**Yazılım Yaşam Döngüsü ve Süreç Modelleri**

Tüm canlıların kendine has bir yaşam döngüsü vardır. Örneğin insanlar doğarlar, büyürler, yaşlanırlar, ölürler, toprak olurlar ve başka hayatlara kaynak olurlar. Buna benzer şekilde hayatın her alanında bir döngü vardır. Bilgisayar mühendisleri olarak bizler de bir proje yapacağımız zaman, bir döngü belirler ve yazılımı bu döngü çerçevesinde geliştirir ve sunarız. Dolayısıyla yazılımların da kendilerine ait bir yaşam döngüleri olduğunu bilmeliyiz. Yazılım yaşam döngüsü (Software Development Life Cycle) genel olarak; bir yazılımın geliştirilme ve bakım süresince uygulanmak üzere belirlenen adımların bütünüdür. Unutulmaması gereken şey ise yazılımların işlevleri ve ihtiyaçları sürekli geliştiği ve değiştiği için yazılımın tek yönlü bir doğru parçası olmadığı, kendini tekrar eden ve yenileyen bir döngü olduğudur.

Yazılım geliştirme aşamaları beş temel bölümden oluşur. Bunlar planlama, analiz, tasarım, gerçekleştirme ve bakımdır.

**Planlama**, ilk olarak gereksinimlerin belirlenmesi ile başlar. Bu adımda müşteri dinlenir, istekler tam ve net bir şekilde belirlenir. Daha sonra üretim aşamasında kullanılacak olan donanım, yazılım ve personel gibi ihtiyaçların miktarları ve türleri belirlenir. Ve projenin planı oluşturulur.

**Analiz** aşamasının amacı belirlenen gereksinimlerin ve yazılım işlevlerini netleştirmek ve bunları belirli bir sistemle dokümante etmektir.

**Tasarım:** Analiz aşamasında belirlenen gereksinimleri karşılayacak olan yazılımın; planlama ve analiz **aşamalarından** faydalanarak bir projesi çizilir. Yazılımda bulunacak özelliklerin nasıl olması gerektiği ve bunların nasıl gerçekleştirileceği belirlenir. Tasarım aşamasında herhangi bir kodlama söz konusu değildir. Tasarım aşaması 2 bölümden oluşur: mimari tasarım ve ayrıntılı tasarım. Mimari tasarımda genel olarak bir plan yapılır ve modüller belirlenir. Ayrıntılı tasarımda ise veri tabanı tasarlanır, kullanılacak olan algoritmalar belirlenir, programın hangi dil veya dillerde yazılacağı kararlaştırılır ve bunlar gibi detaylar belirlenir.

**Gerçekleştirme** aşamasında ise analizi ve planlaması bitmiş olan program için kod yazma işlemi gerçekleştirilir. Bu aşama kodlama, test etme ve kurulum olarak üçe ayrılır. Kodlama esnasında ve sonrasında yapılan önemli bir aşamada test etmedir. Yazılımı mümkün oldukça erken test ederek zaman, para ve prestij gibi kayıplar yaşamamak temel amaçtır. Daha sonra da ürün piyasaya veya müşteriye sunulur.

Teslimden sonra **bakım** aşaması başlar ve programın yaşamı boyunca devam eder. Bakımın temel amacı var olan hataları gidermek veya oluşabilecek hataları önlemek ve programda eksik olan ya da ihtiyaç duyulan yeni özellikleri ekleyerek ürünü geliştirmektir.

Yazılım yaşam döngüsünün daha aktif olması ve en iyi sonucu verebilmesi için yazılım geliştirme süreç modelleri (Software Process Models) kullanılır. Bu modeller yazılım geliştirmenin zorluklarıyla başa çıkabilmek, geliştirmeyi sistematik hale getirmek amacıyla ortaya çıkmıştır. Modellerin ortaya çıkmasında bulundukları dönemin donanımsal ve yazılımsal gelişmişlikleri ile sektörün ihtiyaçları büyük rol oynamıştır.

