**YAZILIM YAŞAM DÖNGÜ SÜREÇLERİ VE ÇEVİK YAZILIM GELİŞTİRME**

**Mücahit Fatih Parlağı[[1]](#footnote-1)**

**Özet**

Bu araştırmada yazılım sektöründeki en büyük sıkıntı olan zamanında ve proje maliyetini aşıma uğratmadan iş yapmak için 1960’lı yıllardan bu yana geliştirilmiş farklı yazılım geliştirme modellerinden ve Çevik (Agile) yazılım geliştirmeden bahsedilecek ve hangisinin günümüzde en çok kullanıldığı nedenleriyle birlikte açıklanacaktır. İnternetteki çeşitli kaynaklardan ve Bilgisayar Bilimi kitabından yararlanılarak elde edilen bulgulara göre adını bir Rugby taktiğinden alan Scrum geliştirme metodunun günümüzde en çok kullanılan metot olduğu görülmüştür. Bu araştırma sonucu yazılım sektöründeki işverenler, yazılım ekip üyeleri ve müşteriler için önemlidir.

**Anahtar Kelimeler:** Çevik, esnek**,** yazılım geliştirme, yaşam döngüsü

**Abstract**

In this research, different software development models and Agile software development that have been developed since the 1960s to deal with the biggest problem in the software industry on time and without exceeding the cost of the project will be discussed, and which will be explained together with the reasons why it is used the most. According to the findings obtained from various sources on the Internet and using the Computer Science book, Scrum development method, which takes its name from a Rugby tactic, has been found to be the most used method today. As a result of this research, it is important for employers, software team members and customers in the software industry.

**Keywords:** Agile, flexible, software development, life cycle

**Giriş**

Her ürünün elde edilmesi sırasında ardı ardına izlenmesi gereken adımlar bulunur. İnsanın emeklemeden yürüyemeyeceği gibi bir ürün geliştirme sürecinde de adımlar sırasıyla izlenmeden son safhaya gelinemez. İşte bu şekilde her üründe olduğu gibi bir yazılım ürününün de ortaya çıkması için gerekli bazı aşamalar vardır. Yazılımın ilk ele alındığı planlama (gereksinim) aşamasından emeklilik aşamasına kadar süren bu sürece yazılım yaşam-döngü süreci adı verilir. Beş ana temelden oluşan ve bunların etrafında gelişip çeşitlenen yazılım yaşam-döngü süreçleri bir yazılımın ömrü boyunca geçirdiği tüm aşamaları kapsar. Gereksinim, analiz, tasarım, gerçekleştirme ve bakım temellerinden oluşan bu süreç yazılımın emekliye ayrılmasıyla son bulur. Yazılımın geliştirme aşamasında oluşabilecek sorunların en aza indirilmesi için bu basamaklar belirlenmiş ve geliştirilmiştir. Plansız yürüyen bir yazılım geliştirme ekibinde karmaşıklıkların çıkması işten dahi değildir. Ayrıca plansız yazılan kodda oluşabilecek hataların önlenmesi için de bu basamaklar oldukça önemlidir. Farklı projelerin, farklı gereksinimlerin gerektirdiği farklı yaşam döngü modellerinde bu aşamalar artabileceği gibi birbiriyle birleşebilir ve farklı haller alabilir.

Gereksinim aşamasında yazılım ürünü ana hatlarıyla belirlenir ve müşteriden gereksinimler alınarak bunların fizibilitesi sorgulanır ve buna uygun bir plan oluşturulur. Bir sonraki aşama olan analiz (çözümleme) aşamasında ise sistem gereksinimler ve işlevler ayrıntılı olarak belirlenir, riskler tartışılır ve yazılım projesinin süresi belirlenir. Daha sonra projenin belirlenen gereksinimlere yanıt verecek şekilde tasarımı ve temel yapısının oluşturulması gelir. Yazılım projesinin gerçekleştirilmesi (kodlama ve test) de bittikten sonra bakım aşamasına geçilir. Bu aşamada gerekli güncellemeler, hata düzeltmeleri e-ve yeni gereksinimlere cevap verebilecek düzeltmeler yapılır. Ayrıca yazılımın değişen teknolojiye yanıt verecek forma getirilmesi de bu aşamaya dahildir. Tüm bu aşamaların sonucunda yazılım ömrünü tamamlayıp emekliye ayrılana kadar gerekli aşamalar tekrarlanır, bakımlar yapılır ve en son da yazılımın ömrü biter.

