem ya da model yoktur. 1960’lı yıllarda sıklıkla kullanılmıştır. Tek kişilik üretim ortamlarında görülen, basit programlar için kodlama aşamasında kullanılır. Programcı için iyi bir model-kaynak olamaz çünkü geriye baktığında hangi aşamalarda ilerlediğini çözümlemeyebilir. Bu nedenle ürünün bakım aşaması kaynağa göre ilerlenemediğinden yapılamaz. Günümüzde kullanılmaz.**

**Barok Model:**

**Gelişi güzel modelin ardından 1970’li yıllarda ortaya çıkmıştır. Bu döngüde temel adımları doğrusal bir biçimde ilerletmek amaçlanır. Bu modelde dokümantasyon işlemi ayrı bir olay olarak ele alınmıştır. Belgeleme aşamasını yazılım geliştirme aşamasının ardından yapılır. Günümüzdeki yazılan kodlardaki modellemelerde belgeleme her aşamada yazılır bu daha işlevseldir. Bu modelin eksiği de tek aşamalı belgeleme olmasıdır. Çünkü her adımda belgeleme yapımı kodlama ardından ortaya çıkabilecek hataların çözümünde anlık olarak giderilmesidir. Günümüzde basit programlar dışında tercih edilmemektedir.**

**Çağlayan Yaşam-Döngü Modeli( Şelale-Waterfell):**

**Barok modeli gibi 1970’li yıllarda ortaya çıkmıştır. Yazılımcı-müşteri arasındaki gereksinimler net şekilde belirlenmiş olup kısa vadeli çok karmaşık olmayan projeler için yararlı bir modeldir. Barok modelinin tam zıttı olarak belgeleme bir aşamada değil her aşamada vardır böylece bakım sürecinde problem çözümü daha belirgin ve efektif şekilde olur. Her aşama bir önceki aşama başlamadan başlayamaz. Geri dönüş belirsizlikleri bu modelde yoktur. Ancak eğer bir hata varsa ancak projenin gerçekleştirme aşamasında belli olur bu sebeple çok fazla zaman kaybına ve ekonomik çöküntüye yol açar. Hem müşteri için hem de yazılımcı için efektif olmadığı ve geçerli kayıplara yol açtığı için tercih edilmez.**

**V Süreç Modeli:**

**Bu modeli çağlayan yaşam-döngü modelinin gerekli güncellemelerinin alıp uyarlandığı model olarak ele alabiliriz. Şelale modeline ek olarak aynı adımların her birine özel deneme, sınama aşaması eklenmiştir. Bu sebeple eğer bir hata varsa o aşama sonrasında hemen hatanın tespitini yapabilip ekonomik ve zaman kayıplarına yol açmadan projeyi doğru adımlarla ilerletebiliriz. Yani hata kaynaklarına ulaşmak kolaydır. Bu modeli V harfine benzer şekilde bir kolunu üretim olarak diğer kolunu ise sınama kolu olarak kullanırız. V modeli esas olarak 3 aşamadan oluşur: Kullanıcı modeli, mimari model, gerçekleştirim modeli. Şelale modelinde olduğu gibi karmaşanın ve belirsizliklerin az olduğu gereksinimlerin ne olduğunun tam belirgin olduğu projelerde V süreç modeli uygun model olarak ele alınabilir.**

**Helezonik (Spiral) Model:**

**Helezonik modeli diğer modellere göre farklı kılan en önemli özellik risk analizini fark yaratabilecek düzeyde ön planda tutmasıdır. Risk analizinin ön planda olmasının sunduğu avantaj ise kod yazılımında herhangi bir hata olursa erken tanı ile çözüme hızlı bir şekilde ulaşılır. Ayrıca sonuca ulaşmak amaçlı her aşamada oluşturulan prototipler sayesinde yazılımcı eğer bir problem varsa her aşamada görme imkanı olduğu için problemi hızlıca çözer. Helezonik model temel aşamadan oluşur:**

