Yazılım Yaşam Döngüsü, Modelleri ve Stratejileri

Özet- Yazılım düzeni kurulurken altında belli başlı modeller tanımlıdır ve yazılım ile ilgili bilgi sahibi olmayan bireyler bu olayın sadece yazılım yazarak ortaya çıktığını düşünürler. Aslına bakarsak her projenin altında bir düzen yatar ve bu düzende bilinmeyen birçok detay vardır. Yazılım yaşam döngüsü olarak bilinen ve alt basamakları olup gruplanan bu model sisteminde tahmin edilemeyecek kadar çok sayıda model mevcuttur. Bu modeller, yazılım dünyası üzerinde yol gösterici etki bırakır ve stratejik yöntemler sayesinde projeleri daha verimli bir halde ortaya çıkarma yönünde yardım sağlar. Bu makale, önde gelen ve çok bilinen modeller olan; Waterfall (Şelale) modeli, artımlı geliştirme modeli (Iterative Model), V modeli, spiral (Helezonik) modeli, çevik (Agile) modeli ve çevik modelin bir stratejisi olan kabul gören Scrum metodu hakkında inceleme sonucu oluşturulmuştur. Özellikle, incelenen her modelin ve stratejik yöntemlerin avantajlarının üzerinde daha çok durulmuş, kullanım alanları hakkında bilgi verilip ne gibi yöntemler dahilinde kullanılması gerektiği de belirtilmiştir. Avantajların anlatıldığı gibi avantaj sağlamayan durumların da üzerinde durularak detaylı bir anlatım yazıya alınmıştır.

1. Giriş

Yazılım yaşam döngüsü olarak adlandırılan ve yazılım hayatına yön veren yöntem olarak kabul gören süreçte, birbirinden farklı geliştirme seçenekleri mevcuttur. Framework terimi de bu durumda özelliğini belli eder ve kullanımının ne kadar önemli olduğunu belirtir. Yazılım ve uygulama geliştirmek için oluşturulmuş bu platformda, inşa edilen kullanım arayüzü ve altyapısı sayesinde kullanıcının geliştirici özelliğini daha kolay ortaya çıkarır. Planlama ve içerisinde bulunduğu geliştirme sürecinin yönetimi konusunda kolaylık sağlar. Framework üzerinde geliştirilmekte olan projede, kullanılan platform ne olduğu fark etmeksizin, birbirinden farklı hatalar ve sorunlar meydana gelecektir. Geçmiş yazılım çağlarında, her bilgisayar ve yazılım bilimci bu sorunların üstesinden gelebilmek için uğraş vermiştir. Ancak, sonuç olarak varılmıştır ki hiçbir yol hata yapmanın önüne geçemez ve her bir durum hata düzeltebilirken başka bölgede hata meydana getirir. Bu düşünceler sonucunda bilgisayar bilimciler, büyük ve kapsamlı projelerde kullanılmak üzere ortaya yöntemler çıkarmış ve bunlara model isimleri verilmiştir. Her modelin süreçlerinde gerçekleşen adımlar birbirinden farklı ama proje türüne göre kullanım değişikliği açısından çeşitlilik sağlayacaktı. Yazılım yaşam döngü modelleri olarak bilinen bu modellerde, genel olarak izlenilen bir yol mevcuttur. İlk olarak planlama yapılır ve projenin görselleştirmesi belirlenir. Gereksinimlerin tespiti sonrası analizi yapılarak gerek duyulan ihtiyaçları gruplandırılır. Yazılımsal modelleme yapılarak projenin tasarım olarak ilk adımı atılır ve kodlamaya başlanılır. Önceki adımlarda elde edilen bilgi ve verileri belli bir belgeleme işlemine sokulur ve doküman elde edilir. Projenin sonlarına doğru test aşaması ele alınır ve herhangi bir hata veya hataya neden olacak unsur var mı tespit edilir. Sonuç olarak, elde edilen projenin bakımı yapılır ve son haline getirilerek müşteriye veya komiteye sunulur.