Süreç modelleri temel olarak 3’e ayrılırlar. Bunlar: düzenleyici süreç modelleri, birleşik süreç modeli ve çevik yazılım süreç modelleri’ dir.

**Düzenleyici Süreç Modelleri**

**Kodla ve Düzelt (Code and Fix):** Genellikle kesinleşmemiş olan bir ürün fikriyle başlar ve ürün hazır olana kadar kodlama devam eder. Çok küçük projeler ya da kısa ömürlü prototipler için uygundur. Herhangi bir planlamaya ihtiyaç duymaz. Uzmanlık gerektirmediği için herkes bu modeli kullanabilir. Ancak bu model genel olarak kontrolsüzdür. Kaynak planlaması yoktur ve bitiş süresi belirsizdir. Ayrıca dokümantasyon mevcut olmadığı için hataları bulması zordur ve bakım yapılabilirliği düşüktür. Sonuç olarak öğrenciler ve bireysel geliştiriciler için uygun fakat takım çalışmaları için faydasızdır.

**Barok Modeli:** Eski bir model olup, yaşam döngüsünün temel adımları doğrusaldır ve aşamalar arası nasıl geri dönüş yapılacağı belirsizdir. Bu model, gerçekleştirme aşamasına daha fazla önem vermektedir. Dokümantasyon, yazılan programın test aşamaları bitirildikten sonra yapılır. Günümüzde ise bu işlem döngünün başından sonuna kadar sürekli yapılır. Bu gibi çeşitli sebeplerden dolayı günümüz projelerinde tercih edilmeyen bir modeldir.

**Şelale (Waterfall) modeli**: Çağlayan modeli olarak da bilinen bu yöntemde SDLC aşamaları lineer bir şekilde uygulanır. Kullanımı ve anlaması basit, yönetimi kolaydır. Küçük ve gereksinimi iyi anlaşılmış projelerde etkili çalışır. Barok modelin aksine belgeleme işi üretimin doğal bir parçası gibidir ve başından sonuna kadar devam eder. Ve yine barok modelin tersine burada aşamalar arası geri dönüşlerin nasıl yapılacağı tanımlıdır. Aşamaların en az birer kez tekrarlanması ile gerçekleştirilir. Ancak çağlayan modeli daha çok statiktir yani değişimlere elverişsizdir. Büyük yazılım projelerinde sürekli farklı fikirler ve ihtiyaçlar ortaya çıktığı için geri dönüşler zordur. Müşterinin istekleri ve sistem gereksinimleri çok detaylı ve net bir şekilde belirlenmelidir. Şelale modeli yazılım projeleri için elverişsiz olmasına karşın temel bir model olması, basitliği ve diğer modellere temel teşkil etmesi açısından önemlidir. Günümüzde ise kullanımı gittikçe azalmaktadır.

**V Modeli (V- shaped Model):** Süreç adımları kodlama aşamasından sonra yukarıya doğru eğim aldığı ve V şeklini oluşturduğu için bu ismi almıştır. Şelale modelinin biraz daha gelişmiş halidir. Sol taraf üretim, sağ taraf ise sınama bölümüdür. Üst ve alt seviye tasarım vardır. Üst seviye daha genel bir tasarımdır. Alt seviye de ise girdiler-çıktılar, beklentiler gibi daha detaylı bir tasarım mevcuttur. Bu yöntemde proje takibi kolaydır. Modelin kullanımı genel olarak basittir. Fakat aşamalar arası tekrarlamaları kullanmaz. Risk çözümleme ile ilgili aktiviteleri içermez.

**Helezonik (Spiral) Model:** Planlama, risk analizi, üretim ve kullanıcı değerlendirmesi olarak 4 ana aşamadan oluşur. Ve bir spiral oluşturacak şekilde dönerek bu aşamalar tekrar eder. Şelale modelinde yok sayılan riskleri göz önünde bulundurur. Proje çevrimlere ayrılır ve her bir çevrimin riskleri ayrı ayrı ele alınır. Çağdaş modellere son derece yakın bir modeldir.