**Planlama: Her aşama sonucunda çıkan ara ürünler için geleceğe yönelik işlevsel odaklı planlamalar yapılır. Risk Analizi: Projede çıkabilecek herhangi bir soruna karşı riskler araştırılıp belirleme çalışması yapılır Üretim: Ara ürünler ayrı birer parça gibi üretilir. Kullanıcı Değerlendirme: Oluşturulan ara üründen sonra kullanıcıdan ürüne göre geri dönüt alınır. Helezonik sistemin problemleri ise küçük basit algoritmalı riski düşük projeler için fazla pahalı bir sistemdir. Karmaşık bir içeriye sahiptir kullanıcı veya yazılımcı geriye baktığı zaman fazla dokümantasyondan dolayı karmaşıklık yaşayabilir. Uzun süren bir modeldir.**

**Arttırımlı Geliştirme Süreç Modeli:**

**Yapılan projeyi tek seferde teslim etmek yerine bakım sürecinde olduğu gibi sürekli destek veren modeldir. Bu modeli kullanan yazılım mühendisi her seferinde yeni işlevler ekler veya yazılan kodlardaki eksikleri belirleyerek çözüm üretir. Yazılım kullanıcı istekleri ve gereksinimlerine göre parça bütün ilişkisiyle parça parça olmak üzere yeni sürümler eklenerek devam eder. Bir tarafta ürün kullanıcı tarafından işlevini yerine getirirken bir yandan da yazılımcı tarafından kod ekleme olayıyla devam eder. Gelişim durmaz devamlıdır. Kod yazımı uzun sürse bile uzun vadeli projelerde kullanılabilecek uygun bir projedir. Ürünün diğer bir avantajı ise ürün kullanılırken aktif olarak kullanıcının olumlu veya olumsuz dönütlerle yazılım mühendisini aktif ve anlık olarak bilgilendirmesidir. Bu sebeple yazılımcı her şeyi görür ve ona göre en iyi çözümü üreterek hatasız kod dizilimine ulaşır. İlk teslim edilen projeyi prototip gibi görebiliriz. İlk prototipe göre yazılımcı gerekli desteği verir. Karmaşık kodlar için parça bütün ilişkisi güttüğü için ideal kodlama modelidir. Günümüzde sıklıkla tercih edilir. Güncelleme adı altında birçok yenilik sunar.**

**Araştırma tabanlı süreç:**

**Deneyimle kullan mantığıyla çalışan çokça tercih edilmeyen bir modeldir. Tamamen belirsizliklerin bulunduğu projelerde aktif olarak kullanılır. Sonuçlar net değildir dolayısıyla maliyet hesabı konusunda net fikir belirlenemez. Günümüzde kullanıldığı alanlar belirsizliklerden dolayı oldukça kısıtlıdır.**

**Evrimsel Geliştirme Süreç Modeli:**

**Müşterilerin isteklerinin tam olarak belirlenemediği veya anlaşılamadığı zamanlarda kullanılır. Müşteri ile bir araya gelip yapılan görüşmelerle birlikte taslak oluşturulur. İlk olarak kafamızda oluşturduğumuz en net planla birlikte projeye başlanılır. Müşteri tarafından diğer talep edilen gereksinimler ve istekler kod yazım sürecine dahil edilir. Böylece ilk prototip oluşturulur. Projenin modele bağlı başarısı bu ilk prototipe bağlıdır. Model sayesinde bu aşamalarda kullanıcının ne istediği zamana yayılarak tam olarak algılanır. Böylelikle hata oranı azalır ve ürün sürekli kontrol edilir. Fakat sürekli bir değişim kod ekleme çıkarma olaylardan kod diziliminde veya üründe hata olabilir.**

**Kodla ve Düzelt Yaşam Döngü Modeli:**

**Küçük yapıdaki karmaşık olmayan projeler için kullanılır. Belgeleme olmadığı için bakımı çok zordur. Direkt ürün gerçekleştirilir. Büyük projeler için kullanılmaz.**

**2. Çevik Modeller:**

**Çevik yazılım: Günümüz dünyasında gelişen son teknolojiyle birlikte kullanıcıların nitelik olarak daha yüksek yazılımlara daha kısa sürede daha az maliyetle ulaşmak istediği için ‘çevik’ olarak gelişen yazılım modelleridir. Bu modellerinn temel amacı yazılımcıların grup projeleriyle birlikte kodlanmak istenen projeye problemlerden ve hatalardan uzak bir şekilde ulaşmak istemesidir. Bu yöntemler günümüzde modern şirketlerde mühendislerin grup halinde motivasyonları yüksek, hatalardan olduğunca uzak bir biçimde çalışmasını sağlar. Bu da mühendislerin işe odaklanmalarını arttırarak projenin daha verimli olmasını sağlar. Bazı çevik yazılım geliştirme modelleri şunlardır:**

