**Yazılım Geliştirme Yaşam Döngüsü (SDLC)**

Büyük ve karmaşık sistemler geliştiren organizasyonların ürün geliştirme süreçlerini sürekli tanımlayıp iyileştirmek gerekmektedir. Kalitesiz ürün geliştiren organizasyonlar itibar, müşteri ve gelir kaybı riski ile karşı karşıya kalabilirler ve hatta organizasyonun geleceğini bile tehlikeye atabilirler. Bu nedenle yazılım geliştirme yaşam döngüsü modellerini proje geliştirme sürecinde rehber edinmemiz çok önemlidir.

Yazılım geliştirme yaşam döngüsü (SDLC), geliştirilecek bir projenin öngörülen bütçeyi aşmadan istenen gereksinimleri karşılayarak zamanında teslim edilebilmesi için projenin planlanmasından itibaren teslim edilme aşamasına kadar adım adım takip edilmesi gereken safha ve bu safhalardan oluşan döngüdür. Yazılım geliştirme sürecinde en sık karşılaşılan sorunlardan biri yazılım geliştiricisinin tecrübesizliğinden kaynaklanan gereksinimlerin doğru tespit edilememesi diğeri ise geliştirme süreci boyunca değişen ve gelişen gereksinimlerdir. Bu durum değişen hedef problemler (moving target problems) olarak literatüre geçmiştir. Müşteri gereksinimlerinde veya iş sahasında herhangi bir değişiklik olması halinde tasarım aşamasına geçilmeden önce değişiklik yapılması istenebilir. Bu noktada eğer yazılım geliştirme yaşam döngüsü doğrusalsa ve geriye dönüşe izin vermiyorsa müşterinin gereksinimlerini karşılamak zorlaşabilir. Bu tarz problemler ve gereksinimlerin dinamik olduğu düşünüldüğünde söz konusu aşamalar bir döngü olarak ele alınmalıdır. Bunun yanı sıra geliştirilecek olan bir proje için yazılım geliştirme yaşam döngüsü temel aşamalarının iyi planlanması ve her adımın etkili bir şekilde uygulanması projenin geleceği açısından büyük önem arz etmektedir. Bu temel aşamalar şu şekildedir:

1. **Gereksinim Safhası (Requirements Phase):** Bir diğer adıyla planlama aşaması yazılım yaşam döngüsünün ilk adımıdır. Bu safhada problem tanımlanmaya ve iş anlaşılmaya çalışılır. Problem anlaşıldıktan sonra “Müşteri ne istiyor, neye ihtiyacı var?” gibi sorular doğrultusunda müşteri gereksinimleri belirlenir. Gereksinimler belirlendikten sonra fizibilite çalışması yapılır yani proje için hem teknik hem de finansal araştırmalar yapılır. Fizibilite çalışması yapılırken olası risk faktörleri sistematik olarak hesaplanarak projenin hedefe ulaşıp ulaşmayacağı, kar getirip getirmeyeceği finansal ve başarım açısından öngörülmeye çalışılır. Fizibilite çalışmasının tamamlanmasıyla projeye dair planlamalar yapılır ve süreç başlamış olur.
2. **Analiz Safhası (Analysis Phase):** Problem daha net bir şekilde tanımlanır ve tüm detaylar ortaya çıkarılır. Gereksinim safhasında belirlenen gereksinimler detaylı bir şekilde analiz edilir ve projede nelerin istenildiğiyle ilgili çalışmalar yapılır. Analiz çalışmaları sırasında müşteri ve yazılım ekibi interaktif bir şekilde iletişimde olmalıdır ki alınan kararlar her iki taraf açısından da tatmin edici olsun. Tüm bu çalışmalar tamamlandıktan sonra iki tarafça da anlaşma sağlanırsa analiz sonuçları şartname dokümanına (specification document) yazılır. Ayrıca bu safhada yazılım proje yönetim planı (Software Project Management Plan) da çıkarılır ve bu planda proje genelinde yapılacaklar kapsamlı bir şekilde ele alınır.
3. **Tasarım Safhası (Design Phase):** Gereksinimler analiz aşamasında net bir şekilde belirlendikten sonra projeyi kodlama ve test etme aşamasına geçmeden önceki son aşamada projenin tasarımı yapılır. Geliştirilecek olan projenin tasarımı yapılırken müşteri ve sistem gereksinimleri karşılanmak üzere sistem mimarisi, özellikleri, yetenekleri ve arayüzleri belirlenir. Yazılım tasarımında başvurulacak en önemli tekniklerden birisi soyutlamadır (abstraction). Soyutlama problemlerin çözümlerini kolaylaştırmak için nesnelerin, olayların ve durumların bazı özelliklerinin gereksiz kısımlarının görmezden gelinmesidir. Bu teknik problemleri basitleştirip gereksiz detaylardan arındırarak önemli kısımlarına odaklanmamızı sağlar. Diğer bir teknik ise modellemedir. Modelleme, tasarlanmak istenen sistemin prototipini önceden görmemizi sağlar. Genel olarak statik ve dinamik modellerden bahsetmek mümkündür. Statik modelde program sırasında değişmeyen yönler ele alınırken dinamik modelde programın çalışması sırasındaki işleyiş ele alınmaktadır. Tüm bu teknikler göz önünde bulundurulduğunda yazılım yaşam döngüsü boyuna iki çeşit yazılım tasarımı yapılabilir. Bunlardan birisi mimari tasarım (architectural design) diğeri ise detaylı tasarımdır (detailed design). Mimari tasarımda geliştirilecek sistem mimarisi oluşturulur ve sistem modüllere ayrılır. Sistem modüllerinin genel yapıları ve organizasyon içindeki etkileşimleri incelenir. Mimari tasarım yapıldıktan sonra tasarım dokümantasyonu oluşturulur. Oluşturulan tasarım dokümanının amacı, yazılım geliştiricinin referans alacağı ve proje sürecinde veya sonrasında dahil olacak yeni yazılımcıların projeyi daha kolay anlayabilmesini sağlayacak teknik bilgilerin yer almasıdır. Detaylı tasarımda ise genel olarak mimari tasarım dokümantasyonu revize edilir. Revizyon tamamlandıktan sonra ayrıntılı tasarımda kullanılacak algoritmalar (algorithms), mimari desenler (architectural patterns), veri tabanı mimarisi (database architecture), sınıflar (class) ve sınıflar arasındaki ilişkiler belirlenir.
4. **Gerçekleştirme Safhası (Implementation Phase):** Tasarımın belli bir olgunluğa ulaşmasıyla birlikte bir önceki aşamada oluşturulan her bir modül için kod yazılır ve her bir modül kodlaması tamamlandığında ayrı ayrı test edilir. Bu test etme aşaması gerek kodlama süresince gerekse kodlama sonrasında gerçekleştirilen bir diğer önemli adımdır. Yazılım ürününün bir bütün olarak ele alınması gerekmektedir. Bu yüzden modüllerin kodlanması ve testi tamamlandıktan sonra tüm modüller birleştirilir fakat bu esnada birtakım sorun ve hatalar ortaya çıkabilir. Bu yanlışlıkların giderilmesi adına entegrasyon testi (integration testing) yapılır. Entegrasyon testinin yapılmasının ardından şartname dokümanı temel alınarak müşteri gereksinimlerinin karşılanıp karşılanmadığına yönelik ürün testi (product testing) yapılır. Ardından gerekli kurulumun yapılmasıyla müşteri gerçek verilerle (data) sistemi test eder. Buna kabul testi (acceptance testing) denir. Eğer bu noktada herhangi bir sorun yoksa bundan sonra artık geliştirilen ürün için bakım safhası (maintenance phase) başlayacaktır.
5. **Teslim Sonrası Bakım Safhası (Post-delivery Maintenance Phase):** Geliştirilen yazılım ürününün gerekli testlerinin gerçekleştirme (implementation) aşamasında tamamlanmasının ardından sahaya teslim edilebilecek bir versiyonu çıkartılır ve teslimle birlikte bakım aşaması da başlar. Teslim sonrası bakım aşaması sistemde herhangi bir değişikliğin yapıldığı aşamadır. Bu bakım 2 türde yapılabilir. Bunlardan bir tanesi düzeltici bakım (corrective maintenance) diğeri ise özellik arttırmadır (enhancement/ software update). Düzeltici bakım geliştirilen yazılım projesindeki hataları düzeltmek (software repair) için yapılan bakımdır. Özellilerin arttırılması için bir diğer deyişle yazılım güncellemesi (software update), yazılım şartname dokümanında yapılan değişiklikleri ve değişikliklerin mevcut yazılıma uyarlanmasını içerir. Bu uyarlamanın da iki çeşidi vardır. Bunlardan ilki kullanıcıların düşünceleri doğrultusunda yazılım ürününün yararlılığını arttırmak üzere yapılan mükemmelleştirici bakımdır (perfective maintenance). Bu tür bakıma geliştirilen projeye müşteri yararına yeni bir özellik eklemek örnek verilebilir. Diğer bakım türü ise ürünün çalışma ortamında meydana gelen değişikliklere karşı yapılan uyarlanabilir bakımdır (adaptive maintenance). Bu tür bakıma projenin farklı bir işletim sistemi veya farklı donanım üzerinde çalıştırılması örnek olarak gösterilebilir.

Yazılım geliştirme sürekli olarak gelişen ve değişen bir gerçeklik modelidir. Bunun için yazılım geliştirme sürecinde veya teslim yapıldıktan sonra yazılım hataları veya eksiklikleriyle karşılaşma olasılığı yüksektir. Bu hataların yazılıma maliyeti farklı aşamalarda farklı şekillerde yansıyacağı için hatalar olabildiğince erken tespit edilmeye çalışılmalıdır. Yazılım geliştirme yaşam döngüsü safhalarından herhangi birinde meydana gelebilecek bir hata düşünüldüğünde maliyet tablosu aşağıdaki gibi olmaktadır:

**Teslim sonrası bakım > Gerçekleştirme > Tasarım > Analiz > Gereksinim**

Yazılım geliştirme yaşam döngüsünde ilk safhalarda yapılan çalışmalar kağıt üzerinde olduğu için hataları düzeltmek daha ucuza mal olacaktır. Ancak hataların son safhalarda tespit edilmesi dokümantasyon ve kod üzerinde değişiklik yapmayı gerektirir. Bunun yanı sıra bakımı yapılan yazılım ürününün tekrar test edilmesi ve müşteriye tekrar kurulması gerekir. Tüm bunlar bakımın maliyetinin daha fazla olmasına sebep olmaktadır. Geliştirilen büyük ölçekli projelerde tespit edilen hataların %60 ve %70 oranında gereksinim, analiz ve tasarım aşamalarında meydana geldiği görülmüştür. Yani yazılım geliştirme yaşam döngüsünde gereksinimlerin müşteri izleniminde iyi tespit edilip analizin doğru yapılması geliştirilen projenin geleceği açısından büyük önem taşırken; hataların en erken dönemde bulunması bakım maliyetini minimalize etmektedir.

