Teknoloji günümüzde hemen her alanda kullanılmakta ve işimizi fazlasıyla kolaylaştırıp hızlandırmakta. Bugünkü teknoloji yokken insanların işlerini nasıl hallettiklerini, hayatlarını nasıl yaşadıklarını düşünemiyoruz bile, bizden sonraki nesillerin de bizim için böyle düşünecekleri şüphesiz çünkü zaman ilerledikçe teknolojinin de gelişme hızı artıyor, yani artan bir ivmeyle gelişiyor teknoloji. Teknolojiye erişim araçlarımız da bilindiği üzere elektronik aletler. Elektronik aletler de 2 temel kısımdan oluşuyor. Donanım (hardware) ve yazılım (software). Bu 2 kısım birbirinden bağımsız pek işe yaramaz, ikisinin de faydalı işler başarması için birbirine ihtiyacı vardır. Donanımı elle tutabilen kısımlar, yazılımı ise donanımlarca tutulan kısımlar olarak düşünebiliriz. Günlük işlerimizi yaparken dış donanım (dokunmatik ekran, klavye, monitör, fare…) ve dış yazılım (grafiksel kullanıcı arayüzü (“graphical user interface”, “GUI”) ile etkileşimde oluruz. Donanım ekipmanlarını ayrı olarak veya elektronik aletler ile entegre halde satın alabiliriz. Bunun yanında genelde basite indirgenip sadece kod yazmaktan ibaret olarak görülen yazılım da donanım gibi üründür ve her ürünün olduğu gibi yazılımın da bir ortaya çıkış ve varoluş süreci, kısacası bir yaşam döngüsü vardır.

Bir yazılım geliştirileceği zaman halihazırda belirlenmiş olan yazılım yaşam döngü modellerinden biri seçilir ve bu model kuralları çizgisinde geliştirilir. Yazılım yaşam döngüsü (software development life cycle) bir yazılımın hazırlanış ve müşteri tarafından kullanılış dönemi boyunca geçirdiği adımlar bütünüdür. Bu döngü stabil veya doğrusal olmamalıdır, yazılım ürününün işlev ve ihtiyaçları değişim ve gelişim durumunda olduğundan döngü de gelişen ve kendini yenileyen/tekrarlayan bir yapıya sahip olmalıdır.

Yazılım geliştirme aşamaları, **planlama**, **analiz**, **tasarım**, **gerçekleştirme** ve **bakım** olmak üzere 5 temel aşamadan oluşur.

* **Planlama:** Bu aşamada öncelikle müşterinin ihtiyaçları belirlenir. Proje maliyeti ve amacı belirlenir. Kullanılacak olan araç ve kişiler kararlaştırılır.
* **Analiz:** Planlama aşamasında belirlenen ihtiyaçlar ve gereksinimler netleştirilir, ardından belgeye dökülür. Bu netleştirme işlemi için yazılım mühendisiyle müşterinin iletişimi güçlü ve sağlıklı olmalıdır.
* **Tasarım:** Müşteri istek ve ihtiyaçları ile projenin detayları birleştirilerek yazılım ürününün projesinin çizildiği adımdır. Tasarım aşaması yemek tarifine benzer, yemek tarifinde yemeğin malzemeleri ve yemeğin yapılış adımları anlatılırken tasarım aşamasında da projenin ana bileşenleri, gereksinimleri ve gerçekleştirme planı bulunur. Yapılacaklar net olarak belirlenir ve kodlamaya girmeden projenin tasarımı oluşturulur. Analiz aşamasında daha çok sorunun ne olduğu üzerinde düşünülürken tasarım aşamasında daha çok sorunun nasıl çözüleceği düşünülür. Bunlar düşünülürken elbette müşterinin istekleri de planın şekillenmesinde rol oynayacaktır.
* Tasarım aşamasında 2 temel teknikten faydalanılır. **Soyutlama (Abstraction)** tekniğinde problemlerin anlaşılmasını ve çözülmesini kolaylaştırmak adına nesnelerin, durumların, olayların bazı özellikleri görmezden gelinir. Bu da problemin en kritik noktalarına odaklanmayı sağlar. **Modelleme** tekniğindeyse hedefin ayrıntıları soyutlanıp geliştirilen sistem görselleştirilir, alınan kararlar belgelenmiş olur. Problemin anlaşılması ve tasarım yaratımı bakımından faydalı bir tekniktir.
* **Gerçekleştirim:** Genelde sancılı geçen ilk 3 adımın ardından gerçekleştirim adımında genelde en sevilen ve zevk alınan **kodlama** işlemi gerçekleştirilir. İlk 3 adımda görev alacak kişi bulmak gerçekleştirim adımında çalışacak kişi bulmaktan daha zordur. Kodlama yapılırken kodlama kalite standartlarına uymak önemli bir faktördür. Debug işlemi de unutulmamalıdır.
* Kodlama kadar hatta kodlamadan da önemli olan aşama **test** aşamasıdır. Testlere mümkün olduğunca erken başlanmalı ve yazılımın yaşam döngüsü boyunca da devam edilmelidir. Test sayısı ve sıklığı ne kadar artarsa oluşacak olan hatalar ve kaybedilecek para miktarı da o kadar azalır. Ayrıca zaman ve saygınlık kaybedilmesinin de önüne geçilmiş olur.
* **Bakım:** Yazılım ürününün önemli testleri tamamlandıktan sonra ilk versiyonu ile kullanıcıya teslim edilmeye hazır hale gelir. Ama teslim adımından sonra çok önemli bir aşama olan bakım aşaması başlar ve yazılım ürünü ömrünü tamamlayana kadar da devam eder. Bu aşamada üründe tespit edilen hatalara veya kullanıcılardan gelen isteklere/ihtiyaçlara göre ürüne güncelleme (update) sunulur ve güncelleme notları da kullanıcı ile paylaşılır.