1. Çeşitli Yazılım Yaşam Döngü Modelleri

a) Waterfall (Şelale) Modeli: En eski ve yaşam döngü modellerinin temeli olarak da bilinen bu model, 1970’li yıllarda Winston Royse isimli bir bilgisayar bilimcisi tarafından bir fikir olarak ortaya atılır. Bunun sonucunda yazılım dünyası tarafından kabul görmeye başlar. Waterfall (Şelale) modeli, isminden de mantık kurularak ilerlendiği sürece, yukarıdan aşağıya doğru su akışı gibi bir izlenim bırakır ve en alttaki adıma kadar sırasıyla ilerler. Kullanıldığı bölümlere bakmak gerekirse, baskın olarak askeri alanlarda geliştirilmeye amaçlanan projelerde ve benzer sektörler için kullanılır. Hata payını en aza indirilerek yapılmayı amaçlar. Bu olayın sebebi ise gereksiz masrafı önlemek ve derin planlama ile zor kalınan durumların önüne geçmektir. [1] Kullanılma ve yönetilme süreci kolay ve sadedir ama kullanılırken değişmesi gereken yerlerin üzerinde oynama yapmak projeyi baştan aşağıya yok edebilir, o yüzden risk barındırdığı ve net olmayan unsurların varlığının yer alabileceği söylenilebilir. Projenin sonlarına doğru yazılım aşamasına geçildiği için geç kalındığı düşüncesi akılda bırakabilir ancak süreç bu şekilde işler. Altı adımda oluşan ve her adımı, her modelde olduğu gibi, minimum hata oranı ile yapılabilme sürecidir. İlk adım olarak sistemsel ve yazılımsal gereksinimleri tespit etme gelir. Projenin geniş çaplı, detaylı ve mümkün olduğu kadar doğru bir şekilde ihtiyaçları elde edilir. Projenin tasarımı ile genel bir hat belirlenir ve detaylı tasarlamaya geçilir. [2] Kodlama işlemi başlatılarak yazılımsal olarak ilk adım atılır ve test edilerek hataların tespiti yapılır. Son olarak da projenin gerekli bakımı ile sonlandırılma gerçekleşir. Eğer herhangi bir adımda hataya sebep olacak bulgu veya sorun yaratan hata bulunursa önceki adıma geri dönülerek düzenleme yapılır. Eğer ki, son an değişikliğine denk gelinirse, proje çıkmaza girebilir çünkü Waterfall (Şelale) modelinin amacı mümkün olduğunca tek denemede, sağlam adımlar atarak projeyi sonlandırmaktır. Son dakika değişikliği bu konuda projeyi zora sokabilir. Üzerinde oynama yapıldığında projeyi yok etme riski barındırdığından dolayı büyük, kapsamlı, karmaşık ve nesneye yönelik projeler için uygunluğu söz konusu değildir. Gereksinimler iyi tespit edildiği sürece küçük projeler için kullanılması uygundur. [3] Eğer ki projenin yapılmasını isteyen müşteri, gereksinimlerin listesini hazırda getirmiş ve üzerinde değişiklik yapılması mümkün değilse, doğal yoldan Waterfall (Şelale) modeli kullanılma ihtiyacı doğar.

