
ANKARA ÜNİVERSİTESİ
AÇIK VE UZAKTAN EĞİTİM FAKÜLTESİ

YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜNDE KULLANILAN FARKLI METODOLOJİLER

Pelin OKÇU

MEZUNİYET PROJESİ
BİLGİSAYAR TEKNOLOJİSİ PROGRAMI

ANKARA ÜNİVERSİTESİ
AÇIK VE UZAKTAN EĞİTİM FAKÜLTESİ

HAZİRAN 2023

İÇİNDEKİLER

1. GİRİŞ

Yazılım geliştirme projeleri, yazılım geliştiricilerin belirli standartlarda yazılım ürünlerini geliştirebilmeleri için yazılım geliştirme yaşam döngüsü (software development lifecycle – SDLC, bundan sonra SDLC olarak anılabilecektir) adı verilen teknik yöntemin uygulanması ile yönetilmektedir. Yazılım geliştirme yaşam döngüsü içerisinde, bir yazılım ürünün oluşabilmesini sağlamak amacı ile, geliştirilecek yazılım ürününün içeriğinin anlaşılmasını sağlayacak (analiz aşaması), belirlenen kapsam ve içeriğe göre tasarlanmasını (tasarım aşaması) ve bu tasarıma göre geliştirilmesine (geliştirme aşaması) yol açacak ve en sonunda geliştirilmiş yazılım ürününün kapsam, içerik ve işlevselliğinin test edilmesi (test aşaması) işlemlerinin gerçekleştirilmesi yer almaktadır.

Yazılım geliştirme yaşam döngüsünde, yukarıda sayılmış olan iş ve işlemlerin gerçekleştirilmesi sırasında, geçmişten günümüze kadar farklı yöntemler uygulanarak, yazılım geliştirime projeleri yönetiminde güncellemeler meydana gelmiştir. Yazılım geliştirme yaşam döngüsü içerisinde yer alan iş ve iş süreçleri, günümüzde artık kendi başına değil, yazılım proje yönetimi metodolojileriyle birlikte anılmaktadır.

Proje yönetim metodolojileri, sadece yazılım projelerinin yönetimi için kullanılmasa da (yani yazılım geliştirme işinden farklı içeriğe sahip projeler için de kullanılabilirler) araştırma projemiz kapsamında, detayları verilecek olan metodolojilerin tamamı yazılım geliştirme projelerine atıf yapmaktadır.

"Yazılım Geliştirme Yaşam Döngüsünde Kullanılan Farklı Metodolojiler" araştırma projesi (bundan sonra Proje olarak anılacaktır) ile yazılım geliştirme projelerinde kullanılan temel proje geliştirme metodolojileri ele alınmıştır.

Yazılım geliştirme yaşam döngüsü içerisinde takip edilmesi gereken adımlar ve bu adımların yer aldığı farklı uygulama tekniklerinin, yazılım geliştirime projelerinde işlenme yöntemleri üzerine araştırma yapılmıştır. Araştırma içerisinde, yazılım geliştirme süreçlerinde uygulanan farklı yöntemler üzerine araştırma sonuçları açıklanmış ve bu yöntemlerin birbirlerine göre olan kıyaslamaları sunulmuştur.

Proje ile yazılım geliştirme projelerinde uygulanabilecek proje geliştirme/proje yönetim metodolojileri arasında seçim yapılması gerektiği durumlarda, uygun metodolojinin seçimi için başvurulabilecek bir kaynak olması hedeflenmiştir.

Proje kapsamında, yazılım geliştirme dünyasında yaygın olarak kullanılmakta olan;
Şelale (Waterfall) Metodolojisi, V-Model Metodolojisi, Helezonik/Sarmal (Spiral) Metodolojisi ve Çevik (Agile-Scrum) Metodolojisi ele alınmış, araştırılmış, karşılaştırılmış ve sunulmuştur.

2. YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ

Yazılım geliştirme yaşam döngüsü, farklı proje geliştirme yöntemleri ile takip edilseler de genel süreç mantığı çerçevesinde;

- Planlama Aşaması
- Analiz Aşaması
- Tasarım Aşaması
- Geliştirme Aşaması
- Test Aşaması
- Teslimat/Kurulum Aşaması
- Eğitim Aşaması

bölümlerinden oluşmaktadır.

Planlama, Teslimat/Kurulum ve Eğitim aşamaları, yazılım geliştirme yaşam döngüsünün uygulanması için gerekli olan ön koşul ve tamamlanması için gerekli olan kapatma eylemlerinin icra edildiği aşamalardır. Bu aşamalar yazılım geliştirme yaşam döngüsü içerisinde ana rol oynamasalar bile, yazılım geliştirme projelerinde, uluslararası standartlara uyum konusunda da mutlaka yer verilmesi gereken, projelerin başlatma ve kapanış esnalarında kendince kuralları belirlenmiş çok önemli aşamalardır. Bu aşamaların detayları bu Proje kapsamında yer almayacak fakat aşağıdaki alt bölümlerde genel özelliklerine yer verilerek, yazılım geliştirme yaşam döngüsünün uygulandığı farklı geliştirme metodolojileri anlatılırken bahsi geçecektir.