Modelin en yararlı özelliklerinden biri de prototipler sayesinde kullanıcılar sistemi erkenden görebilirler. Riske duyarlı yaklaşımı potansiyel zorlukları engeller. Hataları erken gidermeye odaklıdır. Projelerin büyüklüğüne göre prototip sayısı artırıp, azaltılabilir. Bunların yanında kompleks bir modeldir. Spiral sonsuza kadar gidebilir. Küçük ve düşük riskli projeler için maliyeti fazladır. Çünkü devamlı olarak prototip çıkarmak ve dokümantasyon yapmak gereklidir.

**Evrimsel Geliştirme (Evolutionary Development) Modeli:** Bu model ilk tam ölçekli olan modeldir. Projenin başarıya ulaşması ilk evrimin başarısına bağlıdır. “Ne istediğimi bilmiyorum ama görsem tanırım.” mantığına dayanır. Gereksinimi anlamayı basitleştirir. Risk ve hata oranı azdır. Zayıf yanıysa değişiklik denetimine sahip olmamasıdır. Bu sebeple bakımı zordur ve maliyetlidir. Müşteri geri bildirimleriyle ilerlediği için yavaştır. Düzenli bir ürün oluşumu yoktur. Kısa süreli sistemlerde ya da büyük bir sistemin parçaları (ör. Kullanıcı Arayüzleri..) için kullanılabilirler. İki çeşit evrimsel geliştirme vardır: keşifçi geliştirme (exploratory development) ve atılacak prototipleme (throw-away prototyping).

**Artırımsal Geliştirme Modeli (Incremental Development):** Planlama, analiz ve tasarım aşamalarından sonra bir çekirdek yapı oluşturularak bir sürüm üretilir. Ve sistem artırılması yapılarak devamlı bir şekilde gelişmiş sürümler çıkar. Program hazır olana kadar bu döngü devam eder. Her teslimle birlikte müşteriye bir ürün geldiği için sistem erken aşamalarda işlevsellik kazanır. Üzerine katarak ve gelişerek ilerlediği için projenin tamamen batması önlenmiş olur yani başarısızlık riski azalır. Bu döngü sayesinde sistem daha fazla test edilme imkânı bulmuş olur. Böl ve yönet (Divide and Conquer) yaklaşım tarzına uygun bir modeldir. Dezavantajları ise deneyimli personel gerektirir. Artırımları tanımlamak için tüm sistemin tanımlaması gerekmektedir. Artırımların kendi içlerinde tekrarlamalar yapmasına izin vermez.

**Prototipleme (Prototyping):** Modelin mantığı isminden de anlaşılacağı üzere bir prototip üretip üzerine katarak geliştirmektir. Gereksinimlerin (girdiler, çıktılar, işlemler vb.) doğru anlaşılması önemlidir. Bu modelde hızlı planlama, tasarlama ve kodlama öne çıkar. Bu işlemlerden sonra çıkan ürün (prototip) müşterinin kullanımına ve değerlendirmesine sunulur. Müşterinin değerlendirmelerine göre prototip üzerine çalışılır ve tekrar müşteriye sunulur. Kullanıcının isteğine yaklaşmış olan son prototipe göre değerlendirmeler yapılır ve neler yapılacağı konusunda anlaşmaya varılır. Doğrusal modelin döngüye çevrilmiş halidir. Sınırlı sayıdaki tekrardan sonra, yazılımın son hali müşteriye teslim edilir.

En büyük avantajı kullanıcı, sistem gereksinimlerini sürekli görebildiği için yanlış anlaşılmaları önler ve risk kontrolü sağlanmış olur. Ayrıca istenen yeni gereksinimlerin eklenmesini kolaylaştırır. Dezavantajlarına gelirsek, acele edildiği için dokümantasyonu olmayan hızlı ve kirli (quick and dirty) prototipler ortaya çıkar. Hedefler net değilse güvenlik açığı problemleri ortaya çıkabilir. Düzeltme adımı atlanırsa düşük performansa neden olabilir. Müşteri prototipin son ürün gibi olması beklentisinde olabilir.