**Yazılım Geliştirme Yaşam Döngüsü Modelleri (SDLC models),**

Bir proje geliştirmek sadece kodlama aşamasından ibaret değildir. Kodlama yapmaya başlamadan önce hangi yazılım dilinin kullanılacağına karar verilmesi gerekir. Özellikle karmaşık ve büyük sistemler geliştirirken kullanılması elzem olan metodolojiler; sistemin kodlama sürecinin verimli geçmesini sağlayacak ve ortaya bir ürün çıkmasını ciddi anlamda kolaylaştıracaktır.

1. **Barok Model:** 1970’li yıllarda ortaya çıkan bu model yazılım yaşam döngüsünün temel adımlarını doğrusal bir şekilde gerçekleştirmeyi öngörmektedir. Barok modelin en önemli özelliği ise dokümantasyon olgusunu ayrı bir süreç olarak ele almasıdır. Bu modele göre belgeleme süreci yazılım geliştirme döngüsünün aşamaları tamamlandıktan sonra ele alınmalıdır. Günümüzde popüler olarak kullanılan dillere bakıldığında yazılım geliştirme sürecinde en önemli adımlardan bir tanesi dokümantasyonu her aşamada gerektiği kadar yapmaktır. Çünkü proje geliştirirken müşteri veya sistem gereksinimlerimlerinin güncellenmesi veya eklenmesi gereken yeni bir özellik olması halinde en önce başvurulacak kaynak yazılım geliştirme sürecinin aşamalarında ele alınacak olan dokümanlardır.Fakat

barok modelinde dokümantasyon sürecinin ayrı bir safha olarak düşünülmesi bu metotun zayıf yönüdür.

Ayrıca bu modelin safhalar arasındaki geri dönüşler hakkında bilgi vermemesi de zayıf yönlerinden bir tanesidir.

1. **Şelale/ Çağlayan Modeli (Waterfall Methodology):** Geleneksel yazılım geliştirme metodolojisi olarak da bilinen şelale modeli en eski ve en bilinen yazılım geliştirme metotlarından biridir. Bu model uzun yıllar birçok hükümet tarafından kamuya özel yazılım geliştirme projelerinde kullanılmıştır. Fakat son yıllarda kullanımı azalmıştır. Şelale modeli yazılım geliştirme sürecinde temel felsefe, proje sürecindeki bir aşamayı bitirmeden diğer aşamaya geçmemek ve her safhanın sonunda dokümantasyon yapmak üzerine kuruludur. Bu yaklaşımda her aşamanın hiçbir eksik olmadan tamamlanması ve bir sonraki aşamaya geçildiğinde bir daha geriye dönülmemesi tavsiye edilir. Yani şelale modelinin tam verimle çalışması için her aşamanın mükemmele yakın performans göstermesi beklenir ki böyle bir ihtimalden söz etmek çoğu yazılım projesi için pek mümkün değildir. Özellikle büyük ve uzun zaman gerektiren projeler için şelale modelini kullanmak çok risklidir. Çünkü şelale modeli kısa sürede tamamlanabilecek küçük projeler için elzem bir metodolojidir. Bu tarz projelerde ise en önemli safhalar gereksinim analizi ve tasarım aşamalarıdır. Analiz aşamasında mümkün olan tüm detayın tasarıma yansıtılabilmesi için müşteri ve sistem gereksinimlerinin en ince ayrıntısına kadar belirlenmesi gerekir. Tasarım aşamasında geliştirilebilecek yazılımın tüm gereksinimlerin karşılanacağı şekilde detaylı bir çalışma yapılması gerekmektedir. Bu nedenle şelale modelinin kullanıldığı projelerde, proje ekibi en fazla zamanı analiz ve tasarım aşamasında harcamaktadır. Tüm bu efor ve detaylı çalışmalara rağmen uzun zamana yayılan projelerde gereksinimlerin değişmesi kaçınılmazdır. Kodlama, test veya bakım aşamasında ise olabilecek değişikliklerin sisteme/ yazılıma yansıtılma maliyeti ise çok yüksektir. Gerçek yaşamdaki yazılım projeleri genelde yineleme gerektirir. Genelde yazılımın kullanıcıya ulaşma zamanı uzundur. Gereksinim tanımlamaları çoğu kez net bir şekilde yapılmadığından dolayı, yanlışların düzeltilme ve eksikliklerin giderilme maliyetleri yüksektir. Yazılım üretim ekipleri bir an önce kodlama, çalıştırma ve sonucu görme eğiliminde olduklarından bu model ile yapılan üretimlerde ekip mutsuzlaşmakta ve kod yazma dışında kalan ekibe önem verilmemektedir. Tüm bu olguların yanında şelale modeli gereksinim ve tasarım net bir şekilde belirlendiği için daha çok askeri vb. analiz ve tasarımın çok önemli olduğu, hata yapmanın maliyetinin çok yüksek olduğu kritik projelerde kullanılır.