Bu geleneksel süreçler plana uyulması halinde dahi sıkıntılı ve uzun bir süreç haline gelebilmektedir. Sektörde yazılım ürünlerinin zamanında ortaya çıkamaması, değişen gereksinimlere hızlı cevaplar verilememesi, yazılım ürününün zaman içerisinde gelen yeni isteklere göre yapısını geliştirememesi gibi farklı sorunlar ortaya çıkmış ve geleneksel modellerle bu sorunun aşılamaması sonucu 1990’lı yılların sonuna doğru “çevik (agile)” isimli metotlar geliştirilmiştir. Bu metotlar piyasaya hızlı çıkan, değişen gereksinimlere çabucak yanıt veren ve müşteriye kısa sürede ulaşan ürünler sunmayı amaçlar.

**Yazılım Yaşam Döngü Modelleri**

Yazılım yaşam döngü süreçlerinde her projenin boyutu, gereksinimleri veya hedef kitlesi gibi değişkenlere bağlı olarak kullanılan farklı modeller bulunmaktadır.

**Gelişigüzel ve Barok Döngü Modelleri**

Bu modellerin ilki her ne kadar bunu bir model olarak ele almak doğru olmasa da 1960’lı yıllarda kullanılan gelişigüzel modeldir. Genellikle kişiye bağlı yazılım geliştirme şeklinde yapılan bu modelde belirli bir yöntem olmadığından gözetilmesi ve bakımı oldukça zordur.

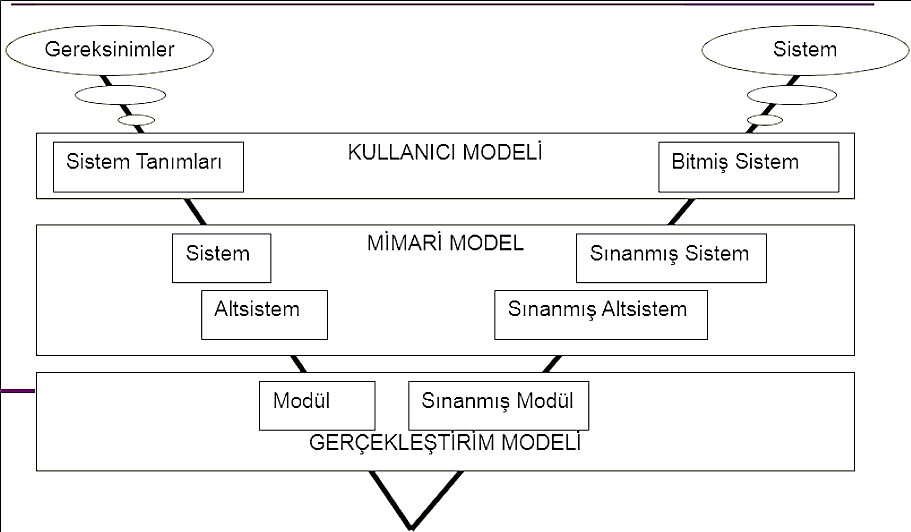
Bir diğer model olan barok model ise yaşam döngü modellerinin temel adımlarını doğrusal bir şekilde izler lakin bu modelde de aşamalar sırasında geriye dönüşlerin nasıl yapılacağı belirlenmemiştir. Dokümantasyonun geliştirme ve testten sonra uygulanan farklı bir süreç olarak ele alınması da bu model günümüz modellerinden ayıran bir diğer özelliğidir. Gelişigüzel model gibi barok model de günümüzde uygulanan modeller arasında olmaktan çıkmıştır.

**Çağlayan Yaşam-Döngü Modeli**

Günümüzdeki yaşam-döngü modellerinin temelini oluşturan çağlayan modeli geçmişte en çok kullanılmış modeldir. Geleneksel yazılım geliştirme modeli olarak da anılan çağlayan modeli temel aşamaların en az birer kez tamamlanmasını öngörür ve bir adım tamamlanmadan diğer adıma geçilmez.

Bu modelin çok iyi tanımlanan ve fazla zaman harcanmayacak projelerde kullanılması uygundur. Dokümantasyonu sürecin bir parçası olarak görmesi çağlayan modelini barok modelden ayıran en belirgin özelliğidir. Barok modelden bir diğer farkı ise yapılması gereken geri dönüşlerin tanımlı olmasıdır. Bu modelde müşteri sürecin bir parçası olmadığından ileride oluşabilecek sorunları en aza indirmek için en başta gereksinimlerin çok iyi bir şekilde belirlenip analizin en ince ayrıntısına kadar yapılması gerekmektedir. Tasarım kısmında da belirlenen bütün gereksinimler göz önünde bulundurularak detaylı bir çalışma yapılmalıdır. Yazılımcılar genel olarak bir an önce kod yazma ve ürün görme eğiliminde olduklarından bu aşamalarda çok vakit harcamış yazılım ekiplerinin mutsuzlaşması kaçınılmazdır. Bu modelin eksiklikleri de bu kısımda başlamaktadır. Ayrıca detaylıca yapılması için fazlaca çalışılmış gereksinimlerin ileriki safhalarda değişmesi durumunda özellikle uzun süren projelerde bu değişikliklerin yazılıma yansıtılması oldukça zor ve maliyetlidir.