2.1. Analiz Aşaması

Bir yazılım ürünün/projesinin geliştirilmesi için geliştiricinin ihtiyaç duyduğu bazı bilgiler bulunmaktadır. Bu bilgiler içerisinde; ürünün/projenin amacı, kapsamı, kullanıcıları, kullanıcıların gerçekleştirecekleri işlemler, farklı yazılım ürünleri ile entegrasyon ihtiyacı olması, entegrasyon içerikleri, kullanıcıların girdi ve çıktı beklentileri gibi çeşitli ve farklı detay seviyelerinde soruların cevapları yer almaktadır. Bu cevapların geliştiricilere sağlanabilmesi için analiz aşamasında kullanıcılar ile analiz aşamasına özel ve proje planlaması aşamasında belirlenmiş teknikler ile gerçekleştirilecek işin analizi yapılır. Analiz aşaması çıktısı olarak, analiz aşamasından sonraki aşama olan tasarım aşamasında kullanılmak üzere (tasarım aşamasına girdi teşkil edecek) bazı çıktılar üretilir. Bu çıktılar, geliştirilecek sistemin/yazılım ürünün gereksinimleri (sistem/alt sistem gereksinim tanımları-system/sub system spesification, yazılım gereksinim tanımları-software

requirement spesification), sistem/yazılım gereksinimlerinin birbirleri ile ilişkisi, sistem/yazılım gereksinimlerinin kullanıcı/müşterinin istekleri ile ilişkisi, sistem/yazılım gereksinimlerinde yer alan kullanıcı rolleri (user role), kullanım durumları (use case) ve kullanım durumlarının aktivite/eylem tanımlarıdır (activity).

2.2 Tasarım Aşaması

Yazılım geliştirme yaşam döngüsü içerisinde, analiz aşamasının ardından tasarım aşaması gelmektedir. Analiz aşamasının sonunda üretilmiş olan çıktılar, tasarım aşamasına girdi olmaktadır. Tasarım aşaması içerisinde, geliştirilecek olan yazılım ürününün farklı başlıklar altındaki tasarım kararları yer almaktadır. Tasarım aşamasında, yazılım ürünü için veri tabanı tasarımı (veri tabanı varlık-ilişki / entity-relationship / E-R modeli, kavramsal tasarım, mantıksal tasarım, fiziksel tasarım), yazılım tasarım tanımları (kullanılacak tasarım deseni, sınıf şemaları, kritik tasarım ögeleri, modül/bileşen ilişkileri), sistem mimari tasarımı (geliştirme, test, ürün gibi kurulum ortamlarının tasarımı, kurulum ortamları arasındaki geçişkenlik ilişkileri, sistem/alt sistem entegrasyonları için iletişim yöntemleri tasarımı) ve arayüz tasarımı (sistem arayüz tasarımı ve kullanıcı arayüz tasarımı) üretilmektedir. Geliştirilecek olan yazılım ürünü farklı mimari alt yapılara ve tasarım modellerine sahip olabilir. Geliştirme yöntemleri arasından uygulanacak olan mimari yapı ve tasarım modeli belirlenerek bu çıktılar üretilir. Tasarım aşamasında üretilen çıktılar geliştirme aşamasına girdi teşkil etmektedir. Geliştirme aşamasının her adımında, önceden belirlenen tasarıma uygun olarak geliştirme (yazılım kodlaması) yapılır.

2.3 Geliştirme Aşaması

Yazılım geliştirme yaşam döngüsü içerisinde önceki aşamalar vasıtası ile edinilmiş bilgiler ışığında geliştirilecek olan yazılım ürününün kodlaması yapılmaktadır. Tasarım aşaması sonunda üretilmiş olan çıktılar, geliştirme aşamasına girdi olmaktadır. Tasarım aşaması çıktılarına göre yazılım geliştirmesi yapılmakta ve her aşamada tasarım çıktıları ile uygunluk kontrolleri her bir yazılım parçacığı için (birim testler ile) gerçekleştirilmektedir. Geliştiriciler, her bir yazılım parçacığını/bileşenini farklı yöntem ve teknik kullanarak geliştirebilecekleri gibi aynı yöntem ve teknikleri kullanarak da belirli bir standart prosedür ile de bütüncül bir geliştirme yolunu izleyebilirler. Her iki yöntem için de geliştirme aşamasında belirli bir standartlaşma mekanizmasının işletilmesi ve geliştirme işlemlerinde

rol alacak kişilerin bu standartlara uyması gerekmektedir. Bunun sağlanabilmesi için geliştirme aşamasının başında veya uygulanan proje geliştirme metodolojisine göre planlama aşamasında belirlenen uygun aşamada kod standartlarının belirlenmesi önemlidir. Geliştirme aşaması çıktısı olan yazılım ürünü/yazılım parçacığı için test aşamasına geçilir.