**Yazılım Yaşam Döngüsü Modelleri**

Yazılım geliştirme modelleri, yazılım geliştirmenin çok önemli bir parçasıdır. Özellikle profosyonel yazılım geliştirme şirketleri tarafından kullanılan birçok model bulunur. Bu çeşitliliğin sebebi projenin büyüklüğü, amacı, şirketin veya ekibin tarzının farklılaşabilmesidir. Bu sebepten de her bir modelin avantajları ve dezavantajları vardır. Bu modellerin genel amacı ise proje gereksinimlerine göre sorunsuz ve istenen şekilde bir yazılım geliştirme sağlamaktır. Bu modeller sadece yazılım geliştirme süreciyle ilgilidir, bu yüzden herhangi bir teknik yönü yoktur.

1. **Gelişigüzel Model:** 1960’larda kullanılan bu modeli model olarak adlandırmak dahi doğru değildir. Çünkü herhangi bir model veya yönteme başvurulmaz, tek kişiye bağlı olarak yapılır. Bu yüzden de zaman, izlenme ve bakım konularında sorunlar yaşanır. Genelde basit öğrenci proje veya ödevlerinde kullanılır fakat diğer alanlarda kullanımı kalmamıştır.
2. **Barok Modeli:** 1970’li yıllarda kullanılan bu modelde yaşam döngüsü adımları doğrusal olarak işlenir ve bu aşamalar arasında geri dönüşlerin nasıl yapılacağı bilinmez. Bu geri dönüşler oldukça önemli olduğundan modelin eksiklerinden biri geri dönüşlerin olmamasıdır. Günümüzde kullanılan modellerde belgeleme hemen her adımda yapılırken barok modelinde gerçekleştirime verilen önem daha fazladır ve belgeleme yazılımın gerçekleştirim ve test aşamaları tamamlandıktan sonra yapılır. Bu durum da barok modelini zaman ve maliyet yönünden kötü etkileyen faktörlerdendir. Bu olumsuz yönlerinden dolayı barok modeli de günümüzde kullanımı kalmayan modellerdendir.
3. **Şelale (Waterfall) Modeli:** Yazılım geliştirme ile ufak bir alakanız dahi olduysa size tanıdık gelecek olan bu model, yazılım geliştirmenin en geleneksel ve sık kullanılan modellerinden biridir. Hatta en tanınan ve en temel model demek de yanlış olmaz. Bu model yazılım geliştirmeyi doğrusal bir akışta işler, yani her aşama en az 1 kez tekrarlanır ve bir aşama tamamlanmadan diğerine geçilmez. Bu yaklaşımda, barok modelinin aksine aşamalar arası geri dönüşler bulunur. Yine de değişime pek elverişli bir model değildir, yapılan geri dönüşler zaman ve maliyette kayıplara sebep olur. Özellikle de büyük projelerde bu kayıplar kritik olabilir. Üzerinde en fazla durulan aşamalar analiz ve tasarım aşamalarıdır. Büyük projelerde gereksinimler değiştiğinde bu aşamalara geri dönmek çok daha zor ve zahmetli olur. Kullanıcı yazılım geliştirilirken sürecin içinde olmadığından kullanıcıdan alınacak geri dönüşler ancak yazılım yayınlandığında gelir ve bu süreden sonra geriye dönmek zaman ve maliyette büyük kayıplara sebep olur. Ayrıca programı yazıp çıktısını almak bu modelde gecikeceğinden çalışan ekip de bu durumdan olumsuz etkilenir. Barok modelinden bir diğer farkı da belgeleme işlemini sürecin doğal bir parçası olarak görüp her aşamada belgeleme yapılmasıdır. Çok net belirlenip anlaşılmış küçük projelerde kullanılması uygun görülse de günümüzde kullanımı azalmakta olan bir modeldir.
   1. **Avantajları:**