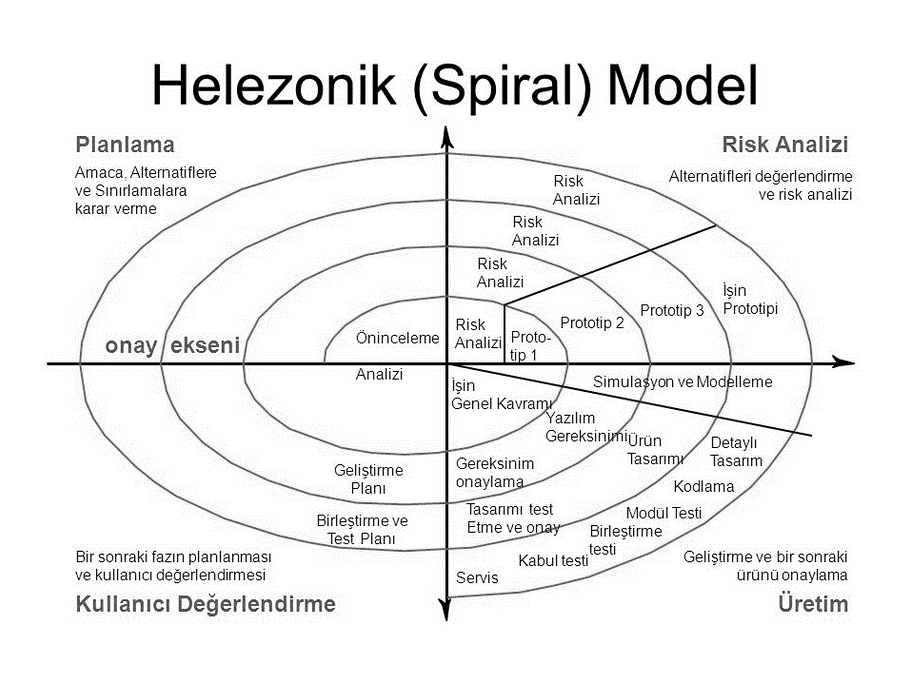
**V Süreç Modeli**

V Modeli çağlayan modeline çok benzemesiyle beraber daha gelişmiştir. İş tanımları belirgin, belirsizlikler azdır. Bu modelde kullanıcının projeyle olan bağı artar. Risk analizinin bulunmaması ve aşamalarda tekrarın bulunmaması modelin eksiklikleridir.

Modelin sol kısmı üretim, sağ kısmı ise test işlemlerini içerir. V modelinin üç temel çıktısı vardır:

* **Kullanıcı Modeli:** Sürecin kullanıcı ile ilgili kısımlarını içerir. Kullanıcının isteklerine göre proje tanımı ve planı oluşturulur.
* **Mimari Model:** Projenin tasarımını ve sınanmasını içerir.
* **Gerçekleştirim Modeli:** Kodlama yapılan ve yazılan kodların denendiği bölümdür.

**Helezonik (Spiral) Model**

Helezonik modelde diğer modellerdeki gibi doğrudan belirlenmiş gereksinim, analiz vb. fazlar yoktur. Bunun yerine modeldeki her döngü bir fazı temsil eder ve bunlar yinelemeli artımsal bir yaklaşımla ele alınır. Her aşamada birer prototip hazırlandığından kullanıcıya projeyi adım adım takip etme olanağı sunar. Bu yaklaşım da yazılımcıların erkenden kodlamaya geçerek mutsuzlaşmalarını önlemekle beraber oluşan hataların daha o aşamadayken kullanıcı ve yöneticiler tarafından da görülerek halledilmesini sağlayarak sonradan bir sorun çıkma olasılığını azaltır. Lakin bu model ancak büyük ve risk olanağı çok olan projeler için uygundur. Küçük ve düşük riskli projelerde bu yaklaşımlar maliyeti arttıracağı gibi karmaşık yapısı ve fazlaca dokümantasyon tutulması projenin haddinden uzun sürmesine de yol açacaktır.

Helezonik yazılım geliştirme modeli dört temel aşamadan oluşur:

* **Planlama:** Her döngüde oluşacak ara ürünün planının yapıldığı aşamadır.
* **Risk Analizi:** Risklerin araştırıldığı, belirlendiği ve çözümlendiği aşamadır.
* **Üretim:** Ara ürün geliştirilen kısımdır.
* **Kullanıcı Değerlendirmesi:** Ara ürün ile ilgili kullanıcının sınama ve değerlendirme yaptığı kısımdır.