2.4 Test Aşaması

Test aşaması ile; geliştirilmiş olan yazılım ürünü/yazılım parçacığı yine planlama aşamasında detaylandırılmış şekilde test işlemlerine tabi tutulur. Test aşaması içerisinde hem geliştirilmiş olan ürünün işlevselliği, yani kullanıcının analiz aşamasında belirtmiş olduğu ihtiyaçların karşılanıp karşılanmadığı ve arayüz isteklerine uygunluğu hem de geliştirilmiş olan ürünün güvenlik, yük, stres ve performans testleri gerçekleştirilmektedir. Test aşamasında uygulanacak farklı test çeşitleri için farklı test yöntemleri kullanılabilir. Test metodolojileri, ayrıca incelenmesi ve detaylandırılması gerekli olan ve kendince uzmanlık gerektiren bir alanda yer almaktadır. Proje içerisinde test tekniklerinin detayı belirtilmemekte, sadece proje geliştirme süreçleri içerisindeki test aşamasının yeri irdelenmektedir.

3. YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ METODOLOJİLERİ

Yazılım geliştirme projelerinin yönetiminde farklı metodolojiler uygulanmaktadır. Yazılım geliştirme projelerinde uygulanan her bir metodoloji, aslında yukarıdaki bölümde yer verilen yazılım geliştirme yaşam döngüsü içerisinde yer alan aşamalardan oluşmaktadır. Fakat her bir yazılım geliştirme projesi metodolojisinde, SDLC aşamalarının uygulanma şekli arasında farklar bulunmaktadır.

3.1. Şelale (Waterfall) Metodolojisi

Şelale (farklı kaynaklarda Çağlayan olarak da anılmaktadır), ingilizce deyimi ile "Waterfall", metodolojisi; SDLC içerisinde yer alan her bir aşamanın, yazılım geliştirme projesinde geliştirilecek yazılım ürününün tamamı için ve projenin takvimi içerisinde her aşamanın bir kez ve ardı ardına uygulandığı yöntemdir. SDLC aşamalarının birbirinin peşi sıra uygulanması ile yazılım ürünün tamamı tek seferde proje sonunda teslimata/kuruluma hazır hale gelmektedir.

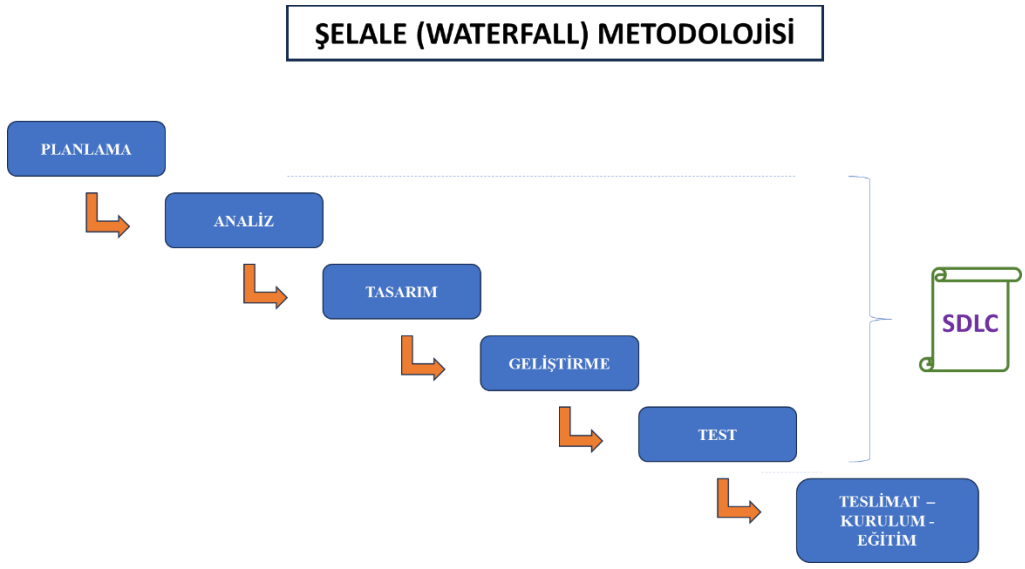
Şelale metodolojisinin uygulanması için proje yönetimi kurgulanırken, projenin kullanıcısı/müşterisi ile gerçekleştirilecek analiz aşamasında, yazılım ürünün bütününün gerçekleştirmesi beklenen işlevler görüşülür, ihtiyaçlar belirlenir ve analiz aşaması çıktıları tüm ihtiyaçları karşılayacak nitelikte gereksinimleri içermektedir. Tasarım aşamasında ise yine tüm gereksinimlere yönelik tasarım kararları ve çıktıları tek seferde yazılım ürününün bütünü için oluşturulmakta ve geliştirime aşamasına geçilmektedir. Yine geliştirme aşaması da tüm yazılım ürününün uçtan uca geliştirilmesinin tamamlanması sonrasında kapanmaktadır. Geliştirilmesi tamamlanmış yazılım ürününün tamamı, belirlenen test tekniklerine göre test edilir ve gerekli durumlarda test sonucunda hata düzeltme işlemleri yapılır. Tüm aşamalar tamamlandıktan sonra ise yazılım ürünü teslimata/kuruluma hazır hale gelmektedir.