**Extreme Programming**

**En bilinen çevik modellerden birisidir. Model üzerinden ilerlenen projedeki en önemli amaç müşteri gereksinimlerini odak noktası olarak belirlemektir. Proje başlamadan belirlenen gereksinimlere göre belgeleme işlemi yapılır. Bu gereksinimlere göre projede ilerleme kaydedilir. Proje oluşum sürecinde maliyet sebebiyle gereksinimlere göre değişiklik yapılmamaya çalışılır. 4 temel maddeden oluşur.**

**1. Basitlik: Yapılan projede yazılan kodun sade, açık ve anlaşılabilir olmasıdır. Gereksiz kod dizilimlerinden ve bakım sürecinde geri dönüşlerde anlaşılamayacak kodlardan uzak durulması gerekir. Ayrı olarak bu maddenin diğer amacı da gereksiz belgeleme işleminden uzak durulmasıdır.**

**2. Cesaret: Proje sürecinde cesur olmayı amaçlayan maddedir. Gereken durumlarda zor kararlar alma konusunda yoğunlaşır. İşin içinden çıkılmadığı durumlarda projeyi baştan almayı sağlar.**

**3. Geri Dönüş: Proje müşteriye teslim edildikten sonra eksikleri ve hataları dönüt olarak alıp düzeltmemizi amaçlar. Ürünün geleceği açısından yenileme kodları da önemlidir.**

**4. İletişim: Birkaç yazılım mühendisinin birlikte oluşturduğu projelerin en önemli hususu iletişimdir. Projenin her basamağında aktif iletişime sahip olunmalı. İletişim hem projenin daha kusursuz olmasını sağlar hem de hızlı biteceği için maliyet anlamında müşteri-yazılımcı arasındaki problemleri çözer.**

**SCRUM**

**Scrum modeli büyük projeler için büyük ekip çalışmalarında kullanılır. İletişim çok önemlidir. Model projeyi belli başlıklar altında parçalar. Bu parçaların her birine “sprint” adı verilir. Ekip arasında her gün iletişim toplantıları oluşturulur. Scrum modelinde 3 temel kavram vardır. Bunları şöyle sıralayabiliriz:**

**1. Roller: Yazılım sahibi, yazılım mühendisleri ve bu mühendislerin ekip yöneticisinden oluşur. Yazılım sahibi projeyi tasarlayan kişi olmakla birlikte ekibin beynidir. Scrum yöneticisi ise scrum takımı olan yazılım mühendislerini etik çerçevesinde belli kurallara göre yazılımın geleceğini denetlemekle görevlidir.**

**2. Toplantılar: Scrum modelinin en önemli parçasıdır. Sürekli olarak ekibin bir araya gelerek aktif iletişim içerisinde kalmalarını sağlar. Aktif iletişim projedeki hataları yok eder ve ürünün geleceği için gereklidir.**

**3. Araçlar: Gereksinimler belgelenir. Proje boyunca ihtiyaç olacak bileşenler ve yapılması gerekenler yazılır. Sprintlere göre planlamalar yapılır ve bu planlamalar sadece takımdakiler tarafından değiştirilir.**

**Scrum günümüzde en çok tercih edilen projedir. Bunun geçerli sebepleri vardır. Tüm modeller içinde evrimini yazılımcı ihtiyaçlarına göre tamamlamakta olan en işlevsel modeldir. Bir projenin en önemli 2 unsuru zaman ve buna bağlı olarak ekonomik kaygılardır. Scrum modeli yazılımcılar arasında günümüz yazılım şirketlerinde oluşturulan projelerde iletişimin ve planlamanın en iyi olduğu modeldir. Yüksek teknolojilerle oluşturulan projelere rahatlıkta uyum sağlar. Net olarak müşterinin isteği bilinmeyen projelerde karmaşanın içinden bu model ile çıkılabilir. Sürekli kendini yenileyen dönemin şartlarına adapte olabilen bir modeldir. Proje sonrası müşteriyle sürekli iletişim içinde kalınır ve yenilemelerde bulunulur. Hata ve eksikler giderilir. Grup çalışması için idealdir. Parçalara bölüp projeyi ilerletmesi projenin en iyi şekilde olması için çok önemlidir. Yazılımcıların günümüzde vazgeçilmezidir.**