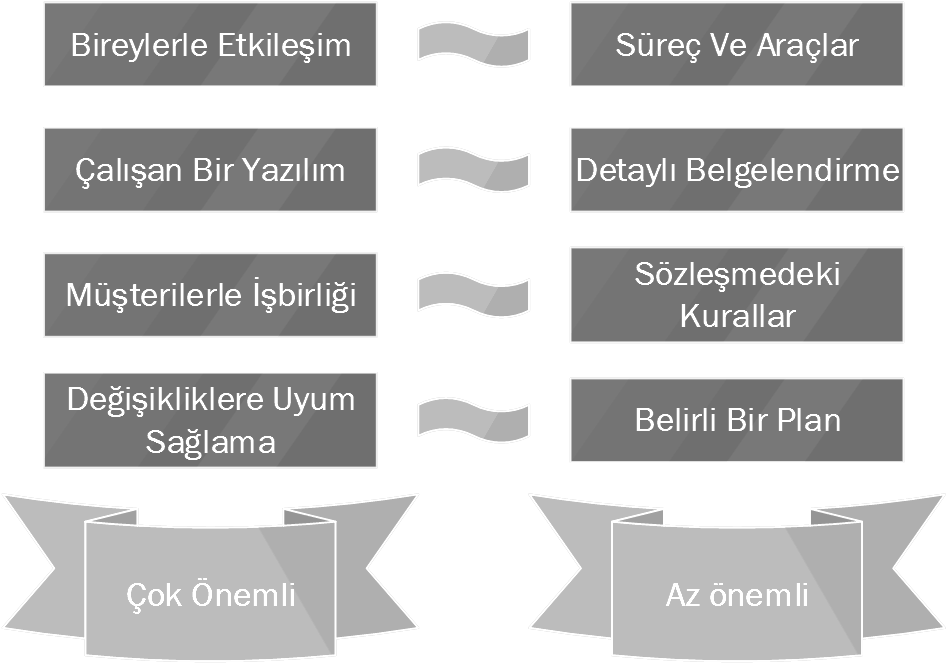
**Artımsal Geliştirme Süreç Modeli**

Artımsal geliştirme modeli, sürecin parçalanarak kullanıcı önceliğine göre sıralandığı ve sonra sırasıyla bu parçalardan birer ara ürün elde edildiği modeldir. Bu ara ürünlerin her biri bittikten sonra kullanıcıya sunulur ve kullanıcı tarafından da bu ara ürünler kullanılır. Sıralanmış her ara ürün bir önceki ara ürünün üstüne bir şeyler eklenerek elde edilir. Her aşamada üretim devam ederken aynı zamanda kullanım da olur. Bu da artımsal modeli uzun sürecek lakin hemen kullanılması gereken veya eksikliklerle de çalışabilecek projeler için biçilmiş kaftan yapar. Her aşamada kullanıcının yazılımı kullanması da üretimdeki karşılaşılabilecek hataları azalttığı gibi gereksinimlerin de doğru şekilde karşılanmasına olanak sağlar. Her sürümden sonra kullanıcı tarafından alınan geribildirimlere göre bir sonraki sürüm kullanıcı istek ve ihtiyaçlarına göre düzenlendiği için müşteri memnuniyeti de artmış olur. Lakin bir parçanın/sürümün geliştirilmesi başladıktan sonra geliştirilmesi dondurulur ve değişsen gereksinimler ancak bir sonraki sürüme aktarılabilir, bu da bir dezavantaj olarak sayılabilir.

**Kodla ve Düzelt Yaşam-Döngü Modeli**

Küçük, birkaç yüz satırdan oluşan projeler için kullanılabilir bir model olan kodla ve düzelt modelinde direkt olarak bir sürüm gerçekleştirilir ve kullanıcı tatmin olana kadar geliştirilir. Bu geliştirme sürecinde bakım da bulunur fakat bakım yapılması dokümantasyon bulunmadığı için oldukça zordur. Yazılımcı direkt ürün geliştirir ve kullanıma sunar. Yazılım emekli olana kadar bu şekilde kullanılır. Bu model çok kolay bir yazılım geliştirme yolu olmasına rağmen en pahalı yoldur.