**Şelale Modelinin Avantajları:**

1. Kullanımı ve anlaşılması basittir.
2. Projenin safhaları ayrı olduğundan iş bölümü ve iş planı projenin en başında net bir şekilde bellidir. Bu durum projenin yönetimini oldukça kolaylaştırır.
3. Analiz ve tasarım aşamasında gereksinimlerin ve tasarımın net bir şekilde ortaya konulmasından dolayı kodlama ve test aşamaları çok kısa sürmektedir. Bu da ileriki aşamalarda çıkan hata sayısını minimuma indirir.

**Şelale Modelinin Dezavantajları:**

1. Karmaşık ve nesne yönelimli projeler için uygun değildir.
2. Devam eden ve uzun projeler için zayıftır.
3. Projede oluşabilecek her türlü değişime elverişsiz ve katı bir modeldir. Yapılan her değişiklik maliyeti büyük oranda artırır.
4. Müşteri memnuniyeti çok zordur çünkü değişim ve gelişime açık bir model değildir. Ayrıca müşteri sadece ilk aşamalarda sürece dahildir.
5. Model safhalardan oluştuğu için ürün son safhada tamamlanır. Gereksinimlerin iyi tanımlanmadığı müşterinin ne istediğinin anlaşılmadığı bir projede bu durum proje bittikten sonra projenin beğenilememesine hatta iptal edilmesine bile neden olmaktadır.
6. **V Süreç Model (V process Model):** V süreç modeli şelale modelinin gelişmiş halidir. Bu modelde sol tarafta üretim sağ tarafta ise o safhanın sınaması yapılır. Her bir üretim aşamasının karşısında bulunan sınama aşamaları sayesinde doğrusal ilerlemek yerine, süreç adımları kodlama evresinden sonra yukarıya doğru eğilir ve tipik V şeklini alır. Bu yüzden hata kaynaklarına dönüş daha kolaydır. V süreç modelinden çıktıları 3 temel başlık altında toplayabiliriz. Bunlar “Kullanıcı modeli”, “Mimari model” ve “Gerçekleştirim Modeli” dir. Kullanıcı modelinde geliştirme sürecinin kullanıcıyla olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları yapılmaktadır. Mimari modelde sistem tasarımı ve oluşacak alt sistemle tüm sistemin sınama işlemlerine ilişkin işlevler dikkate alınmaktadır. Gerçekleştirim modelinde ise yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar ele alınmaktadır. Bu süreç modeli şelale modelinde olduğu gibi belirsizliklerin az, gereksinim ve iş tanımlarının net olduğu bilişim teknolojileri (BT) gibi projeler için uygun bir modeldir. Ayrıca V süreç modeli, kullanıcının projeye katkısını artırmaktadır. Kullanıcı yazılım projesinin geliştirilmesi süresince projeye dahildir.

**V Süreç Modelinin Avantajları:**

1. Verification ve validation (sınama) planları erken aşamalardan itibaren uygulanır.
2. Verification ve validation, teslim edilebilir ürüne de uygulanabilir.
3. Proje yönetimi tarafında takibi ve kullanımı kolaydır.

**V Süreç Modelinin Dezavantajları:**

1. Aynı anda gerçekleştirilmesi gereken olaylara kolay imkan tanımaz.
2. Fazlar arası iterasyon (yineleme) yoktur.
3. Risk yönetim planı yoktur.
4. Yazılım da sistemler gibi zamanla evrimleşir.
5. Geliştirme devam ettikçe iş ve ürün gereksinimleri gelişir ve değişir.
6. Son ürüne ulaşma düz bir çizgi ile ifade edilemez.
7. **Helezonik Model (Spiral Model):** 1998’de Barry Boehm tarafından risk odaklı bir yazılım geliştirme süreci olarak önerilmiştir. Bu modelde yazılım geliştirme süreci, bir aktiviteden diğerine birtakım geri dönüşleri takip etmek yerine spiral olarak temsil edilir. Bu süreç modeli şelale modelinde göz önünde bulundurulmayan olası değişiklikleri proje risklerinden biri olduğunu varsayar ve bu risk faktörlerini ortadan kaldırmak için risk yönetim faaliyetlerini içerir. Spiraldeki her döngü bir fazı ifade eder ve planlama, risk analizi, üretim ve kullanıcı değerlendirmesi olmak üzere 4 sektöre ayrılmıştır. Geliştirilecek projenin her bir aşaması için planlama, risk analizi, üretim ve kullanıcı değerlendirmesi yapılır.
8. **Planlama:** Geliştirilecek prototip için plan yapılmasını, amaçların belirlenmesini ve bir önceki adımda üretilen prototipe entegrasyon yapılmasını sağlar.
9. **Risk Analizi:** Risk seçeneklerinin araştırılması ve risklerin belirlenmesini sağlar.
10. **Üretim:** Prototiplerin geliştirilmesini içerir. Her aşamanın sonunda bir prototip oluşturulur ve bu prototipler geliştirilerek kullanıma hazır son prototip ortaya çıkartılmış olunur.
11. **Kullanıcı Değerlendirmesi:** Prototiple ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmeleri ele alınır.