Yukarıda işleyişi anlatılan şelale metodolojisinin uygulanabilmesi için, analiz aşamasıyla birlikte yazılım ürünün tamamı için yazılım/sistem gereksinimleri belirlenmiş ve net olmalıdır. SDLC aşamalarının herhangi bir adımında bu gereksinimler arasında değişiklik, ekleme ve/veya çıkarma işlemi uygulanmak istendiğinde, geriye dönük olarak yapılacak her

işlem, proje geliştirme maliyetinde büyük değişikliklere yol açmaktadır. Bu sebeple geriye dönük işlemler şelale metodolojisi için "mümkün olamaz" denilemese bile çok maliyetli olacağından, "zor" olarak tanımlanır ve proje paydaşları tarafından değişikliklere sıcak bakılmamaktadır.

Bu sebeple Şelale metodolojisi, istekleri belirlenmiş ve değişiklik payı olmayan ve/veya çok az sayıda değişiklik olması tahminlenen projeler için uygulanması doğru olan bir yöntemdir. Bu tarz projelere örnek olarak, devlet kurumları (bakanlıklar ve savunma sanayii vb.), özel sektörde iş yapan kurum/kuruluşlar arasında iş süreçleri ve ihtiyaçları oturmuş ve değişiklik oranı çok düşük olan kullanıcı/müşteriler için geliştirilecek olan yazılım projeleri sayılabilir.

Şelale metodolojisinde, proje planı içinde kaynak yönetimi gerçekleştirilirken, SDLC aşamalarına ve proje içerisinde yer alacak faaliyetlere özel yetkinliklere sahip ve genellikle ayrışmış ekipler tercih edilmektedir. Her ekip üyesinin proje çalışmalarındaki her işi üstlenmeleri, peşi sıra iş akışına sahip olan şelale metodolojisinde geriye dönük işlemleri çoğaltma riskine neden olabileceği için, farklı alanlardaki işlere özel yetkinliklerdeki ekip üyeslerinin kullanımı önem arz etmektedir. Yazılım projesinin büyüklüğüne göre proje organizasyonunun yönetim şeklinde farklı prensipler izlenebileceği gibi küçük-orta-büyük olarak her ölçekteki yazılım geliştirme projelerinde, kullanıcı/müşterinin geliştirilecek yazılım ürününe katkısı, takvimsel kısıtlar ve iş niteliğinin belirginliği konuları göz önünde bulundurularak şelale metodolojisinin yazılım geliştirme projelerinde seçimi değerlendirilmelidir.



Şekil 3.1 Şelale Metodolojisi

3.2.V-Model Metodolojisi

V-Model metodolojisi, şelale metodolojisinin gelişmiş versiyonu olarak nitelendirilmektedir.

V-Model metodolojisi uygulanırken şelale modelindeki gibi SDLC süreçleri (aşamaları) yazılım ürününün tamamı için işletilmektedir. V-Model uygulaması her bir SDLC aşamasına karşılık gelen test (doğrulama ve geçerleme) aşamasının uygulanması ile tamamlanmaktadır. Bu metodolojinin uygulanması 2 bacak ve 3 aşamada modellenmiştir.

V-Model bacakları;

- Doğrulama (Validation)
- Geçerleme/Onaylama (Verification)

V-Model aşamaları;

- Kullanıcı Modeli
- Mimari Model
- Uygulama/Geliştirme Modeli

Her model, V-Model metodolojisinde bir aşama olarak değerlendirilebilir.

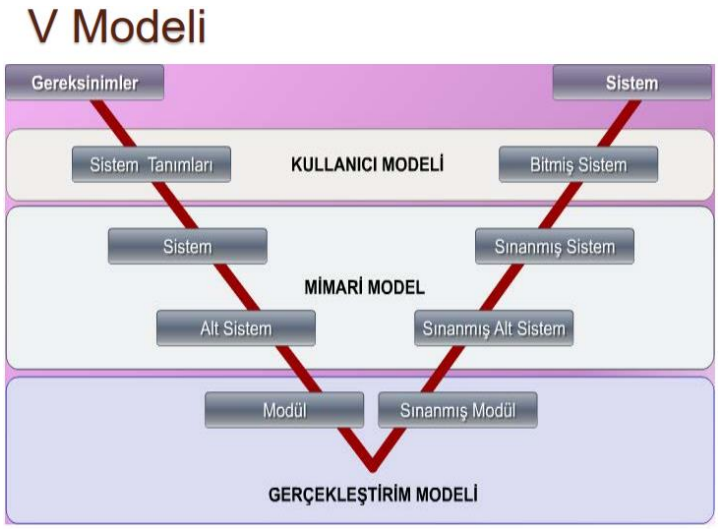
SDLC aşamalarından analiz aşaması, "kullanıcı modeli" içerisinde "yazılım gereksinimleri" ve "sistem tanımları" olarak belirtilir ve analiz aşamasında oluşturulan gereksinimler için sistem tanımları meydana getirilir. Yazılım gereksinimleri ve sistem tanımları yine kullanıcı modeli içerisinde, test aşamasında "Sistem Testi (System Test)" ve "Kabul Testi (Acceptance Test)" ile geçerlenir/onaylanır.

SDLC aşamalarından tasarım aşaması, "mimari model" içerisinde "sistem/alt sistem tasarımı (detay tasarım, kritik tasarım, mimari tasarım, yüksek//düşük seviye tasarım)" şeklinde ele alınır. Tasarım aşamasında gerçekleştirilen işler, mimari model içerisinde test aşamasında "entegrasyon testi" ve "birim test (unit test)" geçerlenir/onaylanır.