* Çok basittir ve anlaşılması kolaydır, bu nedenle yeni başlayan veya acemi yazılım geliştiriciler için faydalı bir modeldir.
* Modelin katılığından dolayı yönetmesi kolaydır. Her aşamanın belirli çıktıları ve özel inceleme süreci vardır.
* Küçük ve kalıp projeler için etkili ve hızlı bir modeldir.
  1. **Dezavantajları:**
* Sadece çok kesin ön gereksinimler varsa kullanılabilir.
* Bakım gerektiren projeler için uygun değildir.
* Son aşamaya gelene kadar elde tutulur bir yazılım üretme şansı yoktur.
* Küçük ve net projeler için kullanışlıyken büyük ve süregelen projeler için uygun değildir.
* Projede meydana gelebilecek değişimlere elverişsiz, katı bir modeldir. Her değişiklik zaman ve maliyette kayıplar doğurur.
* Gelişim ve değişime kapalı olduğundan ve müşteri ürünü ancak yayınladıktan sonra kullanabileceğinden müşteriyi memnun etmek zordur.
  1. **Kullanıldığı Projeler:** Küçük ekipler tarafından küçük ve gereksinimleri net projelerde kullanılır. Büyük ve değişim/gelişim halinde olmayan projeler için idealdir.

1. **V Modeli:** İlerleme adımları isminden de anlaşılacağı üzere doğrusal bir yönde ilerlemez, V şeklindedir. Bu şeklin sol kısmı üretim, sağ kısmı test kısmıdır. Şelale modelinin gelişmiş hali olarak düşünülebilir. Gereksinimlere net şekilde ihtiyaç duyulur ve test aşamaları ön plandadır.
   1. **Avantajları:**

* Doğrulama (verification) ve onaylama (validation) planları erken uygulanır, ayrıca sadece son ürünlerde değil teslim edilebilen tüm ürünlerde uygulanır.
* Takip ve kullanımı kolaydır.
* Hangi aşamanın ne şekilde test edileceği belirgin şekilde gösterilmiş olduğundan yapılanların test ve doğrulanması planlı ve sistematiktir. Bu sayede hatalar daha kolay fark edilip düzeltilebilir.
  1. **Dezavantajları:**
* Şelale modelindeki gibi, ihtiyaçlar ileri aşamalarda anlaşılabilir, maliyet ve zamandan kayıplar olabilir.
* Fazlar arasındaki tekrarlamalara ve aynı zamanda gerçekleşebilecek olaylara imkan tanımaz.
  1. **Kullanıldığı Projeler:** Test aşamalarının önemli olduğu, geri dönüşlerin hemen hiç olmadığı, risk faktörünün göz ardı edilebileceği projelerde kullanılabilir.

1. **Spiral (Helezonik) Model:** Risk analizinin öne çıktığı ve bolca prototip hazırlanan bu modelde ana fikir, projenin erken safhasında riskleri ortadan kaldırmaktır. Geliştirme prosedürü kademeli olarak küçükten büyüğe doğru ilerler. Bu yaklaşım da şelale modelinin düşüncelerini tekrarlarla bir araya getirir. Aynı safhalara geri dönülmesinin zorunludur, şelale modelinde yok sayılan riskler göz önündedir. Her aşamada hedefler belirlenir ve bir müşteriden geri bildirim (feedback) alınır. Bu modelde bir fazdan diğerine geçmek, ilerlemeden önce risklerin tanımlanıp ortadan kaldırılması demektir. Çağdaş modellere son derece yakındır. 4 temel aşaması bulunur:

* **Planlama:** Her aşamada oluşan ara ürünün planlaması ve bir önceki adımda üretilmiş ara ürün ile birleştirilmesidir.
* **Risk Analizi:** Risklerin araştırılıp belirlenmesidir.
* **Üretim:** Ara ürünün üretilmesidir.
* **Kullanıcı Değerlendirmesi:** Kullanıcının ara ürünü sınaması ve değerlendirmesidir.
  1. **Avantajları:**
* Risk analizi ön planda olduğundan riskler projenin erken safhalarında önemli ölçüde azalır.
* Bolca prototip bulunduğundan kullanıcılar sistemi erkenden görebilir.
* Kullanıcı geliştirme süreci boyunca ürüne katkıda bulunabilir.
  1. **Dezavantajları:**
* Karmaşık bir modeldir, spiral sonsuza dek gidebilir.
* Çok fazla ara adım olduğundan belgeleme de çoktur.
* Risk analizindeki başarısızlık tüm projeyi zedeleyebilir.
* Kararlaştırılan bütçe ve zaman sınırını karşılayamama riski vardır, maliyeti yüksek bir modeldir.
  1. **Kullanıldığı Projeler:** Prototiplerin ve risk analizinin bol olması gereken büyük çaplı ve büyük bütçeli projelerde, büyük ekipler tarafından kullanılır.

1. **Artımsal Geliştirme (Prototipleme) Modeli:** Şelale modelinin yaklaşımına dayanır ve müşteri geri bildirimlerine odaklanır. Başlangıç gereksinimleri belirlendikten sonra geliştiriciler ilk ürünü sunar. Ürün son haliyle değil, ara ürünlere bölünerek kullanıcıya sunulur ve her bir ara ürün, bir öncekinin üzerine özellikler konularak çıkarılır. Bu sayede çalışanlar da her artırımdan sonra bu uygulama alanında daha çok deneyim kazanmış olurlar. Buradaki ara ürünlere prototip de denilebilir. Bu prototipler projenin başarısız olma ihtimalini de önemli ölçüde azaltır.
   1. **Avantajları:**

* Yazılım geliştiricileri ile müşteriler geliştirme kısmında güçlü bir iletişim halinde olacağından çok iyi ve elverişli bir ortamda çalışılır ve son ürünün eksikleri az olur.
* Potansiyel riskler daha ilk prototiplerde tespit edilebileceğinden projenin başarısızlık riski önemli ölçüde azalır.
* Gereksinim analizinde ve ürün değerlendirmesinde başarılıdır.
  1. **Dezavantajları:**
* Müşterinin ürün hazırlanmasındaki fazla söz hakkı yazılım ekibini rahatsız edebilir.
* Son sürümün çıkması uzarsa proje bütçesi çıtayı aşabilir.
* Değişiklik sayı ve miktarının çok olması iş akışını etkiler.
  1. **Kullanıldığı Projeler:** Kısa sürede yazılımın ilk meyvesini almak, elde tutulur bir sonuca ulaşmak isteyen yazılımcılar ve projeler için idealdir.

1. **Evrimsel Geliştirme Modeli:** İlk tam ölçekli modeldir, en çok gereksinimlerin netleştirilemediği durumlarda kullanılır. Gereksinimi anlamada fayda sağlar. Düzenli bir ürün oluşumu yoktur, diğer modellere göre daha yavaştır.
   1. **Avantajları:**

* Risk ve hata oranı azdır.
* Gereksinimleri anlamayı kolaylaştırır.
* Erken aşamalarda risk yaşanma şansı düşüktür.
  1. **Dezavantajları:**
* Düzenli ürün oluşumu yoktur, süreç izlenmesi zordur.
* Değişiklikler denetlenemediğinden bakımı zor ve maliyetlidir.
  1. **Kullanıldığı Projeler:** Proje gereksinimlerinin net şekilde belirlenemediği ve süre sıkıntısı olmayan projelerde kullanılır. Coğrafi olarak birbirinden uzak ekiplerin çalışmasına uygundur.

