**Yazılım Yaşam Döngü Modelleri**

Yazılım varlığını belirli bir süre devam ettiren kavramsal bir bütünlüktür, bu açıdan bakarsak yazılım yaşam döngü sürecinin varlığından da bahsedebiliriz. Yazılım süreci bir maliyet gerektirir yine bu maliyetin kontrol ve takibinde yazılım yaşam döngü süreçlerine ihtiyacımız olacaktır. Yazılımda temel kabul edeceğimiz basamaklar bulunur; bunlar gereksinimlerin toplanması, gereksinimlerin analizi, uygulama (kodlama aşaması), test aşaması (kodda var olan sorunların çözülmesi) aşamalarından oluşur. Aşamaların sonucunda yazılım ürün sahibi yeni gereksinim oluşturulmasını isteyebilir; yeni gereksinim ve var olan kodda bazı şeylerin düzeltilmesine yönelik eylemlerin yapıldığı aşamaya da bakım aşaması denir. Biz yazılımcılar bu aşamaları uygularken belli bir akışa ihtiyaç duyarız.

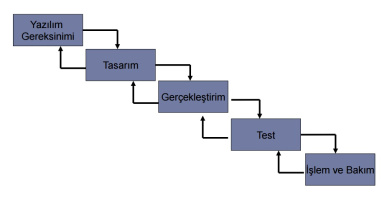
Yazılım gereksinimlerin tespiti, analiz, uygulama, bakım, emeklilik gibi doğal süreçler içerir. Bu temel adımların ve de yazılımın amacına yönelik deneyimlenmiş faktörlerin içerilmesiyle günümüzde yoğun olarak kullanılan yazılım yaşam döngü süreçleri tanımlanmıştır. Yazımızda yazılım yaşam döngü süreçlerinin ilk modelinden günümüze kadar olan değişimini inceleyeceğiz.

**Yap ve Düzelt Modeli**

Genellikle sektöre yeni atılmış yazılımcıların kullandığı bir modeldir; yazılımcı direkt sonuç almayı istiyordur. Yap ve düzelt modelinde yazılımda olması gereken temel unsurların hepsi yer almaz; müşterinin ihtiyaçlarına yönelik yazılım geliştirilir, müşterinin istekleri arttıkça veya müşteri yapılan kısmı beğenmediğinde uygulamaya geri dönülür. İşin planı yapılmaz direk uygulamaya geçilir. Bu model (kısa projeler hariç) uzun süreç gerektirmesi yönünden elverişsizdir.

**Çağlayan Model (Waterfall Model)**

Çağlayan model yaklaşımında öne sürdüğümüz temel adımlar sırasıyla gerçekleştirilir; herhangi bir adımda sorun çıkması halinde o adıma geri dönülerek düzeltilir. Genel kapsamıyla yazılım projelerinin ilerleyişi için kullanılması temel alınmış bir modeldir.Gereksinim tespitini tasarım sürecinin takip etmesi yönünden; “Yap düzelt modele” göre daha komplekstir.

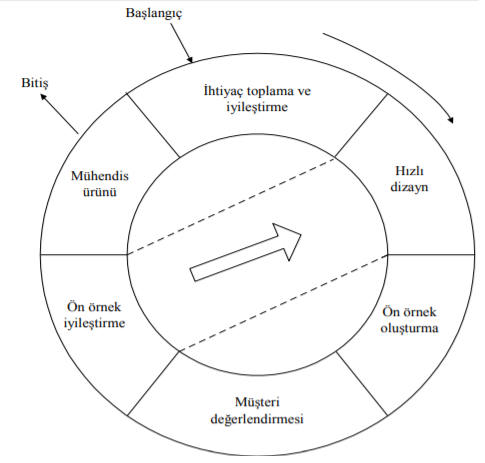
Çağlayan modelin eksik yönlerini tespit etmek istersek; yazılım sürecinde iken müşteri ile olan iletişim azlığını öne sürebiliriz. Uygulama esnasında müşteri ile etkileşimde kalınmadığı için somut müşteri geri dönütleri ancak proje tamamlandığında alınabilmektedir. 