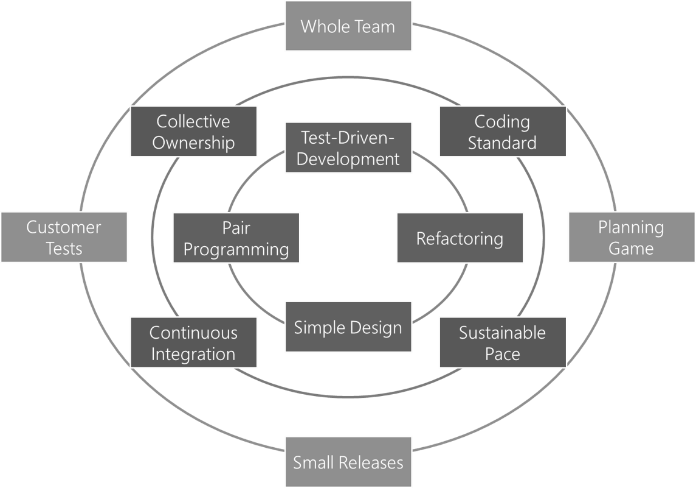
**Çevik (Agile) Geliştirme Süreçleri**

Tüm diğer yazılım süreçlerini incelediğimizde artıları ve eksileri olduğunu görürüz. En mükemmel görünen, projemize en uygun yazılım geliştirme modelini seçsek dahi bazı şeylerin istenildiği gibi gitmemesi kaçınılmazdır. Projelerde süre, maliyet aşımı olmasının yanı sıra yazılımların değişikliklere hızlı bir şekilde cevap verememesi veya yazılım hatalarının geç fark edilmesi gibi etkenlerden kısmi başarı ve memnuniyetle tamamlanan projelerin çoğunlukta olması yazılımcılar tarafında bir soruya ve arayışa neden olmuştur. Yazılımcılar en hızlı ve hatasız şekilde yazılım geliştirmenin yollarını aramış ve bu arayışların sonucunda 1990’lı yılların sonlarına doğru “Çevik (Agile)” geliştirme metotlarını bulmuşlardır. Bu metotlar değişen isteklere en hızlı şekilde yanıt vermenin yanı sıra kısa sürede piyasaya ürün çıkarabilmeyi ve bunu müşteriye ulaştırmayı amaçlar. Bu metotlarda projenin boyutu ne olursa olsun ‘böl ve yönet’ tekniği kullanılır. Yani proje küçük parçalara ayrılır ve her parça ayrı şekilde ele alınır. Her parça sonucunda kullanıcıya rapor verilir. Her yineleme ortalama 2-4 hafta sürer ve hepsinin sonucunda müşteriye çalışan bir program sunulur. Müşterinin elinde sürekli çalışan bir ürün olması müşteri memnuniyetini arttırmakla beraber yazılımcıların kod yazmadan sürekli dokümantasyon üzerinde çalışarak mutsuzlaşmasını da önler. Ayrıca bu küçük parçalar projede geriye dönük hataların düzeltilmesini de kolaylaştırır.

Çevik yazılım geliştirmenin birinci önceliği müşteriye hızlı ve sürekli olarak kullanılabilir ürün sağlamaktır. Detaylı belgelendirme yaparak hem zaman kaybı yaşamak hem de yazılımcıları mutsuzlaştırmak yerine çalışan bir yazılım yapmak her zaman daha önemlidir. Ayrıca katı kurallarla ilerleyip değişiklikleri reddetmektense en ileri safhalarda dahi değişiklikleri kabul edip uyum sağlayarak esnek bir yapı edinmek de çevik sürecin vazgeçilmezlerindendir. Bu da sürecin esnek olmasını sağlar. Tüm ekip birbiriyle sürekli olarak iletişim ve işbirliği halindedir. Ekip içi iletişimin arttığı bu süreçte aynı zamanda kullanıcı da sürecin içinde dahil edildiğinden hata oranı oldukça düşer. Belirli aralıklarla ekip içinde yapılan gözden geçirmelerin de bu noktada rolü büyüktür. Hataların bu şekilde erkenden çözüme kavuşması maliyeti ciddi ölçüde düşürerek başarı oranı ve verimliliği arttırır. Öyle ki bir yazılımın ilk aşamalarında çözülebilecek bir hatanın maliyeti 1 birim iken bu hatanın daha ileriki safhalarda, örneğin bakım aşamasında, çözülmesinin maliyeti 10 birimdir. Verimli, esnek, hata oranı düşük, hızlı ve ucuz çözümler olan çevik yazılım geliştirme metotlarından birkaçı şunlardır:

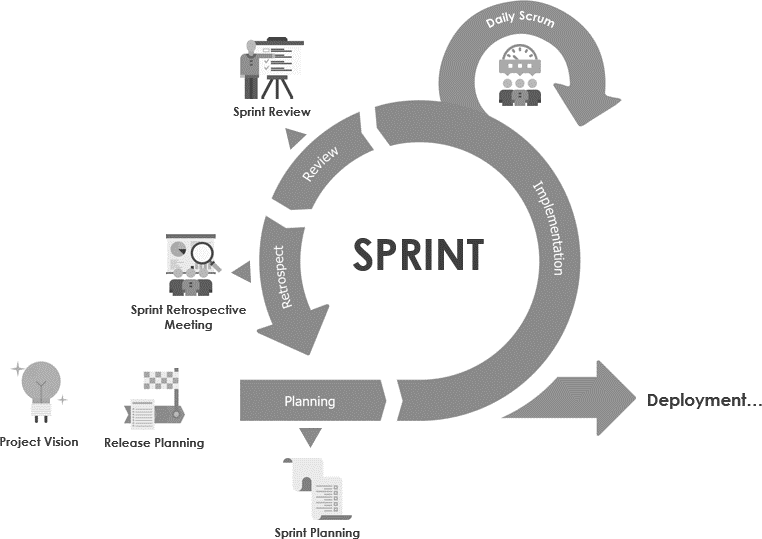
* **Extreme Programming (XP)**

XP, Kent Beck tarafından 1999 yılında çıkarılan kolay, grup içi iletişime önem veren, geri dönüşlerin fazlaca olduğu bir yazılım geliştirme disiplinidir. 12 farklı pratiğe ve 4 temel değere sahiptir. Bu temel değerler iletişim, basitlik, geri bildirim ve cesaret olarak sıralanabilir.

****Basitlik yazılan kodun sade, karmaşıklıktan uzak olmasını, aynı zamanda uzun uzun yazılan dokümantasyondan uzak olmasını temsil eder. Sorunlara en basit çözümler uygulanmalı, detaylarda boğulmaktan kaçınılmalı, sadelik sağlanmalı ve bu sayede de yazılımın daha esnek olması sağlanmalıdır. Bu kolay bir iş gibi gözükse de en zor olan konulardan biridir. Karmaşıklık aynı zamanda projenin başka kişilere aktarılmasını da zorlaştıran bir etkendir. Bu noktada projelerde bulunan genel sıkıntılardan bir diğeri olan insanların aralarındaki iletişim eksikliği ve bunlardan kaynaklanan hatalar karşımıza çıkar. XP disiplininde iletişimin temel değerlerden biri olmasının ana nedeni de iletişim eksikliğinin yol açtığı büyük sorunlardır diyebiliriz. Bu eksikliğin sebep olduğu sorunların farkında olan XP iletişim eksikliğini ortadan kaldırmayı amaçlar ve buradaki iletişim mümkünse yüz yüze olmalıdır. Yüz yüze iletişimin avantajları karşı tarafın kullandığı kelimelerle beraber mimiklerini, ses tonunu ve bunun gibi etkenleri de hissederek daha etkili bir iletişim sağlayabilmektir. Yazışmalarla veya sadece sesli yapılan görüşmelerde bunlar aynı anda bulunmadığı için yanlış anlaşılmaların arttığı bariz bir gerçektir. İşte bu şekilde iletişimi arttırmakla kullanıcıdan alınan geri dönüşler de daha kaliteli ve daha açık olabilir. Dört ana prensibin bir diğeri olan geri dönüşler, iletişim olmadan düşünülemez. Geri bildirimler sayesinde müşteri, proje grubunun bir üyesi haline gelir ve oluşabilecek hatalar en aza indirgenir. Sonradan doğabilecek anlaşmazlıklar kısa aralıklı geri dönüşlerle zamanında çözülür ve bu karşılıklı iletişim sayesinde verimlilik artar. Tüm bunlara rağmen çıkan hatalarda ise XP cesur olmayı, başarısızlıktan korkmayı değil başarısızlıkları en kısa sürede çözmeyi hedefler. Yazılımcının bu noktada içine sinmeyen, gereksinimleri karşılamadığını düşündüğü kısımları hiç çekinmeden cesurca silip baştan başlayabilmesi önemlidir. Bu da aslında XP’nin dört temel noktasından en zorudur diyebiliriz. Yazılımcının kafasındaki “Ya daha kötü olursa?”, “Ya tekrar yazamazsam?” gibi soruları aşması kolay olmayabilir ancak bunları yıkmak XP modelinde çok önemlidir. Bu tarz korkular yazılımcıyı ve dolayısıyla projeyi yavaşlatır. Bu sebeple yazılımcının cesur olması ve cesur kararlar alması çok önemlidir.