1. **Kodla ve Düzelt Modeli:** Herhangi bir plan, program olmadan kodlamaya girişilen bir modeldir. Genelde işinde acemiler tarafından kullanılır. Belgeleme (dokümantasyon) bulunmadığından hata tespiti ve bakım çok zordur. Geri dönüş (feedback) olmadan hedefe dümdüz gidilir. Her ne kadar basit ve ucuz gözükse de en maliyetli modeldir.
   1. **Avantajları:**

* Hiçbir planlama veya belgeleme olmadığından gayet hızlı şekilde ilerler.
* Uzmanlık gerektirmez, hemen herkes tarafından kullanılabilir.
  1. **Dezavantajları:**
* Belgeleme ve planlama olmadığından hata tespiti ve bakım oldukça zor ve masraflıdır.
* Kodları düzeltmek veya bakım yapmak yine zor ve maliyetlidir.
* Tahmin edilebilir bir sonucu yoktur, bitiş tarihi de istenen ürünü verip veremeyeceği de belirsizdir.
  1. **Kullanıldığı Projeler:** Küçük çaplı, bakım istemeyen, kolay projelerde kullanılır. Öğrenciler tarafından da bolca kullanılır fakat önemli ödev veya projelerde kullanılması doğru değildir.

1. **Çevik (Agile) Yazılım Geliştirme:** Adından da anlaşıldığı gibi, yinelemeyi (iteration), hızı ve takım çalışmasını son derece önemser. Proje küçük parçalara ayrılır ve her parça ayrı bir projeymiş gibi işlenir. Bu yönüyle böl ve fethet (divide and conquer) stratejisine benzetilebilir. Her yineleme sonucunda müşteriye bir ürün verilir, bu ürünlerin hızı ve sıklığı yazılım takımlarının uyum ve performansına bağlıdır.

**Prensipler:**

* Müşteriye sık sık ve seri yazılım sunulmalıdır.
* Ekip elemanları arasında sağlıklı ve birebir iletişim olmalıdır.
* Teslim edilen yazılımlar mümkün olduğunca seri ve kaliteli olmalıdır.
* Hemen her adımda yazılımda değişiklik kabul edilir.
* Ekibin birbiriyle sağlıklı ve yüz yüze görüşmesi gerekir.
* Basit ve çalışan yazılım önceliklidir.
* Ekipler kendi kendini organize edebilmeli, hatalarının farkına varıp düzeltmelerde bulunabilmelidirler.
* Değişen şartlara iyi adapte olunabilmelidir.
  1. **Avantajları:**
* Müşterilerin değişen gereksinimlerine kısa sürelerde cevap verebilir.
* Kısa döngüler çalışanların motivasyonunu yüksek tutar ve verimi artırır.
* Değişime daima açıktır.
* Zaman ve maliyetten tasarruf edilir.
* Kalite ve başarım artar.
  1. **Dezavantajları:**
* İhtiyaçlar sürekli değiştiğinden çok çalışma gerektirir.
* Bazı takımların diğerinden daha iyi olması takım arası çatışmalar doğurabilir.
* Takımdaki elemanlar arasında uyumsuzluk olma ihtimali vardır.
* Kurumsal yapılarda uygulaması kolay değildir.

Çevik Yazılım Geliştirmenin de farklı metodolojileri bulunuyor. Bunların en yaygınları olan Extreme Programming ve Scrum. Şimdi bu iki metodolojiye değinelim.

* **Extreme Programming (XP):** Yazılım gereksinimlerinin maliyetini azaltmayı amaçlar. Müşterinin yazılım geliştirme sürecinde büyük katılım göstermesi amaçlanır. Bu sebeple dengesiz ve kararlaştırılamamış atmosferlerde başarı sağlar. Grup iletişimine büyük önem verir ve geri dönüşler oldukça fazladır.

XP’nin 12 farklı pratiği bulunur. Bunlar planlama oyunu, ekipte müşteri, önce test, basit tasarım, çiftli programlama, sürekli entegrasyon, kısa aralıklı sürümler, yeniden yapılandırma, ortak kod sahiplenme, metafor, kodlama standardı ve haftada 40 saattir.