**Yeniden Kullanıma Yönelik Geliştirme (Re-use Based Development):** Son yıllarda popülaritesi artan bir yaklaşım türüdür. Programı gerçekleştirecek kuruluş tarafından daha önceden yazılmış veya dışarıdan hazır temin edilmiş yazılımların kullanılmasını temel alır. Kuruluşların olgunlukları ne kadar fazla olursa bu modeli kullanmaları o kadar kolaylaşır. Bu modeli kullanmak için ciddi birikime (önceden yazılmış farklı yazılımlara) ve tecrübeye sahip olmak gereklidir. Avantajlarına gelecek olursak önceden oluşturulmuş sınıflar tekrar kullanılabilir ve böylece kısa sürede yazılım geliştirebilmek mümkündür. Modelin kendisi basit ve anlaşılırdır. Ancak gereksinimleri anlamak güç olabilir. Belirli bir birikim ve uzmanlık yoksa zorlanılır. Maliyeti yüksektir. Ve tüm bunlara rağmen başarım garantisi yoktur. Bu yaklaşımın önemi halen artmaktadır ama kullanımı limitlidir.

**Birleşik Süreç (Unified Process)**

Nesne tabanlı (OOP) yazılım geliştirmek için var olan yöntemlerin kullanılmasıyla edinilen deneyimlerle bu süreçlerin en iyi özellikleri bir araya getirilerek bütünleştirilmiş bir yazılım geliştirme sürecidir.

Yinelemeli (Iterative): Problemdeki istekler bir bütün olarak görülmez. Problemdeki hedeflerin bir kısmı ele alınır ve sınanmış bir ürün ortaya konulur. Her ürün oluşumunun sonunda bir sonraki yinelemeye (iterasyon) geçilir ve yeni istekler ele alınarak geliştirilir. Her iterasyon sonunda hedeflenen ürüne yaklaşılır. Her bir iterasyona ayrı bir proje gibi bakılır ve tüm aşamalardan geçirilerek sınanmış ve çalışır bir ürün elde edilir.

Artırmalı ve Evrimsel (Incremental and Evolutionary): Her yinelemenin ardından farklı ve yeni istekler ele alındığı için elde edilen ürünlerin özellikleri artarak gelişir ve istenilen ürüne daha fazla yaklaşılmış olunur.

Risk Güdümlü ( Risk-Driven): İlk iterasyonda en riskli kısımlar ele alınmalıdır. Böylece proje daha ilerlemeden oluşabilecek büyük problemler görülür ve gerekli önlemler alınabilir. Örneğin bütçe, personel miktarı veya zaman planı gibi değişkenler ayarlanabilir.

Avantajlarına gelecek olursak, proje adım adım geliştiği için değişen isteklere uygundur. Riskler erkenden belirlenip giderilebilir. Her iterasyonda deneyim kazanılır. Sürekli ürün elde edildiği için takımdaki moral ve şevk artar. Tüm bunların yanında karmaşık bir sitemdir. Dokümantasyonu çok iyi yapılmalıdır. Ve dokümantasyon yükü ağır olduğu için maliyet fazla olabilir.

**Çevik Yazılım Geliştirme Süreci (Agile Programming)**

Yenilemeli (iterative) geliştirmeyi temel alan bazı yazılım geliştirme metodolojilerine dayanarak harmanlanmış, yenilikçi bir yazılım geliştirme süreçleri topluluğudur. “Heavyweight” olarak da bilinen eski, yavaş ve bürokratik sistemlere tepki olarak geliştirilmiştir. Agile metotları genellikle takım çalışmalarının, sık denetimlerin, düzenli ürün sunmanın ve müşteri ihtiyaçlarıyla firmanın yeteneklerinin uyumlu bir şekilde bir araya gelerek hızlı ve kaliteli bir biçimde ortaya ürün koyan süreçlerdir. “İlerlemenin en iyi göstergesi, çalışan yazılımdır” prensibiyle devamlı surette müşteriye bir ürün sunulur. Takımların kendi kendilerine organize olmaları önemlidir. Bunun için düzenli aralıklarla, takımlar kendilerini değerlendirerek nasıl daha etkili olabileceklerine bakar, çalışmalarını buna göre düzenler ve devam ederler.