b) Artımlı Geliştirme (Iterative) Modeli: Öncelikli olarak gereksinimler, belli başlı sıklıklarla, elde edilir ve gerekli adımlar belli sıraya sokularak parça parça işlem yapılır. Waterfall (Şelale) modeline göre daha esnek bir yapıya sahiptir ama bazı önemli noktalarda belirsizlik ön plana çıkar ve Waterfall (Şelale) modeline kıyasla, belirsizlik açısından, anlaşılması zordur. Böl ve yönet stratejisi ile her adımda üstüne biraz daha koyarak geliştirilme açığa çıkar. Modelin adından anlaşılacağı üzere, her adımı arttırarak geliştirme yöntemi ile yenilemeye açık bir teknik denebilir. Normalde, mantık üzerinden gidilirse, kolay olan adımlar daha hızlı biter ve zor olan aşamaları beklemek zorunda kalır ancak bu modelde kolay olan bölümler bittiğinde zor olan bölümü beklemesi gerekmez çünkü böl ve yönet stratejisi bunu gerektirir. Çok fazla çalışana ihtiyaç duyulmadan elde edilen geliştirmelerle hem kullanıcı boyutunu ele alır hem de üreterek tecrübe açısından çalışanlara veya üstünde duranlara kazanç sağlar ancak ne kadar az çalışan olsa bile takım çalışması yine de önemlidir çünkü herhangi bir çalışan kendi benliği ile hareket ederse belgesel olarak projeye zarar verebilir. [4] Takım çalışması sırasında, birbirleriyle aynı yönde ilerleyen bazı adımlarda yanlış anlaşılma ve geliştirme söz konusu olabilir. Bunun sonucunda da projenin geleceği açısından sorun yaratma şansı artar. Tür olarak küçük ve orta dereceli projelerin geliştirilmesi ve tamamlanması için kullanılır. Zaman açısından uzun sürdüğü kanaatine varılabilir ancak esnekliği vasıtasıyla ve maliyetinin düşük olması sebebi ile başarı garantisini de bir müddet etkiler ve garantili iş açışa çıkartır.

c) V Modeli: Şelale modeline kısmi olarak benzerlik gösterir çünkü kullanımı ve yönetimi kolay ve sadedir ancak test aşamasının önemi şelale modeline göre daha fazladır ve katıdır. Her adımın, diğer çoğu modelde olduğu gibi, bir sonraki adıma geçmeden tamamlanması gerekir ancak adımlar doğrusal olarak ilerlemez. Kullanılan projede, esnekliğini ve projedeki edindiği yer ayarını yapmak zordur ve normalden daha fazla maliyet ihtiyacı doğurur. Her adımın kendine özel dağılım süreci vardır. İki doğrulama fazından meydana gelir. Verification (doğruluk) ve Validation (geçerlilik) olarak adlandırılan bu fazlar, gerekli test kontrollerini doğru yapılabilmesi açısından önemlidir. İlk faz olan Verification sonlandığında kod yazılarak test aşamaları için başlangıç yapılır. Diğer modellere kıyasla, yazılım fazından doğan prototip erken zamanda ortaya çıkmaz. [5] Uygulama aşamasında yazılım gelişir ve spiral (Helezonik) modelde olduğu gibi erkenden prototip üretimi oluşmaz. İkinci faz olan Validation, her adımında Verification fazının denk gelen adımlarını kontrol ederek yine ve yeniden test eder. Test sırasında ortaya çıkan belli başlı hatalar, V modeli tarafından kolayca ortadan kaldırılamaz çünkü model düzgün ve anlaşılır bir çözüm yolu sunmaz. Çok kontrollü bir model olan V modeli, bu özelliği açısından şelale modelinin bir üstü ve gelişmişi olarak da tanımlanır. Waterfall (Şelale) modelinde olduğu gibi, ihtiyaçlar iyi tespit edildiği sürece küçük projelerde kullanılması için uygundur.

d) Spiral (Helezonik) Modeli: Öne çıkan en önemli özellik yinelemenin üzerinde çok durulmasıdır çünkü risk açısından büyük olan, maliyeti fazla, karmaşık ve büyük projeler için kullanılır. Diğer modellere kıyasla kodlama adımı ile daha erken karşılaşılır. Tamamlanma süresi ortalama 6-24 ay arasıdır. Dört fazdan meydana gelen bu modelde, öncelikle planlama yapılır. Risk analizi, üretim ve kullanıcının değerlendirmesi takip eder ancak bu şekilde sonlanmaz. Değerlendirme sonucunda dairesel olarak adımlar tekrardan birbirini takip eder ve planlama ile döngü yeniden başlangıç yapmış olur. Her dönemeçte geliştirilme gerçekleşir ve son olarak kullanıcının son değerlendirmesi ile proje sonlandırılır. Risk analizi yapılır diyerek geçmek doğru olmaz çünkü aşamanın ortaya çıkması için ciddi bir profesyonellik ihtiyacı barındırır ve projenin en önemli adımıdır. Bu adıma bağlı olarak projenin kaderi şekillenir ve belirlenir. Maliyet açısından detaylı incelemek gerekirse projenin sağlam olması açısından, birçok adımın da sahip olduğu gibi, prototip üretimi veya gerekli adımların fazlalığı sebebi ile maliyet olayı diğer modellere göre bir kat fazladır. [6] Proje türü olarak büyük projeler tercih edilir. Küçük projelerde kullanılmaz, bunun nedeni ise de küçük projelerde risk analizi veya analiz aşamasında bu modelde olduğu kadar üzerinde durulması gerekilmez. Başarı garantisi açısından yüksektir ancak çok maliyetlidir.