**Şekil 1.1** çağlayan model

Bu durumda ilerleyişte yeniden düzenlenmesi istenen bir nokta varsa süreç dışında fark edilip sürece tekrar dönülmesini gerektirir. Bu durumda istenilen kısmın önceden fark edilmesinden daha maliyetli olur.

**Ön Örnek Geliştirme Modeli**

Ön örnek geliştirme modeli yazılımda istenilen gereksinimlerin tam olarak anlaşılmadığı durumlarda ve kullanılacak işletim isteminin kararlaştırılamaması durumlarında tercih edilir. Gereksinimlerin tespiti yapıldıktan sonra hızlı bir tasarım aşaması yapılır. Geliştirilmesi istenen yazılımın bir örneği yapılır. Bu örnek kullanıcıya sunulup istenilenlerin tam olarak anlaşılıp anlaşılmadığına karar verildikten sonra asıl yazılıma geçirilmesi sürecidir. Oluşturulan örnek çalışan bir yazılım ise gerekli düzenlenmeler yapılıp müşteriye teslim edilir.

Bu modelde de bazı durumlarda sorunlarla karşılaşılabilmektedir. Bunlardan biri müşterinin “ön örnek” yaratma kavramının neden gerekli olduğunu bilmemesidir. Müşteri genel olarak sonuç odaklı beklentidedir. Diğer bir karşılaşılan problem ise müşterinin teslimatı daha kısa bir zamanda istediğinde oluşabilen bir durumdur, kısa bir zamanda teslimatın yapılabilmesi için iyi bilinen fakat uygun olmayan bir işletim sistemi veya algoritma seçilebilir; bu seçimler de yazılımın ilerleyen safhalarında problemlere sebep olabilir. Bu model ile çağlayan modeli karşılaştırırsak aradaki en belirgin fark gereksinimlerin tam olarak anlaşılmaması 

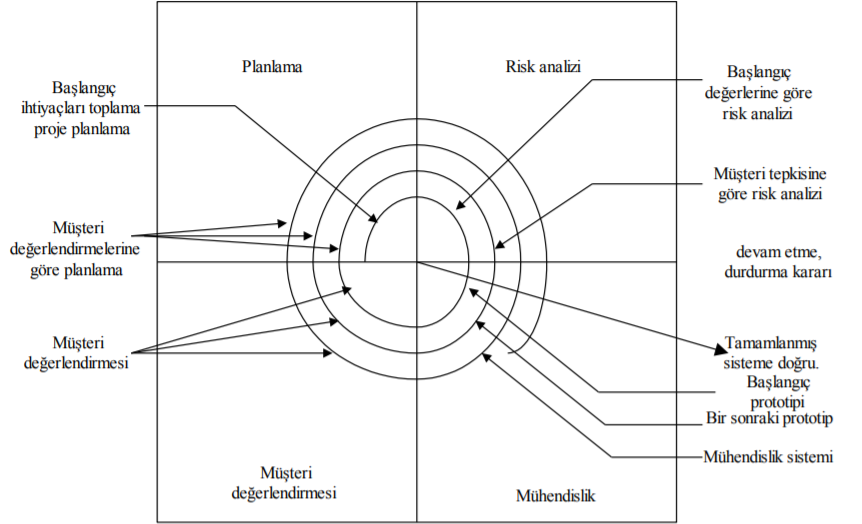
**Şekil 1.2** Ön Örnek Model

ve çözüm olarak yazılımın bir ön modelini oluşturmaya yönelinmesidir.

**Sarmal Model (Spiral Model)**

Sarmal model daha önce tanımladığımız çağlayan model ve ön örnek modelin en iyi yönlerinin harmanlanması ve bunun yanında risk analizinin de eklenmesiyle problemin çözümüne yönelik en etkili modellerden birini oluşturur.

Bu model genellikle büyük ölçekli projelerde tercih edilir. Bu tarz projelerde tercih edilmesinin nedeni temel yazılım süreçlerini adım adım gerçekleştirirken her adımda projenin ilerleyişinin müşterinin değerlendirmesine sunulması ve her adımda olası risk faktörlerinin önceden belirlenmesi ile birlikte hatayı en aza indirgenecek şekilde projenin yürütülmesidir.