**Helezonik Metodun Avantajları:**

1. En önemli avantajı, üretim süreci boyunca prototiplerin geliştirilmesi ve geliştirilen prototiplerin kullanıcı tarafından sınanmasıdır. Bu durum yazılımı kullanacak personelin sürece erken katılmasını ve ileride oluşabilecek istenmeyen durumların ortadan kaldırılmasını sağlar.
2. Gerek proje sahibinin gerekse yüklenici tarafındaki yöneticilerin proje boyunca çalışan prototip yazılımlarla karşılaşması projeyi daha kolay takip edebilmeleri ve hak ediş planlaması yapabilmelerini sağlar.
3. Yazılımın kodlanması ve sınanması daha erken başlar.
4. Proje küçük parçalara bölünür, en riskli kısımlar önce gerçekleştirilir.
5. Pek çok yazılım modelini bünyesinde barındırır.
6. Riske duyarlı yaklaşımı potansiyel zorlukları engeller.

**Helezonik Metodun Dezavantajları:**

1. Küçük ve düşük riskli projeler maliyetli bir metottur.
2. Komplextir ve spiral sonsuza gidebilir.
3. Ara adımların fazlalığı nedeniyle fazla dokümantasyon gerektirir.
4. Öznel risk değerlendirme deneyimine dayalıdır.
5. **Evrimsel Geliştirme Süreç Modeli (Evolutionary Development Process Model):**

Evrimsel geliştirme süreç modeli ilk tam ölçekli modeldir. Coğrafi olarak geniş alana yayılmış, çok birimli bankacılık gibi projelerde kullanım için önerilen bir modeldir. Bu süreç modelinde sistem, zaman içinde kazanılan anlayışa göre sürümler şeklinde geliştirilir. Her aşamada üretilen ürünler, üretildikleri alan için tam işlevselliği içermektedirler. Müşteriden elde edilen gereksinimlerle projenin geliştirilmesine başlanır ve ilk sürüm üretilir. Müşteri tarafından gelen taleplere göre sisteme yeni özellikler eklenerek yeni sürümler elde edilir. Modelin başarısı geçirdiği ilk evrimin başarısına bağlıdır.

**Evrimsel Geliştirme Süreç Modeli Avantajları:**

1. Kullanıcıların kendi gereksinimleri daha iyi anlamalarını sağlar.
2. Sürekli değerlendirme erken aşamalardaki geliştirme risklerini azaltır.
3. Projenin geliştirilmesi süresince ortaya çıkabilecek riskleri ve hataları azaltır.

**Evrimsel Geliştirme Süreç Modeli Dezavantajları:**

1. Sürecin görünürlüğü azdır (düzenli teslim edilebilir ürün yoktur).
2. Sistemler sıklıkla iyi yapılandırılamaz (sürekli değişiklik sistemin yapısına zarar verir).
3. Bakımı zordur.
4. Yazılım gereksinimlerini yenilemek gerekebilir.
5. **Artımlı Geliştirme Süreç Modeli (Incremental Development Process Model):** Artımlı geliştirme süreç modeli, şelale yaşam-döngü modeli ile evrimsel geliştirme süreç modelini temel alır ve şelale modeli unsularını yinelemeli bir şekilde kullanır. Bir yazılım ürünü geliştirilirken sürekli olarak değişen müşteri gereksinimlerini karşılayan ve ortaya çıkan kaçınılmaz hataların zamanında düzeltilmesini sağlayan bir süreç modelidir. Bu modelde sistemi tek seferde teslim etmek yerine geliştirme ve teslim parçalara bölünür. Her teslim beklenen işlevselliğin bir parçasını karşılar. Müşteri gereksinimleri önemine göre önceliklendirilir ve öncelikli gereksinimler bağımlılıklarına göre sıralanarak her yinelemede bu gereksinimlerin bir kısmı tamamlanır. Bir parçanın geliştirmesi başladığında gereksinimler dondurulur. Olası değişiklikler sonraki teslimlerde ele alınır. Üretilen her teslim sürümü birbirini kapsayacak ve giderek artan sayıda işlev içerecek şekilde geliştirilir.

**Artımlı Süreç Modelini Avantajları:**

1. Sistem için gerekli olan gereksinimler müşteriyle birlikte belirlenir.
2. Gereksinimlerin önemine göre teslim edilecek artımlar belirlenir.
3. Öncelikle en önemli gereksinimleri karşılayan çekirdek sistem geliştirilir.
4. Erken artımlar prototip gibi davranarak gereksinimlerin daha iyi anlaşılmasını sağlar.
5. Tüm projenin başarısız olma riskini azaltır.
6. En önemli sistem özellikleri daha fazla sınanma (test edilme) imkanı bulur.
7. Divide & conquer yaklaşımıdır.