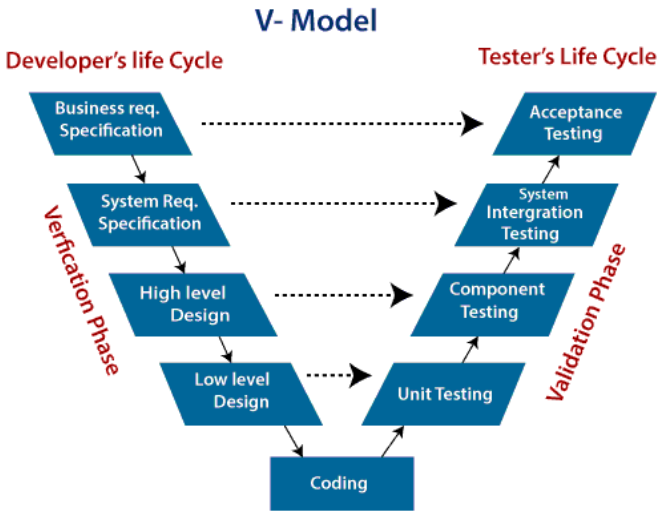
SDLC aşamalarında geliştirme aşaması, "uygulama/geliştirme modeli" içerisinde "kodlama/geliştirme" olarak ele alınır. Bu aşamada test işlemleri ayrıca uygulanmamakta, kullanıcı modeli ve mimari model içerisinde test işlemleri tüm yazılım ürünü için gerçekleştirilmektedir.

V-Model metodolojisi aynı şelale metodolojisi gibi iş kuralları ve ihtiyaçların önceden belirlendiği ve net olduğu durumlardaki yazılım geliştirme projeleri için uygulanabilirdir. Yine savunma sanayii projeleri bu metodolojinin kullanımı için örnek verilebilir. Doğrulama ve geçerleme yöntemlerinin iç içe ve birbirleri ile ilişkisinin yoğun şekilde işletildiği savunma sanayii projeleri V-Model metodolojisinin en yaygın kullanıldığı alandır.

V-Model metodolojisinin kullanımı, yine şelale metodolojisinde olduğu gibi kullanıcı/müşterinin proje geliştirme aşamasına katkısı, iş belirginliği ve takvimsel kısıtlar göz önünde bulundurularak karar verilmesi gereken bir konudur. Şelale metodolojisinde olduğu gibi proje geliştirme sürecinde yer alacak ekip üyelerinin alan bazlı ayrıştırılması yaygın olarak kullanılan bir yöntemdir.



Şekil 3.2 V-Model Metodolojisi-1



Şekil 3.3 V-Model Metodolojisi-2

3.3. Helezonik (Spiral) Metodoloji

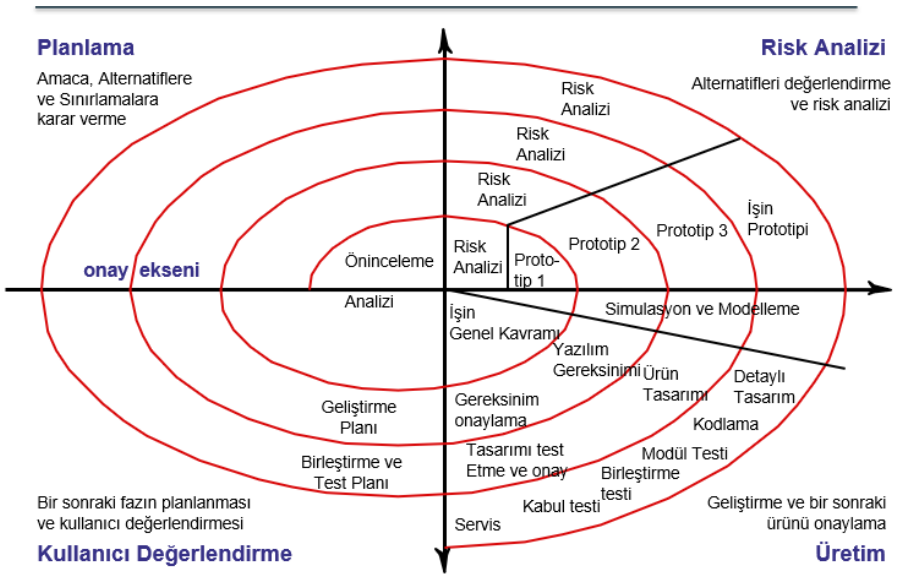
Helezonik (spiral) metodoloji, şelale veya V-Model metodolojilerinden farklı olarak, SDLC aşamalarının, sarmal halde, proje boyunca birbirini takip eden, birden fazla kez döngü halinde ilerlemeli (iterative) şekilde uygulandığı metodolojidir.

Spiral metodoloji ile yazılım geliştirme projelerinde birden fazla kez yazılım parçacığı/ürünü için sürüm oluşturulur. Her bir tasarım ögesi için bir spiral döngüsünde SDLC süreçlerinin tamamı işletilir ve sonraki spiral döngüsü bir öncekinin üzerine eklenerek ilerleme sağlanır. Her döngü içinde geliştirilen yazılım parçacıkları, projenin sonuna gelindiğinde bütünleşik ürünün tamamını oluşturmuş olur.