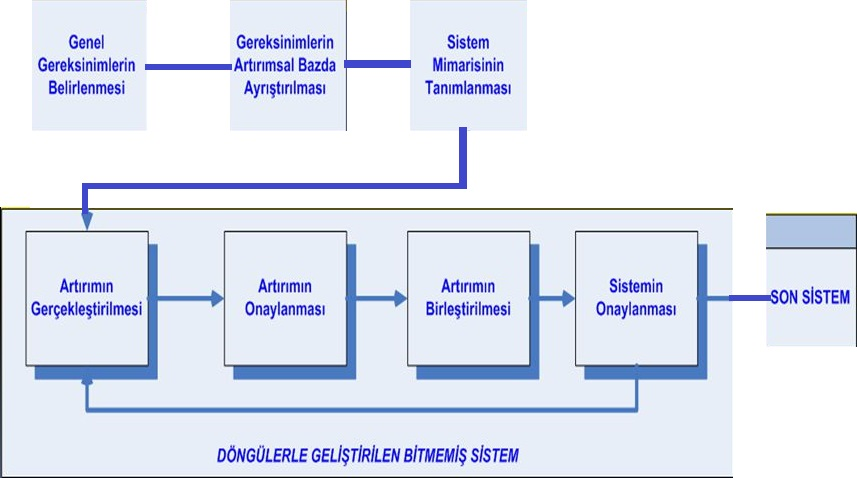
**Şekil 1.3** Sarmal Model

Sarmal model içerdiği risk analizi uygulamasıyla, bu uygulamayı gerçekleştirebilecek kişilere ihtiyaç duyar. Eğer oluşabilecek riskleri önceden belirleyebilen bir kişi varlığı yoksa bu model gerçekleştirilemeyecektir. Bunun yanı sıra model uzun bir süreç ve adımlar dizisinden oluştuğu için müşteri tarafından kabul edilebilmesi sıkıntılı olabilir.

**Artışlı Model (Incremental Model)**

Artışlı model genel kapsamıyla çağlayan modele benzerlik gösterir; bu modeli çağlayan modelden ayıran unsur yazılım geliştiricilerinin öncelikle müşterinin öncelikli gereksinimlerini baz alıp uygulamaya koymasıdır. Bu öncelik gerektiren fonksiyonların her biri bir yapı olarak ele alınır. Her bir yapı bitiminde yeni yapıya geçildiği gibi önceki yapının da bakımı yapılır. Bu durum müşterilerin de istediği bir yaklaşımdır. Müşteri bu sayede daha az önemli olan fonksiyonların uygulanması için beklemek zorunda kalmaz.

Bu modelin olumsuz kısımlarını değerlendirmek istersek; modelin her proje için uygun olmadığını görürüz. Artışlı model yaklaşımı istenilen fonksiyonların modüler olarak ayrılabilen tipte olmasını gerektirir. Ne yazık ki farklı bir fonksiyona bağlı olmadan görevini yerine getirebilen fonksiyonların sayısı oldukça azdır. Bu durumda da modülerlik oluşması beklenemez.



**Şekil 1.4** Artışlı Model

Geçmişten günümüze kadar olan modelleri genel kapsamıyla ele aldık. Şimdi de çevik programlama yaklaşımı ile bilinen programlama metodojilerine göz gezdireceğiz.

**Çevik Programlama Yaklaşımı Nedir?**

Çevik programlama yaklaşımı yazılım dünyasındakilerin günümüze kadar karşılaştıkları zorluk ve deneyimlerin üzerinde durularak bu zorlukların aşılmasına yönelik ortaya atılan ideal bir yaklaşımdır.

Öncelikle yazılımdan genel kapsamıyla nelerin beklendiği üzerinde duralım;

Müşteri siparişin en kısa sürede hazır olasını ister. Gereksinimlerin tam anlamıyla karşılanabilmesinin yanı sıra süreç içerisinde oluşabilecek yeni gereksinimlerin de karşılanabilmesi için esnek bir yapısının olması gerekir.