Yazılım geliştirmeyi kolay ve esnek bir şekilde ele alan XP’nin 12 pratiği şu şekilde sıralanabilir:

* Planlama oyunu
* Ekipte müşteri
* Önce test
* Basit tasarım
* Çiftli programlama
* Sürekli entegrasyon
* Kısa aralıklı sürümler
* Yeniden yapılandırma
* Ortak kod sahiplenme
* Metafor
* Kodlama standardı
* Haftada 40 saat
* **SCRUM**

Scrum ismini Rugby sporundaki bir hücum taktiğinden almaktadır. Bu modelde de iş parçalara bölünür ve bu parçaların her birine “sprint” adı verilir. Scrum sadece yazılım geliştirmeye değil, her yere uyarlanabilecek bir proje yönetim yaklaşımıdır. Bu modelde kullanılan böl ve yönet mantığı, karmaşık ortamlarda adım adım geliştirme yapan yazılım ekiplerine oldukça uygundur. Ekip içi iletişimin fazlaca öneme sahip olması bu durum için büyük bir etkendir. Ekip içi iletişim öylesine gereklidir ki günlük “Scrum Meetings” ismi verilen 15 dakikalık toplantılar yapılır ve bu şekilde iş takibi kolaylaşmış olur. Bu toplantıların önemini anlamak için şöyle düşünebiliriz; ekipteki her birey günlük olarak yaptıklarını diğer ekip bireylerine anları ve diğer bireylerin de yorumlarıyla beraber hatalar, eksiklikler hızlıca anlaşılabilir. Diğer bir yandan ise o gün yapacaklarını birbirine anlatan ekip bireyleri güne motive olmuş bir şekilde başlar ve iş verimliliği artar. İnsanın yaptığı ve yapacağı şeyin belirli olmasının, önünü görmesinin motive edici bir şey olduğu aşikardır. Bu noktada şöyle bir örneğe başvurabiliriz; bir yüzücü sisli bir havada bir metre bile öteyi göremezken, 100 metre olduğunu bildiği bir parkurda yüzmeye başlasın. Yüzücü ne kadar hızlı yüzerse yüzsün kat ettiği mesafeyi ya da bitişe ne kadar yaklaştığını bilemez ve kendini sonsuz bir suyun içinde gibi hisseder. Aynı şekilde yüzücü sis olmadığı bir gün aynı parkurda yüzdüğünde ne kadar yol kat ettiğini ve bitiş çizgisini gördüğü için bir moral kaybı yaşamayacak, aksine ilerledikçe daha da motive olacaktır. İlk durumdaki yüzücü belki parkuru bitirmeden yarısında bu yarışı bitiremeyeceğini düşünüp pes edecekken diğer yüzücü hızlı bir şekilde bitirecektir. Aynı şekilde yaptığı ve yapacağı işi bilen yazılımcılar da işine daha motive bir biçimde başlayacak ve daha verimli çalışacaktır.

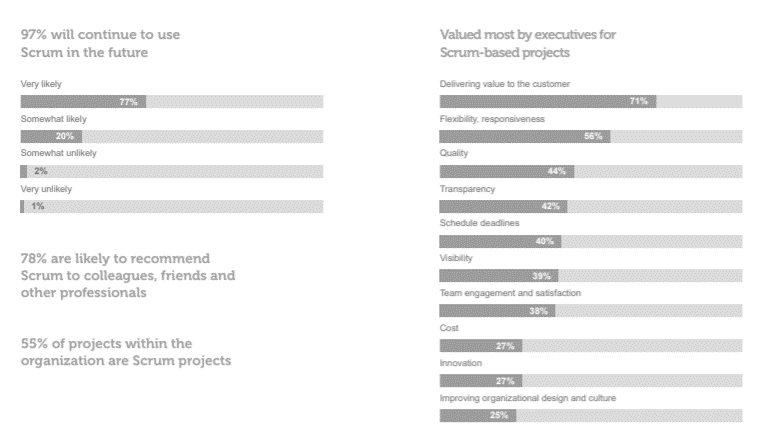
Scrum’da üç temel kavram vardır:

1. **Roller:** Ürün sahibi, Scrum yöneticisi ve Scrum takımından oluşur. Ürün sahibi müşteri tarafını temsil eder ve geri dönüşler sağlar. Scrum yöneticisi Scrum sürecini takip eder ve yönetir. Scrum takımı ise genellikle 5-9 kişiden oluşan ve sürekli iletişim halinde olan yazılım geliştirme ekibidir.
2. **Toplantılar:** Scrum’da olmazsa olmazdır. Her gün yapılan Scrum toplantıları haricinde sprint planlama ve sprint gözden geçirme isimli toplantılar da yapılır.
3. **Bileşenler/Araçlar:** Ürün gereksinim dokümanı, sprint dokümanı ve sprint kalan zaman grafiğini içine alır.

**SCRUM Neden Kazandı?**

Günümüzde en çok kullanılan proje yönetim biçimi olan SCRUM kısa sürede ve düşük maliyetle proje geliştirilmesine olanak sağlaması sayesinde birçok şirket tarafından tercih edilmektedir. Yinelemeli bir anlayışa sahip olması ve değişen gereksinimlere kolayca cevap verebilmesi de tercih edilme nedenleri arasındadır. SCRUM ile geliştirilen projelerde kullanıcıdan sürekli olarak gelen geribildirimlerin ve ekip içinde kurulan iyi iletişimin de yardımıyla hatalar kolaylıkla ve erkenden çözülür.