Avantajlarına gelecek olursak, sık çıktı üretip beğeniye sunularak müşteri ihtiyaçlarına göre revize edilir. Kısa sürede müşteri memnuniyeti sağlanır. Doğal insan eğilimine yatkın olduğundan fazla eğitim gerektirmez, hızlı bir şekilde adapte olunabilir. Değişime açıklık ve esneklik seviyesi yüksektir. Projede planlama ve yürütme bir arada olduğundan plan aşamasında ayrıntıya kaçmak yerine iterasyonun planı yapmak yeterlidir.

Her süreçte olduğu gibi çevik yazılımın da dezavantajları vardır. Kurumsal bir yapıda uygulama zordur. Daha çok bağımsız şirketler tercih ederler. Sürekli değişen ihtiyaçlardan dolayı aşırı çalışma gereklidir. Ürünün başarısı ile proje başarısı ortaktır ve bu yüzden kariyer riski doğurur. Ve takımlar üzerinde hedefe ulaşmak için baskı oluşur.

Uç Değer Programlama (Extreme Programing - XP), Scrum, Çevik Tümleşik Süreç(Agile Unified Process -AUP) ve Özellik Güdümlü Geliştirme (Feature-Driven Development - FDD) günümüzde yaygın olarak kullanılan Çevik Yazılım Metotlarından bazılarıdır.

**Uç Programlama (Extreme Programing - XP):** Tüm gereksinimlerin senaryolar şeklinde oluşturulduktan sonra bu senaryoların işlere bölünerek küçük gruplara paylaştırıldığı bir yöntemdir. Yazılımcılar çiftler halinde çalışırlar. Takım içi görev paylaşımı yoktur. Herkes tasarım, kodlama ve test aşamalarında görev alır. Bir müşteri temsilcisi de geliştirici takımlarla beraber çalışır. Aynı iş üzerinde 2 haftadan fazla çalışılmaz. Kullanıcı gereksinimleri değişken veya tam olarak net olmadığı durumlarda kullanışlıdır. Daha çok küçük ve orta ölçekli projelerde kullanılırlar.

Yazılım geliştirmede basitliği ve verimi sağlayabilmek için XP 12 pratik değeri göz önüne alır. Bunlar; planlama oyunu, ayakta toplantı, ekipte müşteri, kısa aralıklarla yeni sürüm, geriye bakış, metafor, ortak sorumluluk, sürekli entegrasyon, kod standartları, kalıcı tempo, test etme, sade tasarım, yeniden yapılandırma ve eşli programlamadır.

Bu modelin 4 temel değeri vardır: **iletişim**, **basitlik**, **geri bildirim** ve **cesaret**.

**İletişim,** projenin aksamasını önlemede önemli rol oynar. Ekip içi iletişim ve yazılımcı ile kullanıcı arasındaki iletişim devamlı ve yüz yüze olmalıdır. Bu iletişimler hızlı olursa projenin de aksama ihtimali ortadan kalkar.

**Basitlik,** zorunlu ihtiyaçları hedef alarak, esnek ve pratik çözümlerle problemleri çözmektir. Ancak basitliği sağlamak zordur; çünkü basitleştirmek isterken ayrıntı gibi görünen önemli hususları atlamak, başka problemlere sebebiyet verebilir.

**Geri bildirim,** projenin başarıya ulaşması için vazgeçilmez bir ihtiyaçtır. Gerek yazılımcıların yaptıkları birim testleriyle, gerekse müşteriye sunulan sürümlerle sağlanır. Yazılım ekibiyle müşterinin belirli aralıklarla buluşarak değerlendirmeler yapması sayesinde ilerde oluşabilecek hatalar engellenir ve projenin istenilen doğrultuda ilerlediğinden emin olunur.