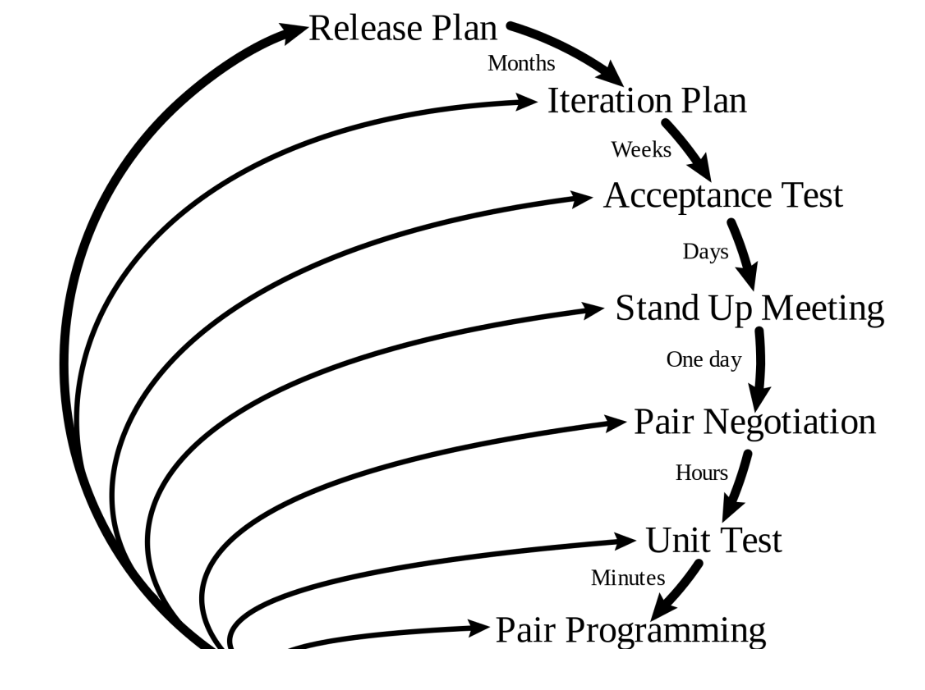
Çevik programlama bahsedilen beklentilerin tamamını karşılayabilecek şekilde düşünülmüştür;

Hızlı, verimli, hata oranı minimum, yeniden yapılandırılabilir, başarılı bir yaklaşım oluşturulmuştur.

Bu yaklaşımda gereksinimler önem sıralarına göre 2-4 haftalık gerçekleştirebilecek kısımlara ayrılır. 2-4 hafta sonucunda müşteri projenin en önemli kısmını barındıran çalışır programı teslim alır. 2-4 hafta çok kısa bir zaman dilimidir. Hız faktörünün bu şekilde iyileştirilesinde grup içi iletişim önemli rol oynar. İletişimi sağlıklı olan deneyimli tasarım ve uygulama aşamalarına hakim olan kişilerden oluşan grupların bu yaklaşımı benimsemesi uygundur. Aksi taktirde yaklaşım sağlıklı bir şekilde uygulanamaz ve beklenen sonuçlar elde edilemez.

Çevik programlamanın en çok kullanıldığı yaklaşımlardan iki tanesini bu yazımızda inceleyeceğiz.

**Extreme Programming (XP)**

Extreme programlama yazılım ekibi içi iletişime önem veren bir yaklaşımdır. İletişim değerinin yanında daha birçok unsurun üzerinde durur. Bunları basitlik, cesaret ve geri bildirim kavramları olarak ele alabiliriz. Kavramların üzerinde tek tek durmak gerekirse; basit bir çözüm çoğu zaman problemin karmaşık çözümünden daha zor elde edilir. Extreme programlama mantığında uygulanan problemler zaten zor olacağı için olabildiğince basit bir yol izlenip karmaşıklığın önüne geçilmesi hedef alınır.