**Artımlı Süreç Modelinin Dezavantajları:**

1. Artımların tanımlanması için tüm sistemin tanımlanmasına ihtiyaç vardır.
2. Gereksinimleri doğru boyuttaki artımlara atamak bazen zor olabilir.
3. Tecrübeli bir ekip gerektirir.
4. Artımların kendi içinde tekrarlamasına izin vermez.
5. **Kodla ve Düzelt Yazılım Yaşam Döngü Modeli (Code and Fix Life-Cycle Model):** Günümüzde birçok yazılım firmasının yazılım geliştirirken tercih ettiği yazılım geliştirme yaşam döngüsü modellerinden biridir. Başlangıçta 100-200 satır koddan oluşan projelerin geliştirilmesi için düşünülmüş olsa da günümüz de her türlü yazılım geliştirme faaliyeti için kullanılmaktadır. Yazılım ürünü gereksinim, analiz ve tasarım safhalarının herhangi biri dikkate alınmaksızın direkt olarak kodlama ve test işlemine tabi tutulur. Böylece ilk safhada yazılım ürününün ilk sürümü ortaya çıkartılır. Yazılım ürünü en son istenilen şekle gelinceye kadar devamlı geliştirilir. Bu yaşam-döngü modelinde bakım safhası vardır fakat sisteme ait dokümantasyon yapılmadığı için bakım yapmak çok zordur. Kodla ve düzelt yaşam-döngü modeli yazılım geliştirmenin en kolay yolu ancak en pahalı yöntemidir. Kodlama safhasından sonra bir yazılım ürünü içerisindeki değişikliklerin maliyetini düşünüldüğünde bu maliyet çok yüksektir. Buna ilave olarak şartname ve tasarım dokümanı olmaksızın ürünün bakımını yapmak fazlasıyla zordur. Yazılım geliştirmek kolay olduğu için ne yazık ki birçok yazılım projesinde bu model kullanılmaktadır.

**Kodla ve Düzelt Yaşam Döngü Modelinin Avantaları:**

1. Herhangi bir planlamaya ihtiyaç duymaz.
2. Çok küçük projelerde ya da kısa ömürlü prototiplerde uygulanabilir.
3. Program aşamaları çabuk geçilir.
4. Uzman görüşüne ihtiyaç düşüktür, herkes bu modeli kullanabilir.

**Kodla ve Düzelt Yaşam Döngü Modelinin Dezavantajları:**

1. Kontrollü değildir.
2. Kaynak planlaması yoktur.
3. Bitiş süresi belli değildir.
4. Hataların bulunması ve doğrulanması zordur.
5. Gerçekleştirme aşamasında (implementation phase) yazılan kodları düzeltmek maliyetli olabilir.
6. Kodlar kullanıcının ihtiyacını karşılamayabilir.
7. Kodlar sonradan değiştirilmek üzere planlanmadığından esnek değildir, değiştirilmesi zordur.
8. **Prototipleme Yaşam-Döngü Modeli (Prototyping):** Gereksinimler hızlıca toplanarak işe başlanılır. Geliştiriciler ve kullanıcılar aynı masa etrafında buluşarak yazılımdan elde edilecek bütün çıktılara, bu çıktılar için gerekli girdilerin nasıl sağlanacağına, nasıl korunacağına, hangi işlemlere tabii tutulacağına karar verirler. Daha sonra hızlıca yapılan bir tasarımla yazılımın kullanıcıya yansıyacak yönünü aktaran bir ilk örnek üretilir. Prototip, kullanıcının kullanımı ve değerlendirmesine sunulur. Bu değerlendirmelere bakılarak ilk örnek üzerinde gerekli değişiklikler yapılır. Prototipin yeni hali kullanıcı tarafından yeniden değerlendirilir. Böylece kullanıcının istediği yazılıma iyice yaklaşılmış bir ilk örnek üzerinde yazılımın neler yapacağı konusunda kullanıcı ile anlaşmaya varılır. Doğrusal modelin döngüsel versiyonudur. Bu modelde gereksinim analizi ve prototipleme için tasarım yapıldıktan sonra geliştirme süreci başlatılır. Prototipleme yaratıldıktan sonra müşteriye değerlendirme için verilir. Müşteri paketi test eder ve düşüncelerini ürünü müşterinin tam beklentilerine göre düzenleyen geliştiriciye iletir. Sınırlı sayıdaki yinelemelerden sonra son yazılım paketi müşteriye verilir. Bu metodolojide, yazılım müşteri ve geliştirici arasında periyodik bilgi gidip gelmeleri sonucunda gelişir.

**Prototipleme Metodolojisinin Avantajları:**

1. Kullanıcı sistem gereksinimlerini görebilir.
2. Karmaşa ve sistem yanılgılarını engeller.
3. Yeni ve beklenmeyen gereksinimler netleştirilebilir.
4. Risk kontrolü sağlanır.

**Prototipleme Metodolojisinin Dezavantajları:**

1. Belgelendirmesi olmayan hızlı ve kirli prototipler oluşmasına neden olur.
2. Prototip hedefleri net değilse kod hackleme ya da jenga başlar.
3. Müşteri prototipten de son ürün gibi performans ve görünüm bekler.

Modeller birbiriyle karşılaştırıldığında süreç modellerinin aslında birbirinden pek de farklı olmadığı görülür ve bu modeller çoğu zaman birlikte kullanılırlar. Yazılım süreç modelleri yazılımın türüne ve kullanım alanına göre seçilmelidir. Yazılım süreç modelinin seçimi için oluşabilecek riskleri tolare edebilme kapasitesi, projenin büyüklüğü, karmaşıklığı, gerçekleştirecek kurumun yapısı, zaman, maliyet gibi kriterlere dikkat edilmelidir. Günümüzde en çok kullanılan model agile modeldir. Kısa sürede esnek bir şekilde geliştirilerek yüksek başarımla sonuçlanan bir model olduğundan en çok tercih edilen modeldir.