Spiral metodoloji ile yüksek riske sahip, yani kullanıcı/müşteri isteklerinde değişikliklerin yoğun yaşanabileceği büyük ve kullanıcı tarafından prototip yazılım ürünü beklentisi olan projelerin geliştirilmesi uygun olmaktadır.

Spiral metodolojinin uygulandığı yazılım geliştirme projelerinde, değişiklik yönetimi, iyi şekilde kurgulanmalı ve projenin tüm paydaşları tarafından başarılı şekilde uygulanmalıdır. Değişiklik yönetimi, yazılım ürünlerinin geliştirildiği her sektör için uygulanması zor bir alan olduğundan, Spiral metodolojinin uygulanması da ancak projenin başlaması, yani planlama aşamasında belirlenmiş iş/proje yönetim ve değişiklik yönetim planlarına uyumluluk ile mümkün olabilmektedir. Spiral döngüler arasında yaşanması muhtemel olan her değişiklik için belgelendirme ve değişiklik takibi çok önemli bir rol oynamaktadır.

Spiral metodoloji uygulanan bir yazılım geliştirme projesinin, iş ihtiyaçlarında değişiklikler olabilmesi gibi bir esneklik bulunsa bile, bu değişikliklerin ölçümlenmesi projede kaynak kullanımı ve takvimsel kısıtların yönetilmesi konularında yüksek bir esneklik olduğu anlamına gelmemektedir. Bu iki faktör üzerindeki esneklik, projenin maliyetini doğrudan etkileyen unsurlardan biri olduğu için yine şelale metodolojisindeki kaynak ve takvim yönetimi kısıtlamaları genellikle spiral metodolojide yer almaktadır. Bu sebeple ilerlemeli metodolojilerin atası olarak kabul edilen spiral metodoloji, güncel olarak kullanılan tam esnek iterasyonlara sahip yönetim metodolojiler kadar gelişmiş görülmemektedir.



Şekil 3.4 Helezonik (Spiral) Metodoloji

3.4. Çevik (Agile-Scrum) Metodoloji

Çevik metodoloji, spiral metodolojinin gelişmiş bir versiyonudur. Spiral metodolojinin içerisinde yer alan her bir döngüye (iterasyon) karşılık çevik metodolojide koşum (sprint) yer almaktadır.

Çevik metodoloji içerisinde farklı alt kollar bulunmaktadır. Bu araştırma projesi içerisinde Scrum metodolojisi irdelenmiştir.

- Srcum Metodolojisi
- LeSS Metodolojisi (Large-Scale Scrum, Geniş Ölçekli Scrum Metodolojisi)
- SAFe Metodolojisi (Scaled Agile Framework, Ölçeklenen Çevik Çerçeve Metodolojisi)
- KANBAN Metodolojisi

Scrum metodolojisi ile yazılım ürünü parçalar halinde geliştirilmektedir. Her yazılım parçası scrum koşumları başlamadan önce mümkün olduğu kadar tahminlenir. Belirlenen her yazılım parçacığı için bir önceliklendirme yapılır. Önceliklendirilmiş yazılım parçacıklarının listesine "Product Backlog" ismi verilir ve scrum koşumlarına bu liste içerisinden elemanlar seçilerek dahil edilir. Scrum koşumlarına dahil edilen elemanların listesine ise "Sprint Backlog" ismi verilir. Scrum koşumları içerisinde yazılım parçacıkları belirlenen önceliklendirme sırası ile geliştirilir. Geliştirme işlemi için SDLC aşamaları uygulanır. Scrum metodolojisi içerisinde SDLC aşamaları düzenli olarak takip edilemeyebilir. Scrum için önemli olan, her koşum içinde, o koşuma dahil edilmiş geliştirme

ögesinin tamamlanmasıdır.

Scrum metodolojisini spiral metodolojiden ayıran nokta, her spiral döngüsünde bir sonraki geliştirme ögesi bir öncekinin üzerine eklenerek geliştirme yapılırken, scrum metodolojisi içinde bir koşuma dahil edilen geliştirme ögesi bir önceki koşumun devamı olmak zorunda değildir.