Son olarak **cesaret**, 4 temel değer arasında en zorudur. Başarısızlık üzerine korkmadan gitmek, başarısızlığın sonucuna değil sebebine ve telafisine yoğunlaşılması gerekir. Başarısızlık durumunda yazılımcılar hız kesmeden devam etmeli, gerekirse baştan başlayabilmelidirler.

**SCRUM:** Scrum, kelime olarak rugby oyununda oluşturulan küçük ekiplere verilen isimdir. Günümüzde çokça tercih edilen bir metottur. Eski yöntemlerdeki gibi projede izlenmesi gereken adımlar belirtilmemekte, bunun yerine basit ama önemli birkaç kuralı sayesinde esnek ama üretken bir işleyiş sunmaktadır. Scrum’ın sağladığı faydalardan biri de, işleyiş sürecini şeffaf hale getirerek aksaklıkları ortaya çıkarır. Böylelikle proje ekibinin hataları fark ederek sürekli olarak iyileştirme yapmalarına olanak sağlar.

Scrum’ın genel olarak çalışma mantığına gelecek olursak, yapılacak olan iş (**Product Backlog**) parçalara bölünür. Daha sonra işin belirli bir parçası (**Sprint Backlog**) ele alınır ve 2-4 hafta süreyle bu işin tamamlanması planlanır. Genelde 4 hafta olan bu süreye “**Sprint**” denir.

Klasik metodojilerde (Waterfall, …) kullandığımız sıralı yaklaşımda: (**Sequential Approach**) gereksinim, analiz, tasarım gerçekleştirme gibi aşamalar vardır. Bu metodolojiler bir aşama bitmeden diğerine geçişe izin vermez. Scrum’ın bize getirdiği **overlapping** (örtüşme) yaklaşımıyla ise tüm sprinte bakıldığında bu 4 aşama örtüşen bir şekilde beraber ilerler. Aşamalar bir bütünmüş gibi hareket edildiği için bir aşamada oyalanarak sıkılma olmaz ve sürekli bir ürün ortaya konulduğu için de müşteri bundan memnun olur.

Scrum’ da 3 temel kavram vardır. Bunlar **“Roller”** , **“Toplantılar”** ve **“Araçlar”** dır.

**Rollere** gelirsek ilk olarak bir **ürün Sahibi (Product Owner)** olmalıdır. Bu kişi illa müşterinin kendisi olmak zorunda değildir. Bir müşteri temsilcisi veya Scrum ekibinin bir parçası da olabilir. Önemli olan bu kişinin müşteriyle iletişim kurabilmesi, müşterinin ne istediğini bilmesidir. Bu kişi ekiple birlikte hareket eder ve projede bizzat yer alır.

**Scrum Yöneticisi’ nin (Scrum Master)** görevi takıma emirler verip yönetmek değildir. Scrum master: takıma yardımcı olan, bir problem olduğunda onu çözen, takımlar ve ürün sahibiyle olan iletişime yardımcı olan, takımın ve organizasyonun Scrum’a adapte olmasını sağlayarak üretkenliğin artmasına katkıda bulunan kişidir.

**Scrum Takımı (Scrum Team)** , birbirleriyle iletişim halinde olan ve programlama işini yapan gruptur. Kendi içlerinde farklı görevleri olabilir ama hedefleri ortaktır. Genellikle 5 ila 10 kişiden oluşur.

**Toplantılar**:

**Sprint Planning** de sprintlerde neler yapılacağı, gereksinimlerin listesi, takımların belirlenmesi, risk analizleri ve maliyet hesaplamaları yapılır.

Tüm sprint boyunca her gün takım üyeleri bir araya gelerek 15 dakika boyunca ayaküstü toplantılar yaparlar. Bu toplantılara “**Daily Scrum Meeting**” denir. Bu toplantılarda önceki gün neler yapıldığı, ne gibi sorunlarla karşılaşıldığı, o gün neler yapılacağı ve nasıl yapılacağı hakkında özet şeklinde konuşmalar yapılır. Ayaküstü toplantı denmesinin sebebi, toplantı denilince akılda şekillenen karşılıklı oturup slaytlar üzerine konuşmalar değil, herhangi bir grup üyesinin masasının başında bile olsa toplanarak takım içi iletişimi kaybetmemek ve birbirine yardımcı olmak için görüş alışverişi yapılmasıdır.