**Şekil 2.1** Extreme programming

Cesaret kavramı geliştiricinin hazır hale getirdiği veya ilerleme kaydettiği bir yazılım ürününün hata ile karşılaşması durumunda problemi anlayıp çözmeye çalışması ihtiyaç halinde gerçekleştirdiği yere kadar olan kısmı iptal edip projeye tekrar başlamasını ele alır. Çoğu zaman geliştirici tekrar hata yapabileceği düşüncesiyle projeyi baştan yapmaya başlamak istemez (gerçekleştirdiği kısmı).

Geri bildirim kavramı proje ilerlerken açığa çıkabilecek hata oranını en aza indirmeyi amaçlar. Geri bildirim iletişimin doğal bir sonucudur. Buradan da iletişimin grup için neden bu kadar değerli olduğunu anlayabiliriz.

Ele alınan bu 4 değerli kavramın yanı sıra 12 ile sınırlı olan extreme programming temel pratiklerine de göz atalım. Bu pratiklerden ilki planlama oyunudur. Kulağa eğlenceli gelen bu yaklaşım önceliklerine göre parçalara bölünmüş proje kısımlarının ne kadar sürelerde gerçekleştirilebileceğini belirlemede ekip üyelerini ortak karar vermesini sağlar. Ekipte müşteri pratiği yazılım gerçekleştirilirken yanlış anlaşılan kısımlar varsa müşterinin işe dahil olmasıyla görülmesi ve fazla ilerleme kaydetmeden bu kısımların halledilmesine olanak sağlar. Bu kısımlar önceden belirlenip çözüme ulaştırılmazsa daha sonra maliyet ve zaman sorunu ile karşımıza çıkar. Önce test pratiğinde yazılacak olan programın öncesinde bir benzeri yapılır olası hatalar burada tespit edildikten sonra daha sağlıklı bir program yazılmasına olanak sağlar. Basit tasarım pratiği programın kolay anlaşılması ve kullanıcının kullanırken zorluk yaşamaması için önemlidir. Çiftli programlama pratiği programda üzerinde çalışılan kısmın birden fazla kişi tarafından aynı zaman dilimleri içerisinde yapılması yaklaşımıdır. Bu pratik aynı resme farklı bakış açılarıyla yaklaşım şeklinde betimlenebilir. Olası hataların fark edilebilmesi imkanını sağlar. Daha iyi sonuç alan yazılımcı yaptığı yazılımla diğerlerinin gelişimine katkı sağlar. Bu pratik yazılıma yeni atılmış bireyler için oldukça yararlıdır (usta-çırak ilişkisi). Sürekli entegrasyon pratiği yazılıma yeni eklentileri ve sistem değişikliklerini sağlamaya yönelik pratiktir. Daha sonrasında yapılan entegrasyonlar test edilir. Olası hata riskini azaltmada yardımcı olur. Kısa aralıklı sürümler pratiği yazılım bütününü 2-4 haftada gerçekleştirilebilecek minik parçalara ayırıp her 2-4 hafta sonunda müşteriye çalışan bir yazılım ürünü teslim etme yaklaşımıdır. Müşteri memnuniyeti açısından önemli rol oynar. Yeniden yapılandırma pratiği yazılan programın daha basit bir şekilde yeniden yazılmaya çalışılmasını ve müşteri tarafından gelen yeni isteklerin sisteme daha kolay uygulanması için gerekli yapılandırmaları kapsar. Ortak kod sahiplenme pratiği yazılımın yazılım ekibinin ortak ürünü olduğunu benimseme yaklaşımıdır. Benzetim (metafor) pratiği yazılımda uygulanacak kısmın daha önce yapılmış örneklerinden yaralanarak veya direkt kullanarak zamandan tasarruf edilmesi yaklaşımıdır. Kodlama standardı pratiği yazılım ekibi üyelerinin birbirlerinin yazdığı kodları anlayabilmesi yönünden önemlidir. Kullanılacak yazılım teknik bileşenlerinin standartlaştırılması yaklaşımıdır. Son olarak haftada 40 saat pratiği yazılım ekibinin maksimum verimle çalışabilme süresi olduğu fazla mesailerde verimin düşeceğini ileri süren bir yaklaşımdır.