Scrum Alliance’ın 2017-2018 State of Scrum raporuna baktığımızda kuruluşların Scrum’ı seçme sebeplerinin başında müşteriye iyi bir değer sunmak olduğunu görüyoruz. Ayrıca bu metodolojinin esnek ve değişikliklere hızlı cevap verebilen yapıda olması, yaşam kalitesini arttırması da şirketlerin Scrum’ı seçme sebeplerinden bazıları.

Yazılım geliştirmeyi çevik bir biçime dönüştüren Scrum odaklanmayı ve iletişim halinde olmayı arttırır ve bu sayede rakiplerine nazaran çok daha başarılı projeler ortaya çıkarır. Çoğu yazılımcı tarafından bir metodolojiden çok bir kültür olarak tanımlanan Scrum bireyler arasında kendi kendine organize olmayı teşvik eder, insanlara değer verir.

Bazı araştırmalar ise Scrum’ın temel gücünün sadelik olduğunu ortaya atmakta. Anlamanın ve takip etmenin diğer birçok yöntemden basit olması sadeliği gücünü ortaya koymakta. XP veya diğer bilinen Çevik metotlardan farklı olarak Scrum kuralcı değildir ve çok fazla teknik disiplin gerektirmez. Ekibin neyi nasıl yapması gerektiğine kendi başına karar vermesi için olanak sağlar. Bu sayede hızlanarak hızlı, ucuz ve çevik bir şekilde yazılım geliştirebilirler. Bunun yanında Scrum’ı asıl öne çıkaran huşu ise sertifikasyondur. Sertifikalı Extreme Programmer yoktur. Buna karşın binlerce sertifikalı ScrumMaster, ürün sahibi, geliştiriciler… vs. vardır. Standardizasyon ve sertifikasyon nedeniyle Scrum’a başlamak düşük risklidir. Şirketleri Scrum’ı denemeye iten sebeplerden biri de bu olmuştur. Sertifikasyonun bir diğer faydası ise işe alımları kolaylaştırması. Şöyle ki diğer metotlarda yeni gelen bir ekip üyesinin kullanılan modele uyum sağlama süresi uzun, zahmetli ve maliyetli olabilir. Lakin Scrum’da sertifikalı bir ekip üyesinin modele hemen uyum sağlayabileceğini bilen işverenler daha kolay işe alım gerçekleştirebilirler. Aynı zamanda geliştiriciler tarafından bakıldığında da onlar için de bu sertifikalar işe alınma olanağını arttırıyor.

Sonuç olarak sadelik, çeviklik ve tabi ki sertifikasyonun da yardımıyla Scrum kazandı ve günümüzde en çok kullanılan proje yönetim metodu oldu.

# **Kaynakça**

Ali, U. (B.T.). *Extreme Programming*. steemkr: https://steemkr.com/technology/@samayyy/extreme-programming adresinden alındı

Alnıak, F. (2017, Mayıs 22). *Yazılım Mühendisliği: Yazılım Süreç Modelleri*. furkanalnıak: http://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/ adresinden alındı

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001). *Çevik Yazılım Geliştirme Manifestosu*. agilemanifesto: http://agilemanifesto.org/iso/tr/manifesto.html adresinden alındı

Bird, J. (2012, Kasım 27). *Building Real Software*. swreflections: http://swreflections.blogspot.com/2012/11/why-scrum-won.html adresinden alındı

Brookshear, J. G. (2007). *Computer Science an Overwiev.* Boston: Pearson Educational.

Çetin, Ö. H. (2019, Mart 10). *Yazılım Yaşam Döngü Modelleri*. medium: https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742 adresinden alındı

Kılınç, D. (2020). İzmir Bakırçay Üniversitesi Yazılım Mühendisliğine Giriş Dersi 2. ve 3. Hafta Ders Notları. İzmir, Menemen, Türkiye.

Petkar, P. (2015, Ekim 5). *Why is the Scrum process so popular in the software industry*. quora: https://www.quora.com/Why-is-the-Scrum-process-so-popular-in-the-software-industry adresinden alındı

Scrum Alliance. (2018). *STATE OF SCRUM 2017-2018.* Scrum Alliance.

Telef, Ç. (2015, Haziran 23). *Çevik Yazılım Geliştirme*. caglartelef: https://caglartelef.com/cevik-yazilim-gelistirme/ adresinden alındı

Townsend, J. (2013, Şubat 7). *Why Is Scrum So Popular*. techwell: https://www.techwell.com/2013/02/why-scrum-so-popular adresinden alındı

1. Öğr., İZ.B.Ü. Mühendislik ve Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü, m.fatihparlagi@gmail.com [↑](#footnote-ref-1)