**Tüm yazılım geliştirme modellerinin gelişimi için ne yapılabilir?**

**Doğa nasıl evrimleşiyorsa yazılım da aynı şekilde müşterilerin yani insanın ihtiyaçlarına göre evrimleşir. Yazılımın doğru bir şekilde doğru zamanlamada iyi bir planla oluşturulması gerekir. Bu yüzden planlamanın ana mantığı olarak yazılım geliştirme modelleri kullanılır. Bu modellerin gelişimindeki en önemli faktör ihtiyaçlara göre yazılımcıların kurdukları ihtiyaç iletişimidir. Zamanla gereksinimlere göre modeller evrimleşir ve en işlevsel halini almaya çalışır. Özellikle şirket altında çalışan yazılımcılar içindir.**

**Hangi projede hangi modeli kullanmalıyız ve hangi yazılım geliştirme modeli en idealidir?**

**Gelişigüzel ve barok modelleri artık eski ve işe yararlılığı düşük olduğu için görevlerini yitirdi. Bunun en büyük sebebi gerekli aşamalarının olmaması ve dokümantasyon işleminin yetersiz kalmasıdır. Şelale modeli basit ve karmaşası olmayan modeller için kullanılabilir çünkü iş bölümü detaylıdır. Kullanılmamasının en büyük sebebi ise iletişimin projenin geleceğini tehlikeye atacak derecede zayıf olmasıdır. Değişimlere kapalıdır, aşamaları değişmez. Şelale modelinin üst versiyonu olarak V modeli tercih edilir. V modelinin kullanımı kolaydır ve planlama daha detaylıdır. Ancak ürün teslim sonrası bakım aşamalarının yetersizliğinden dolayı günümüz şartlarında nadiren tercih edilmektedir. Spiral model ise karmaşık projelerde dokümantasyonun yani belgeleme işleminin çok olduğu projelerde kullanılır. Büyük projeler için kullanışlıdır çünkü hatalar anlık olarak fark edilip çözümlenebilir. Artımsal geliştirme modelinde ise müşteri ilişkileri kuvvetli olduğu için istenilen koda açıkça ulaşılır. Test edilme tekrarı fazla olduğu için tercih edilebilir. Kodla ve düzelt modeli basit programlarda kullanılabilir. Günümüzde önemli ve büyük projelerde genel olarak çevik modeller tercih edilir çünkü bu modellerin ana unsurları aktif iletişim, kusursuz kodlama ve ekonomik sebeplere dayalı verimli zamanlama bulundurmasıdır. Bu çevik modeller dönemin şartlarına ve yazılımcının ihtiyaçlarına göre geliştirebilir ve değişime olumlu yönde her zaman açıktır. Çok kişinin çalıştığı projelerde görev dağılımı açısından işlevselliğin en yüksek seviyede olduğu için önemlidir. Bakım süreci sürekli devam eder projelerde emeklilik durumu yoktur. Yazılımcı tarafından ihtiyaca göre belirlenen güncellemeler aktif olarak kullanıcıya sunulur. Büyük projeler için diğer bir model olan evrimsel geliştirme metodu da kullanılabilir.**

**Emirhan Soylu-210601070**

**Github Hesabı:** **https://github.com/soyluemir**

**Medium Hesabı:** **https://medium.com/@soyluuemir**

**Linkedin Hesabı:** **https://www.linkedin.com/in/emirhan-soylu-150b05228/**

**KAYNAKÇA**

[**https://iskulubu.com/yazilim/yazilim-gelistirme-yasam-dongusu/**](https://iskulubu.com/yazilim/yazilim-gelistirme-yasam-dongusu/)

[**https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742**](https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742)

**Doç Dr. Deniz Kılınç, Bakırçay Üniversitesi Yazılım Mühendisliğine Giriş ders sunumları**

[**https://tr.linkedin.com/pulse/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-nedir-veysel-ugur-kizmaz**](https://tr.linkedin.com/pulse/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-nedir-veysel-ugur-kizmaz)

[**https://osmanozaydin.com/yazilim-yasam-dongusu-ve-agile-yazilim-gelistirme/**](https://osmanozaydin.com/yazilim-yasam-dongusu-ve-agile-yazilim-gelistirme/)