**Scrum**

Scrum programlama yaklaşımı kompleks ve ilerisi tam anlamıyla görülemeyen projelerde uygulanır. Gerçekleştirilmesi istenen bütünün gereksinim önceliklerine göre küçük parçalara (sprint) ayırılmasını benimser (product backlog). Bu küçük parçalar 30 gün içerisinde halledilebilir parçalar olmalıdırlar (sprint backlog). Parçalar gerçekleştirilirken yazılım ekibi günde 15 dakikalık toplantılar yaparak yazılımın genel kapsamıyla ilgili birbirleriyle iletişim halinde olurlar. Bu 15 dakikalık toplantılar projenin sağlıklı ilerletilebilmesi için oldukça önemlidir (sprint backlog). Scrum programlama yaklaşımının günümüzde bu kadar popüler olmasının nedeni yapılması istenen işin kısa süreler içerisinde halledilebilir olmasını vaat ediyor oluşudur. Scrum programlama yaklaşımının temel kavramları; roller, toplantılar ve bileşenler/araçlardan oluşur.

**1-Roller**

Müşteri projenin ilerleyişinde yazılım ekibine geri dönütler vererek istenilen şeklini almasından sorumludur. Scrum yöneticisi ekibin yazılım ortamını belirler. Yazılımın ileri safhalarında ekibin nerelerde zorlandığını gözlemleyerek ekibe destek olur. Scrum takımı yazılımı gerçekleştirecek bireyler topluluğudur.

**2-Toplantılar**

Sprint planlama toplantıları genel olarak bir yazılıma başlanmadan önce yapılan gereksinimlerin belirlenmesi, yapılacakların küçük kısımlara ayrılması, küçük kısımların ekip üyelerine dağıtılması, olası ilerde karşılaşılacak hataların tespiti, yazılımın hangi sistem ve araçlar kullanılarak yapılacağı gibi temel problemlerin karşılaştırıldığı toplantılardır. Sprint gözden geçirme toplantıları planlama toplantılarında elde edilen verinin planlanıp uygulamaya geçilmesi süreçlerinin konuşulduğu toplantılardır. Günlük scrum toplantıları ekip içerisinde geri dönüşlerin alınmasını sağlar. Günde 15 dakika boyunca, o gün içinde ne kadar ilerleme kaydedilebileceği, oluşan sorunlara yönelik neler yapılabilineceği gibi yazılım sürecinin doğal olarak getirdiği durumlar tartışılır.

**3-Bileşenler/Araçlar**

Ürün gereksinim dokümanı müşterinin yazılımdan beklediği kısımların belgelenmiş halidir. Bu belgeye ihtiyaca göre yeni gereksinimler eklenebilir. Aynı zaman gereksiz olduğu anlaşılan (müşterinin geri dönüşüyle) kısımlar kaldırılarak sadeleştirilebilir. Sprint dokümanı yapılması kararlaştırılan işi ve ekip içi görev dağılımını belgeler. Sprint kalan zaman grafiği ekip üyelerinin bir iş (sprint) için tanımlanan süre ile gerçekleştirdiği süreyi girdiği belgelerdir. Ekip üyeleri bu belgeye bakarak kalan işi ve zamanı daha iyi gözlemleyip bu gözleme göre kalan zamanı planlamadırlar.

Yararlanılan kaynaklar:

* <http://ybsansiklopedi.com/wp-content/uploads/2015/08/Yaz%C4%B1l%C4%B1m-Geli%C5%9Ftirme-Modelleri-Yaz%C4%B1l%C4%B1m-Ya%C5%9Fam-D%C3%B6ng%C3%BCs%C3%BCSDLCYBS.pdf>
* <https://polen.itu.edu.tr/handle/11527/5973>
* <https://mmf.ders.bakircay.edu.tr/mod/resource/view.php?id=767>
* <https://mmf.ders.bakircay.edu.tr/mod/resource/view.php?id=824>