e) Çevik (Agile) Modeli: 1990’lı yıllarda yazılım gelişimi açısından değişime uğramış ve 2001 yılında bir grup bilgisayar bilimcileri tarafından son haline, şu an dünyada herkesin bildiği hale, getirilmiş ve kabul görmüştür. Çevik (Agile) modeli, öncelikle her modelde olduğu gibi ekipte yer alan herkes bir fikir alışverişi yapmak üzere toplanır. Gereksinimler üzerinde durulur ve her bireyin anlaması sağlanır. Projenin gelişimi için plan oluşturulur ancak ayrıntıya girilmez çünkü tekrarlama süreci boyunca projedeki adımların esnekliği sebebiyle değişime açıktır. Dört adım üzerine yoğunlaşılır. Belirtildiği üzere gereksinimler belirlenir, tasarım ve mimari yapısı oluşturulur, yazılımsal çalışmalara başlanılır ve test edilerek geri dönüşüm sağlanır. Sık sık yinelemeye açıktır çünkü adı üzerinde, çevikliğe ve uyum sağlamaya odaklı bir modeldir. Tek sağlam ve oturaklı denemede projeyi bitirmek yerine, bir döngü içinde yinelemeli olarak geliştirilmeyi daha da arttırmayı amaçlar ve projenin iyileşme sürecini daha verimli gerçekleştirir. [7] Bu yüzdendir ki, ürünü ortaya çıkarma açısından zaman kazandırma potansiyeli diğer modellere kıyasla yüksektir. Normalde çoğu modellerde döngü sırasında bir sorun çıktığında veya kontrol amaçlı dönüldüğünde önceki adıma bakılır. Ancak bu modelde gereksinim adımındayken kendinden önceki adıma ve sonraki adıma bakarak gereksinimlerin sağlamlığı kontrol edilebilir. Takım çalışmasının en sağlam ve sıkı görüldüğü modellerdendir. Yazılım dünyasının her alanında kullanılabilir. En uyumlu olarak çalıştığı alan web tabanlı uygulamalardır çünkü yinelemeli özelliği sayesinde herhangi bir adımda ortaya çıkan birbirinden değişik hataların düzeltilmesinde baskın olarak kullanılır. Çok emek ve gayret gerektirir. Nedeni ise gereksinimlerin sürekli değişime açık olmasıdır.

1. Arka plan

SCRUM: Proje yönetimi için kullanılan bir stratejik yöntemdir. 1990’lı yıllarda ortaya çıkmış ve geliştirilebilir herhangi bir durumda, sadece yazılım olması şart değil, kullanılabilir olduğu anlaşılmıştır. Geliştirilmesi gereken veya baştan yaratılan projelerde, karmaşıklığın önüne geçmek adına küçük çaplı birimlere bölerek ilerlemeyi amaçlar. Bu yapısal süreç, çevik (agile) modelinin çok popüler olan bir yönetim tekniği (proje yönetim süreci) olarak bilinir ve uyumsuzluğun önüne geçilebilmek adına yapılan bir stratejidir. [8] Proje süresince geliştirme bölümünde görev alan çalışanlar belli hatalar ile karşı karşıya kalabilirler ve projenin iyi bir şekilde sonuçlanabilmesi adına en iyi çözüm yolunu bulmaları gerekir. Çevik (Agile) yazılım geliştirme modeli, Scrum stratejisi ile bu sorunun ortadan kalkmasını en verimli yoldan çözümler. Her gün düzenli toplantılar düzenlenerek (15 dakika kadar), her çalışanın neler yaptığının takibi sürülür. Bu şekilde ilerleme kaydedilerek, daha denetimli ve verimli ürünler ortaya çıkar. Scrum stratejisinin kendine özgün terminolojik kelimeleri mevcuttur:

1. Scrum takımı (Scrum Team): Takımda her durumda birbirlerini destekleyecek ve arkalarını kollayacak yazılım geliştiriciler mevcuttur. Aktivite olarak her durum planlanır ve nasıl bir harita çizilmiş ise o yolda ilerleme kaydedilir. Tek başına yapılması gerekenlere odaklanıp ilerleme kaydetmek yerine, takım halinde ilerleme kaydetme stratejisi uygulanır. Her takım üyesi, birbirleri ile iletişim halindedir. Takımlar, 5 ile 9 kişiden oluşur, duruma göre daha az ve daha fazla üye de bulunabilir.
2. Ürün sahibi (Product Owner): Projenin gereksinimleri, ihtiyaç olan kısımlarını ve özelliklerinden bahsetmek ve projeyi yapacak insanlara karşı açıklamakla görevlidir. Sahip olacağı ürün hakkında en iyisi olması için çabalar.
3. Scrum yöneticisi (Scrum Master): Projeyi geliştirmekle görevli olan çalışanlara liderlik ederek onlara yol gösterir. Bütün süreçte ortaya çıkabilecek olası hataları ve riskleri yönetir. Geliştirici takımın verimli olduğu sürece engel olabilecek dış etkenlerden ve yorgunluk (tükenmişlik) gibi olası bireysel sıkıntılarda korur ve projenin en iyi şekilde bitmesine olanak sağlar.
4. Ürün Gereksinim Dokümanı (Product Backlog): Proje adına yeni özelliklerin, var olan özellikler üzerindeki değişikliklerin, hata düzenlemelerinin ve altyapı üzerinde yapılan değişikliklerin listesi ile meydana gelen bir birikim. Proje süresince meydana gelen yeni durumlar sonucunda güncelleme gerekebilir. Bu yüzdendir ki devamlı bakımı esastır. Bakım sonucunda elde edilen gereksiz dokümanlar silinerek projede yer kaplamamasına olanak sağlanabilir. Güncel kalabilme durumuna çok dikkat edildiğinden dolayı proje süreci verimli geçer ve kısa süre içerisinde varılması gereken sonuca varılır.
5. Sprint Dokümanı (Sprint Backlog): Ürün gereksinim dokümanı aracılığı ile elde edilen veriyi, işi ve yapılması gereken görevleri ele alır. Projenin son halinde istenilenin ne olduğu üzerine odaklanır. Tablo oluşturulur ve scrum takımındaki üyelerin hangi süreçlerde olduklarını buralarda incelerler. Tablonun üzerinde oluşturulan notlar neticesinde, birbirlerinden bağımsız gibi ilerledikleri gözükse de herkes birbirinden haberdardır. Her üye, diğer her üyeye bağlı iş yapar. Tablo üzerinde, tabloyu oluşturan üyelere bağlı olarak, çoğu zaman; başlangıç yapılmadı, yapılmakta olan ve sonlanan gibi türevler üretilir ve alt başlıklarına hangi kademede olduklarını belirtirler.
6. Koşu Planlama Toplantısı (Sprint Planning Meeting): Projenin sahibi önderliğinde, ürün gereksinim dokümanı scrum takımına sunulur. Bu toplantı içerisinde proje sahibi tarafından ortaya sunulan doküman, her takım üyesinin görev dağılımı yapmasını sağlar. Toplantı sayesinde, koşu (sprint) süreci başlamadan her çalışan ne yapması gerektiğini iyice kavrar. Projeyi ortaya çıkarma sürecinde yapılacak olan toplantıların en uzunu çoğunlukla bu toplantı olur çünkü ürün sahibi ile scrum takımı arasında sprint öncesi ele alınan ilk tanışma budur. Potansiyel olarak ortaya çıkabilecek başarılı ve başarısız durumlar ağırlıklı olmak üzere gecikme yaşanabilir durumlar da konuşulur ve sprint öncesi yapılan ilk toplantı sonlanmış olur.
7. Ayakta Yapılan Günlük Scrum Toplantısı (Daily Stand-Up Meeting): Ayakta yapılmasının amacı kısa zaman almasıdır. Herkes yerlerine yerleşmeden, ayaktayken toplantı gerçekleştirilir. 15 dakikayı geçmez. Bu toplantıda her üyenin neleri yaptığını ve neler üzerinde zorlanıp asistanın yardımına ihtiyaç duyduğunun üzerinde durulur. Çoğunlukla güne başlarken gerçekleştirilen bu toplantıda, ürün sahibi de yer alır. İçerisinde bulundukları sprint sürecinde ne gibi sorunlarla karşılaşabileceklerinin üzerinde de durmaya devam ederler. Her üye önceki günde neler yaptığını ve bugün içerinde ne gibi görevleri üstleneceğini anlatır. Bu durum sonucunda scrum takımının her üyesi, iletişim halinde kalarak düzenli ilerleme kaydeder.
8. Sonuç