**Çevik Yazılım Geliştirme Metodolojileri (Agile Software Development):** Yazılım geliştirme süreci sıkıntılı ve uzun süren bir dönemdir ve bu sektörde yazılım sürümlerinin zamanında ortaya çıkarılamaması, değişiklik isteklerine zamanında cevap verilememesi, yazılım hatalarının geç fark edilmesi ve zaman içerisinde gelen isteklere göre sistemin kendi yapısını geliştirememesi gibi çeşitli sorunlar ortaya çıkmıştır. Bunun yanında yazılım projelerinde üretilen özellik ve fonksiyonların yaklaşık %64’ü neredeyse hiç kullanılmamaktadır. Bu sorunların aşılmasına yönelik çalışmalar sonucu 1990’lı yılların sonuna doğru çevik (agile) olarak isimlendirilen metotlar geliştirilmiştir. Çevik yazılım geliştirme metotları; piyasaya çok çabuk ürün çıkarabilme, değişen isteklere hızlı yanıt verme ve en kısa sürede bir yazılım ürününü müşteri hizmetine sunmayı amaçlamaktadır. Çevik yazılım geliştirme metotları, verimliliği yüksek, esnek, hata oranı düşük, hızlı ve ucuz çözümler sağlamaktadırlar. Çevik yazılım geliştirme metotları, kendi içerisinde özü aynı fakat pratikleri farklılaşan çeşitli metodolojilere ayrılmaktadır. Bu metodolojilerden biri scrum metodudur.

**Scrum:** Agile yaklaşım altındaki birçok teknik arasından en çok tercih edilen yazılım geliştirme tekniğidir. Yazılım geliştirme ekibinin elinde komplex bir problem olduğunda ve bu problem belli kural setlerine bağlı kalmak suretiyle çözülmek istendiğinde süreci kontrollü bir şekilde yöneterek piyasaya ürünü en kısa ve ucuz yoldan üretmemize sağlar.Scrumda projeniz,bağımsız olarak ele alınan küçük döngülere (sprint) ayrılır. Bu sprintlerin her biri yaklaşık 2-4 hafta sürer ve projenizi hızlı bir şekilde geliştirmenize ve teslim tarihlerinize zamanında ulaşmanıza yardımcı olur. Her sprint tamamlandıktan sonra size geri bildirimlerini veren paydaşlara (product owner, müşteri, kullanıcılar vb.) sunulur. Ardından bir sonraki sprinte geçmeden önce onların önerilerini uygularsınız. Bu yöntem önce ürünün tamamını geliştirip sonra müşteriye sunulan klasik metodolojideki (waterfall) bir proje yönetimi sürecinden oldukça farklıdır. Bunun yanı sıra scrum, müşteri isteklerine hızlı cevap verebilen, esnek, kalitesi yüksek çıktılar üretebilmesine ve scrum ekibinin kendi kendine organize olmasına imkan tanıyan bir yöntemdir.

**Scrumda Temel Kavramlar**

1. **Roller(Roles)**
2. **Ürün sahibi (Product Owner):** Projenin iş değeri açısından geri dönüşü ile sorumludur. Ekibin bir parçasıdır, müşteri tarafından görevlendirilmiştir, detayları takip eder, feedbackler verir.
3. **Scrum Yöneticisi (Scrum Master):** Takımın scrumın temel değerlerine, pratiklerine ve kurallarına bağlı kalmasını garanti altına alır. Takımı ve organizasyonu scruma adapte eder. Takımın dış etkilerden korunmasını ve sadece kendi işine yoğunlaşarak üretkenliğinin artmasından sorumludur.
4. **Scrum Takımı (Scrum Team):** Scrm takımı, sürekli iletişim halinde olan ve tek bir hedefe ulaşmak için müdahale edilen kişilerden oluşur. Gereksinimlerin süre tahminini yaparlar. Bu takım 5-9 kişiden oluşur. Sprinte başlarken hedefi seçip çalışma sonucunu belirlerler. Sprint hedefine ulaşmak için proje sınırları dahilinde her şeyi yapmakta serbesttirler. Kendi kendilerini organize ederler. Çalışma sonuçlarını belli aralıklar ile ürün sahibine (product owner) gösterirler.
5. **Toplantılar (Meetings):**
6. **Koşu Planlama ( Sprint Planning):** Geniş kapsamlı gereksinim listesinin çıkarılması, başarılı geliştirme için uygun dağıtım gereksinimlerinin belirlenmesi, dağıtımlar için gereksinimlerin eşleştirilmenin yapılması, dağıtımlar için takımların belirlenmesi, risk değerlendirmesi ve risk kontrollerinin belirlenmesi, gözden geçirmeler ve olası gereksinim değişikliklerinin belirlenmesi, geliştirme araçları ve altyapısının onaylanması, dağıtım, geliştirme ve pazarlama maliyetlerinin hesaplanması, yönetimi ve destekleri gözden geçirme ve onaylama gibi işlemler bu planlama toplantısı sırasında gerçekleştirilir.
7. **Koşu Gözden Geçirme (Sprint Review):** Her sprint (koşu) başlangıcında planlama toplantısı yapılır. Toplantının ilk kısmında ürün sahibi ile takım ürün gereksinim listesini gözden geçirir ve gereksinim elemanlarını hedeflerini ve içeriklerini belirler. Toplantını ikinci kısmında takım üyeleri ürün gereksinim listesinin en üstünden başlayarak bu sprintte gerçekleştirilecek gereksinimlerden oluşan sprint gereksinim listesini oluştururlar. Bu scrumın anahtar uygulamasıdır. Takım, ürün sahibi tarafından belirtilen önceliklendirilmiş gereksinimlerden ne kadarını yapacağını belirler ve taahhüt eder.
8. **Günlük Scrum Toplantısı (Daily Scrum Meeting):** Sprint başladıktan sonra takım sürecin başka bir anahtar aktivitesi olan günlük scrum toplantılarını gerçekleştirir. Bu toplantılar her iş gününde genelde belirlenen sabah saatlerinde tüm takımın katılımıyla kısa süreli olmak suretiyle ayakta gerçekleştirilir. Takımın ilerleyişini ve karşılaştıkları engelleri görmek için önemli bir fırsattır. Teker teker tüm ekip üyeleri şu soruların cevaplarını verir;

-Dün ne yaptım?