Bu modelin 4 temel değeri bulunur:

1. **İletişim:** Gerek yazılım ekibinin kendi içindeki, gerekse ekibin kullanıcılarla iletişi yazılımın başarıya hızlı ve problemsiz ulaşmasında çok önemli bir rol oynar. XP, iletişime büyük önem verir ve bu problemleri ortadan kaldırmayı amaçlar.
2. **Basitlik:** Yapılması zorunlu olan işlerin yapılıp fazlalıklardan kaçınmayı, en temel gereksinimleri karşılamayı hedefler. Yine de basit olmaya çalışırken önemli detaylar gözden kaçırılmamalıdır.
3. **Geri Bildirim:** Müşteriden gelen geri bildirim (feedback), yazılım ekibine büyük kolaylık sağlar. Özellikle de sürecin sonunda değil de süreç boyunca gelen geri bildirimler, son versiyonun mümkün olduğunca istenen şekilde olmasını sağlar. Ayrıca ileride oluşabilecek hatalar da geç kalmadan engellenmiş olur.
4. **Cesaret:** 4 temel değer arasında en zoru cesarettir. Daha sonuç alınmadan bırakılan projeler, başarısız olmasından korkulup vazgeçilen projeler başarısız olanlardan daha kötüdür çünkü bir projenin sona gelmeden başarısız olunacağı veya yapılamayacağı kesin olarak bilinemez, tabi 10 birim zaman gerektiren proje için 1 birim zaman varsa o ayrı konu. “You miss 100% of the shots you don’t take” der Wayne Gretzky, bu söz yaşamda karar alıp yapmadığımız işlerin hiçbirini başaramayacağımızı da söyler bize. Yani harekete geçmek, yerinde durmaktan iyidir çoğu zaman.

**Extreme Programming Avantajları:**

* Kaliteli bir yazılım oluşturmak için modern yöntemler kullanılır.
* Müşteri katılımına önem verilir.
* Mantıklı plan ve program oluşturulur.

**Extreme Programming Dezavantajları:**

* Müşteriler için pahalıya mâl olan sık aralıklarla toplantı gerektirir.
* Geliştiriciler için her seferinde benimsenmesi gerçekten zor olan çok fazla geliştirme değişikliği gerektirir.
* Adından da anlaşılacağı üzere çok fazla emek ve çalışma ister.

**Extreme Programming Kullanım Alanları:** İşlevselliği sabit olması gerekmeyen hemen her durumda kullanılabilir. Zaten neredeyse tüm yazılım geliştirme modellerinin özelliklerini barındırdığından bu açıdan avantajlıdır. Karmaşık ve riskli durumlarda kullanılması özellikle tavsiye edilir. Zaman yönünden de en hızlı sonuç veren modellerdendir.

* **Scrum:** Neredeyse her projeye uygulanabilen açık ara en esnek yazılım geliştirme modelidir. Bu nedenle değişen gereksinimli projelerle kolayca ilgilenir. XP için söylediğimiz hemen her özelliği scrum için de söyleyebiliriz, ayrıca dezavantajları XP’den daha azdır. Günümüzde en çok tercih edilen yazılım geliştirme modelidir. Adı, Rugby sporundaki bir hücum taktiğinden gelir. Scrum yönteminde de bu taktikteki gibi takım arkadaşları hep birlikte çalışır.

Genel bir proje yönetim yaklaşımı olduğundan sadece yazılım geliştirmede değil, hemen her yerde kullanılabilir. Karmaşık işleri “sprint” denilen küçük parçalara ayırarak yapmak esastır. Burada da yine “divide and conquer” stratejisine benzer bir yaklaşım bulunur.

Scrum’da 3 temel kavram bulunur:

1. **Roller**

* Ürün Sahibi
* Scrum Yöneticisi
* Scrum Takımı

1. **Toplantılar**

* Sprint Planlama
* Sprint Gözden Geçirme
* Günlük Scrum Toplantısı

1. **Araçlar**

* Ürün Gereksinim Dokümanı
* Sprint Dokümanı
* Sprint Kalan Zaman Grafiği

**Scrum Avantajları:**

* Kararları vermekten sorumlu olan ekip, onları her zaman tetikte güncellemeler ile birlikte ilerler.
* İş gereksinimi belgesi önemsiz kabul edilir.
* Günlük toplantılar geliştiricinin bireysel verimliliğini ölçmesine yardımcı olur. Bu durum, ekip üyelerinin verimliliklerinin artmasına yardımcı olur.
* Zaman ve paradan büyük miktarda tasarruf edilir.
* Karmaşık projelerde etkilidir.
* İletişim ve takım çalışması önemli olduğundan hatalar erkenden fark edilip düzeltilir.
* Çok esnek olduğundan hemen her değişime uyum sağlayabilir.