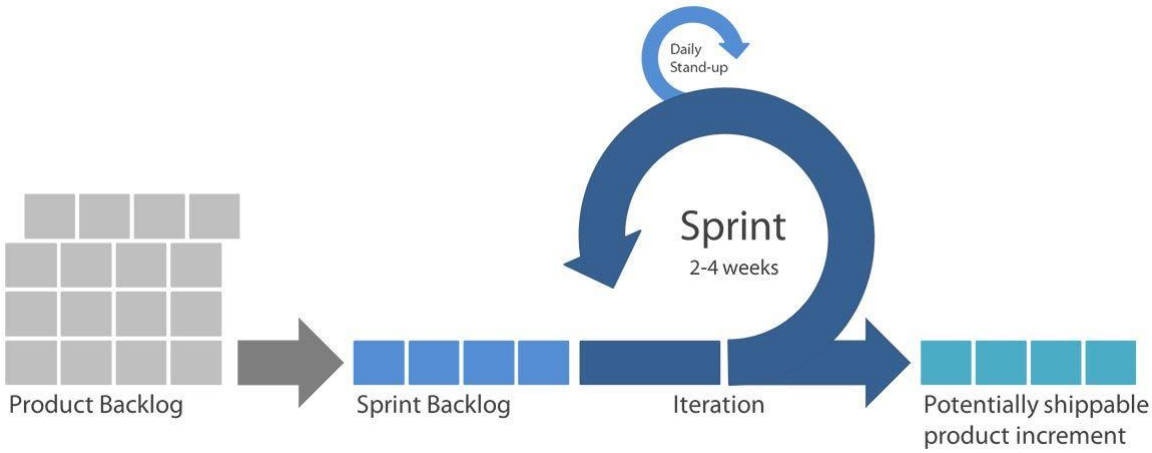
Scrum metodolojisinin bir çevik yöntem olmasının en büyük sebebi, koşum planına dahil edilen geliştirme ögeleri, koşum esnasında dahi değişikliğe uğrayabilir. Buna ek olarak daha önceki bir koşum içerisinde tamamlanmış bir geliştirme ögesi/parçası için de proje takvimi içerisindeki herhangi bir koşum esnasında değişiklik talebi uygulanabilmektedir. Bu durum, çevik metodolojinin, kullanıcı/müşterinin isteklerine proje süresince çok hızlı yanıt verebiliyor olması esnekliğini getirirken, projenin planlama aşamasındaki tahminleme işleminin zorlaşmasına yol açmaktadır. Bu sebeple scrum metodolojisi, yazılım geliştirme projelerinde, projenin ilk başlama aşamasında, harekete geçilecek noktanın, yani geliştirilmesi ilk öncelikli bir veya birkaç yazılım parçacığının bilinmesinin yeterli olması gibi bir kolaylık sağlamaktadır.

Scrum metodolojisi ile bir koşum içerisindeki geliştirme ögesinin geliştirilmesi sırasında karşılaşılan sorun/hatalara yönelik hemen bir sonraki koşum içerisinde gerekli işler yapılabilmektedir. Hatta koşum planlamaları her koşum öncesinde bu durumlar göz önünde bulundurularak gerçekleştirilir.

Geliştirilecek yazılım ürününün tamamı için gereksinimlerin net olarak tanımlanmış olmasının gerekmemesi, tüm geliştirme ögelerinin belirli olduğu durumda çevik yöntemler ile geliştirme işleminin yapılamayacağı anlamına gelmemektedir. İş planlamasının tamamı önden yapılabilen yazılım geliştirme projeleri için de scrum metodolojisi, scrum ekibinin ve scrum çalışma planının da uygun yapılmış olması koşulu ile uygulanması doğru olabilecek bir yöntemdir.

Scrum metodolojisinin uygulanması için, yazılım geliştirme projesinin yürütüldüğü organizasyon ve bu organizasyonun her kademedeki üyesinin, kullanıcı/müşterinin projeye olan katkısının, scrum ekibinde yer alacak geliştirme üyelerinin yani aslında tüm paydaşların uygun yapıya ve hazırlığa sahip olması gerekmektedir. Scrum ekibi içinde her üye eşit olarak

geliştirme ekibinin bir üyesi olduğundan, geleneksel geliştirme metodolojilerindeki ekip üyesi rollerinden scrum metodolojisinde bahsedilmemektedir. Scrum koşumlarında geliştirilecek yazılım parçacıkları, ekip üyelerinin bir koşum içerisinde üstlenebileceği yük kadar olacağından, scrum ekibi üyelerinin scrum metodolojisinin kullanıldığı geliştirme projesi dışında farklı bir görevde bulunması uygun olmamaktadır. Bu sebeple yazılım geliştirme projelerinde, hem yukarıda belirtilen faktörler göz önüne alınarak hem de scrum ekibinin yetkinliği bir bütün olarak değerlendirilerek scrum metodolojisinin uygulanması konusuna karar verilmesi gerekmektedir.



Şekil 3.5 Scrum Metodolojisi

3.5. Uygulama/Araç Kullanımı

Yazılım geliştirme projelerinde SDLC aşamalarının takibinde bazı araçların/uygulamaların kullanımı, proje geliştirme süreci içerisinde iş ve görev takibinin yapılabilmesini kolaylaştırmaktadır.

SDLC aşamalarının yazılım geliştirme projelerinde uygulanması, sadece her aşamada yapılacak işin tanımlanması için değil aynı zamanda kullanıcı/müşteri isteği ve ihtiyacının, proje geliştirmesi tamamlandığında, karşılanıp karşılanmadığının takibinin yapılması anlamına da gelmektedir. Bu takibin yapılabilmesi için iş ihtiyacı tanımından başlayarak sırası ile sistem/yazılım gereksinimleri, tasarım kararları, geliştirme ve test/doğrulama adımlarının birbirleri ile ilişkilerinin takip edilmesi gerekmektedir. Her adımın takibi ile kullanıcı/müşteri tarafından tariflenen iş ihtiyacının teslimat ile verilen yazılım ürünü

üzerinden karşılanıp karşılanmadığı izlenebilir. Bu izleme için "gereksinim izlenebilirlik matrisi" oluşturulur.