-Bugün ne yapacağım?

-Önümde olan engeller ve karşılaştığım sorunlar neler?

Scrum Master, toplantı esnasında not tutar ve sorun yaşayan üyelere toplantıdan sonra yardımcı olur.

1. **Bileşenler/ Araçlar (Artiacts)**
2. **Ürün gereksinim Dokümanı (Product Backlog):** Proje boyunca yapılması gereken iş elemanlarının basit bir listesidir. Ürün gereksinim listesi canlı bir dokümandır, geçerli ve kullanışlı olması için devamlı bakım gerektirir. Birçok yeni eleman zamanla eklenebilir, birçok eleman daha küçük parçalara ayrılabilir, bazı gereksinimlerin artık gereksiz veya anlamsız olduğu fark edilip silinebilir. Listedeki dokümanlar her gün değişiklik gösterebilir. Ürün gereksinim listesi genellikle kullanıcı hikayelerinden oluşur ve kullanıcı bakış açısından bakılır.
3. **Koşu Dokümanı (Sprint Backlog):** Sprint backlog, mevcut sprint için product backlogdan iş ve görevleri kapsar. Sprint backlogdaki işlerin amacı, sprint sonunda son ürünün bir parçası olan bir işlevselliği ya da çalışabilir bir parçayı elde etmektir. Sprint backlog sadece takım tarafından değiştirilmektedir.
4. **Sprint Kalan Zaman Grafiği (Burndown Chart):** Bu grafik iterasyon (sprint) boyunca işlerin ne kadarının yapıldığı ile normalde ne kadar yapılması gerektiğinin karşılaştırılabilmesini sağlar. Bir iterasyonun toplam 100 saatten ve 20 günden oluştuğu farz edilirse her gün 5 saatlik iş yapılması gerekir. Takım elemanları her gün ne kadarlık iş gerçekleştirdikleri bilgisini girerler.

**Scrumı Neden Tercih Etmeliyiz?**

Scrum yaklaşımlarda, geliştirme ekibi tarafından yapılan işler ardışık olarak değil aynı anda gerçekleştirilir. Development team geliştirmeye başlamadan önce klasik yaklaşımdaki gibi ürün gereksinimlerini hatta tasarımın eksiksiz bir şekilde tamamlanmış olmasını beklemez ve hemen geliştirmeye başlarlar. Projenin ömrü boyunca ve sonrasında bile her şey esnek ve değiştirilebilirdir. Aynı şey tüm kodlama tamamlandıktan sonra sadece sonunda değil sürekli olarak yapılan test kodları için de geçerlidir. Ayrıca scrum, takımları deneyim yoluyla öğrenmeye, bir problem üzerinde çalışırken kendi kendine organize olmaya ve sürekli gelişmek için kazançlar ve kayıplar üzerinde düşünmeye teşvik eder. Bu sayede takıma adapte olma yeteneği kazandırır. Takım üyelerinin işine daha çok sahip çıkmasını sağlar ve üretkenliği maksimuma çıkarır.

**REFERANSLAR:**

<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/@tunaytoksoz/yazilim-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-sdlc-ve-modelleri%CC%87-c3fe40f6e4e8>  
<https://medium.com/architectural-patterns/yaz%C4%B1l%C4%B1m-geli%C5%9Ftirme-modelleri-62915545c51e>  
<https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>  
[https://tr.wikipedia.org/wiki/V-Model\_(Yaz%C4%B1l%C4%B1m\_geli%C5%9Ftirme)](https://tr.wikipedia.org/wiki/V-Model_%28Yaz%C4%B1l%C4%B1m_geli%C5%9Ftirme%29)  
<https://enprobilisim.com/yazilim-gelistirme-sureci-modelleri-sdmp/>  
<https://batuhanakpunar.medium.com/b%C3%B6l%C3%BCm-1-scrum-nedir-neden-scrum-b00b48f25e54#:~:text=Peki%20neden%20Scrum%3F&text=Scrum%2C%20tak%C4%B1mlar%C4%B1%20deneyimler%20yoluyla%20%C3%B6%C4%9Frenmeye,%E2%80%9Cadapte%20olabilme%E2%80%9D%20yetene%C4%9Fi%20kazand%C4%B1r%C4%B1r>.

**Medium:** <https://medium.com/@ceng.betulkircil/yaz%C4%B1l%C4%B1m-geli%C5%9Ftirme-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-sdlc-3a0f1105bf0d>

**Linkedin:** <https://www.linkedin.com/in/bet%C3%BCl-k%C4%B1r%C3%A7%C4%B1l-2b97991b7/>

**Github:** <https://github.com/BetullKircil>

Betül Kırçıl

200601017