Her sprint ardından bir **Sprint Gözden Geçirme (Sprint Review)** yapılır. Buraya herkes davet edilebilir ve yapılanlar gözden geçirilir. Bu toplantılar uzun sürebilir. Ardından bir sonraki sprinte hazırlık yapılır.

Üçüncü temel değer olan **Araçlar ’ın (Artifacts)** başında **Ürün Gereksinim Dokümanı(Product Backlog)** gelir. Proje boyunca tamamlanması gereken işlerin listesidir. Listeyi **product owner** yönetir. Bu liste genelde **kullanıcı hikayelerinden (user story)** oluşur. Yani kullanıcının isteğine göre yeni gereksinimler eklenir veya çıkarılır.

**Sprint Listesi (Sprint Backlog)**, bir sprint turunda takımın yapması gereken işlerin listesidir. **Product Backlog** ’ın bir alt kümesi olarak düşünülebilir. Bir sprint döngü için seçilen gereksinimler, görevlere bölünerek takımda bulunan üyelere dağıtılır.

**Sprint Kalan Zaman Grafiği (Burndown Chart),** bir sprint turunda görevler yerine getirildikçe yapılan iş ile kalan iş arasındaki bağıntıyı ortaya çıkaran grafiktir. Belirlen sürenin yeterli olup olamadığı, işin gidişatında bir sorun olup olamadığı ve işin zamanında bitip bitmeyeceği hakkında takıma ve yöneticilere bilgi sağlar.

Scrum, uzmanlık gerektiren ve maliyeti yüksek bir modeldir. Ancak kısa sürmesi, kolay uygulanması ve başarı garantisinin yüksek olması nedeniyle günümüzde kullanımı oldukça artmaktadır. Google, Microsoft, IBM ve Yahoo gibi büyük şirketler tarafından tercih edilmektedir.

Yazılım süreç modelleri birbirlerinden tamamen ayrı değillerdir ve çoğu zaman beraber kullanılmaktadırlar. Bir proje için hangi modellerin ve hangi adımların uygulanacağını “**süreç mimarı**” seçer. Bütün modellerdeki güçlü ve zayıf yanları kıyaslayarak projeye en uygun olanları seçer. Bu seçim yapılırken; projenin büyüklüğü, karmaşıklığı, şirketin yapısı, zaman ve maliyet gibi kriterler göz önüne alınır. Günümüzde en çok tercih edilen ve geçiş yapılmak istenen model, Agile Modellerdir. Kısa zamanda, esnek biçimde ve yüksek başarı sağlaması tercih sebeplerinin başında gelir.

Referanslar

* <https://www.youtube.com/watch?v=u2rU8Wss4bw&t=935s>
* <https://www.youtube.com/watch?v=nHv3-VtiP38>
* <https://www.codex.com.tr/yazilim-gelistirme-modelleri>
* <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>
* <http://www.acm-software.com/scrum/>
* <http://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>
* Doç. Dr. Resul Daş , “Yazılım Geliştirme Süreç Modelleri Bölüm-2” ([https://slideplayer.biz.tr/slide/12386328/](https://slideplayer.biz.tr/slide/12386328/%20))
* <https://caglartelef.com/yazilim-yasam-dongusu/>
* <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>
* Yrd. Doç. Dr. Hacer Karacan , “Yazılım Proje Yönetimi” ([https://docplayer.biz.tr/2197805-Yazilim-proje-yonetimi-yrd-doc-dr-hacer-karacan.html#](https://docplayer.biz.tr/2197805-Yazilim-proje-yonetimi-yrd-doc-dr-hacer-karacan.html%23))
* <https://www.bayramucuncu.com/scrum-kavramlari/>

Hayri Rıza Çimen

180601060