**Scrum Dezavantajları:**

* Tahmin edilen proje maliyet ve zamanı tutarlı olmazsa sıkıntı yaşanır.
* Küçük ve hızlı ilerleyen projeler için idealken büyük projeler için uygun değildir.
* Ekip üyeleri deneyimli olmalıdır, acemi üyelerle başarıya ulaşması çok zordur.

**Scrum Kullanım Alanları:** Günümüzde en çok kullanılan modeldir, yazılım dışında hemen her alanda da kullanılır. Özellikle karmaşık, büyük ölçütte olmayan, zaman sınırı bulunan projeler için idealdir.

**Scrum’ın Bu Kadar Popüler Olmasının Sebepleri**

* Çünkü Scrum En Esnek Modeldir:

Bir geliştirme ekibi öngörülemeyen zorluklarla karşılaşırsa çeşitli değişiklikler ve ayarlamalar yapılabilir. Projeye başlamadan aşırı plan yapıp zaman kaybetmek yerine ilerledikçe uyum sağlamayı öngörür.

* Çünkü Scrum Hızlı Sonuç Verir:

Scrum, erken ve düzenli geri dönüşler sağlayan kısa koşular (sprint) üzerine kuruludur. Bu durum riskleri azaltır ve hızlı sonuçlar almayı sağlar. Müşteriler sonuçları daha erken görür ve gerçekleştirim adımlarına düzenli olarak dahil olur.

* Çünkü Scrum Takım Odaklıdır:

Scrum, ekipleri motive etmek ve ihtiyaç duyulan çalışma ortamını sağlamak için tasarlanmıştır. Ekipleri motive etmek için çeşitli uygulamalar bulunur. Bunlardan bazıları işbirliği, öz-yönetim (self-management) ve hızlı sonuçlardır. Hızlı sonuç almak müşteriyi mutlu ettiği kadar yazılım ekibini de motive eder.

* Çünkü Scrum Müşteri Memnuniyetini Ön Planda Tutar:

İş hayatında, son sözü müşteri söyler. Scrum modeli de müşteri memnuniyetini ön planda tutar ve müşterileri mutlu etmeyi amaçlayan birkaç temel teknik bulunur. Bunlar müşteri katılımı, artımlı ödemeler ve düşük risktir. Ürün piyasaya aşamalı olarak sürüldüğünden, ödemeler de artımlıdır.

**Son Olarak**

Yazılım geliştirme modelleri birbirine benzerlikler gösterir ve genelde birlikte kullanılır. Bir projede hangi modelin kullanılacağı, projenin büyüklüğü, süresi, gereksinimlerinin netliği, maliyeti, hızlı sonuç isteyip istememesi gibi faktörlere göre belirlenir. Günümüzde en çok çevik modeller kullanılmasına rağmen diğer projeler de zaman zaman kullanılmaktadır, bu durum bazen şartlardan kaynaklanırken bazen de tembellik veya yanlış analizden kaynaklanabilir. Siz siz olun yazılım geliştirirken doğru modeli seçin, bu seçimi başkası yapıyorsa da yanlış seçim yapıldığını düşündüğünüzde uyarıda bulunmaktan çekinmeyin. Bu sayede ileride size veya ekibinize büyük katkı sağlayabilirsiniz.

Yazının işinize yaramış olması umuduyla, hoşça kalın…

**Kaynaklar**

* Doç. Dr. Deniz Kılınç, İzmir Bakırçay Üniversitesi BİL102 Dersi 2. ve 3. Hafta Ders Notları
* <https://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-disadvantages/#anchor12>
* <https://www.intellectsoft.net/blog/top-12-software-development-methodologies-you-should-know/>
* <https://acodez.in/12-best-software-development-methodologies-pros-cons/#Extreme_Programming_Methodology>
* <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>
* <https://hygger.io/blog/a-brief-introduction-to-extreme-programming/#:~:text=If%20there%20is%20no%20time,be%20slower%20and%20less%20efficient.&text=Most%20of%20the%20researchers%20and,use%20of%20Extreme%20Programming%20method.>
* <https://e27.co/scrum-popular-among-developers-5-unconditional-benefits-development-approach-20170516/>
* <https://www.professionaldevelopment.ie/who-uses-scrum>
* <https://blog.soprasteria.co.uk/2016/01/22/four-reasons-why-scrum-is-so-popular/>