Bu makalede; yazılım yaşam döngüsünün ne olduğunun, modeller arasından seçilen en bilindik beş ögesinin detayları ile anlatımının ve scrum proje yönetim stratejisinin neden popüler olduğunun üzerinde durulmuştur. Birden çok makale, veri ve bilgi analizi sonucunda ortaya çıkarılmıştır. Son olarak makalenin üzerinde durduğu konunun vurgulanması amacı ile; proje açığa çıkarılırken oluşturulan Scrum yönetiminde, scrum yöneticisi olarak her toplantının süresinde yapılabildiği kadar kısa yapılması amaçlanması gerekmektedir. Neden olarak ise de her takım üyesinden maksimum verimi elde etmek amaçlandığında, toplantının sinir bozucu bir hale gelmemesi gerekmektedir. Sprint aşamasını iyi yönetilmelidir. Her model için de geçerli olarak, test ve kodlama aşamasında proje geliştiricileri ile ürünün sahibi arasındaki iletişim süresi kısa ve öz tutulmalıdır. Sonuç olarak, proje geliştiricileri ve yöneticiler, ortada bulunan ve üzerinde çözülmesi amacı ile oluşturulmuş problemin en az hata oranı ile çözebilmeleri gereklidir. Aralarında oluşan iletişimin gerekli kuvvete oluşması proje sahibi ve proje üzerinde çalışanlar açısından da önem taşır.

REFERANSLAR:

1. A Dictionary of Computer Science (7th ed.). Oxford University Press. 2016. p. 44. ISBN 9780199688975.
2. Everatt, G.D.; McLeod Jr., R. (2007). ["Chapter 2: The Software Development Life Cycle"](https://books.google.com/books?id=z8UdPmvkBHEC&pg=PA29). Software Testing: Testing Across the Entire Software Development Life Cycle. John Wiley & Sons. pp. 29–58. ISBN 9780470146347.
3. Gray Pilgrim, “Waterfall Model vs Agile”, Website http://www.buzzle.com/articles/waterfall- model- vs-agile.html, Jan, 2012
4. Kay, Russell (May 14, 2002). ["QuickStudy: System Development Life Cycle"](http://www.computerworld.com/s/article/71151/System_Development_Life_Cycle). ComputerWorld.
5. Raymond Lewallen, “SoftwareDevelopment Life Cycle”, 2005
6. Study & Comparison Of SoftwareDevelopment Life Cycle Models Gourav Khurana ,Sachin Gupta
7. ["SDLC Overview: Models & Methodologies"](https://aristeksystems.com/blog/sdlc-overview/). Retrieved 2021-12-12.
8. ["What is Scrum?"](https://www.scrum.org/resources/what-is-scrum). December 24, 2019.

**Muhammed Samed Karakuş – 210601698**

**Linkedin:** [**https://www.linkedin.com/in/samedkarakusceng/**](https://www.linkedin.com/in/samedkarakusceng/)

**GitHub:** [**https://github.com/samedkarakus**](https://github.com/samedkarakus)

**Medium:** [**https://medium.com/@samedkarakus.ceng**](https://medium.com/@samedkarakus.ceng)