Bu şekilde gereksinim izlenebilirliğinin sağlanabilmesi için yazılım geliştirme sektöründe pek çok ücretli veya ücretsiz uygulama/araç bulunmaktadır. Bu araçlara örnek olarak Attlassian ailesinde yer alan Jira uygulaması (ücretli) veya açık kaynak bir uygulama olan Redmine uygulaması (ücretsiz) gösterilebilir.

Bu tip araçlar, görev takibinin yapılmasına imkân sağladığı için her bir yazılım geliştirme projesinde, hangi metodolojinin kullanıldığına bağlı olmadan kullanılabilmektedir. Fakat çevik metodolojiler için bu tarz araçların kullanımı çok daha fazla önem arz etmektedir. Çünkü, çevik metodolojiler, daha az doküman tabanlı çıktının oluşturulması yönünde iş planına sahiptirler. Çevik metodolojilerde çıktılar, doküman ağırlığından çok, yazılım parçacıklarının tamamlanmasına öncelik vermektedir. Doküman ağırlığı olmadığı için, geliştirilen yazılım ürünü ile ilgili yapılan işlemler ve geliştirme sırasında alınan kararların, bilgi birikimi çerçevesinde kayıtlı olmasının en güzel ve kolay yolu bu tarz görev yönetim araçlarını kullanmaktır.

4. SONUÇ VE ÖNERİLER

Proje raporumuz içerisinde incelemiş olduğumuz, Şelale, V-Model, Spiral ve Çevik-Scrum metodolojilerinin tamamında SDLC sürecinin, yani analiz (ihtiyaç belirleme, gereksinim toplama, kullanıcı ihtiyaçlarına göre yazılım ürününün gereksinimlerinin belirlenmesi), tasarım (yazılım ürününün her bir bileşeninin tasarlanması), geliştirme (tasarlanmış ürün bileşenlerinin kodlanması, entegrasyon ihtiyaçlarının geliştirilmesi) ve test (geliştirilmesi sağlanan her bir bileşen/yazılım parçacığı/sistem parçacığının ihtiyaca uygunluğu takip edilerek doğrulamasının yapılması) aşamalarının işletimi sağlanmaktadır.

Metodolojiler arasındaki farklar, projelerin yönetimi ve geliştirmesi aşamalarındaki kaynak, kapsam, maliyet, iş ihtiyacı olgunluğu, kullanıcı/müşteri ve geliştirici arasındaki iletişim ve uyum gibi kriterler üzerindeki çalışma yöntemlerinde yaşanan kısıt ve esnekliklere bağlı olarak oluşmuştur. Sayılan her bir kriter için çeşitli analiz ve risk belirleme çalışmaları yapılarak yazılım geliştirme projelerinin yönetim şekilleri belirlenmelidir.

İhtiyaçları önceden belirlenmiş ve netleştirilmiş olan, düzenli bir süreç akışına sahip, bu süreç ve iş akışları üzerinde orta ve uzun vadeli periyotlardaki değişiklik oranının düşük olduğu, SDLC süreci boyunca ihtiyaç sahibi ile geliştirici tarafın ortak çalışmasının ve geliştirme takviminin kısıtlı olduğu kurum/kuruluşlar için geliştirilmesi istenen yazılım ürünleri için kurgulanacak olan yazılım geliştirme projelerinde izlenecek metodolojinin Şelale veya V-Model metodolojisi olması daha uygun olacaktır.

Öte yandan, ihtiyaçların tamamının önceden net bir şekilde belirlenemediği, belirlenebilse bile ihtiyaçlar üzerinde değişiklik olması ihtimali yoğun olarak ön görülen, ihtiyaçlarda değişiklik ön görülmese de hem ihtiyaç sahibi ve geliştirici arasındaki iletişim ve uyumlu çalışma düzeninin SDLC sürecinin her aşamasında esnek şekilde uygulanabileceği hem de geliştirilmesi beklenen yazılım ürününün belirli periyotlarda gözlemlenecek ve/veya kullanılacak şekilde çıktı üretmesinin beklenmesi, geliştirici ekip tarafından ihtiyacın karşılanması sırasında yaşanan her türlü olaya karşı adaptasyonun yüksek seviyede olması beklendiği durumlarda iterasyonlu metodolojiler benimsenmelidir. İterasyonlu metodolojiler arasında çevik-scrum metodolojisinin kullanımı günümüzde daha yaygın bir

hal almıştır. Spiral metodolojinin kullanımı ise tek başına güncelliğini yitirmiş ve geleneksel yöntemler arasında proje süresince iş takibinin yapılabilmesi için daha fazla maliyet gerektiğinden yaygın kullanım için çok fazla tercih edilen metodolojiler arasında yer almamaktadır.

Bahsi geçen metodolojilerin tek başına kullanımları detaylı şekilde açıklanmıştır. Fakat, günümüzde yazılım geliştirme projelerinde, proje ve geliştirme süreçleri kurgulanırken metodolojilerin karma (mixed) kullanımları da mümkün olabilmektedir. Örneğin, proje kurgusu içerisinde SDLC süreci şelale metodolojisi kullanılarak yazılım ürünü geliştirmesi yapılabilirken, SDLC sürecinin geliştirme aşamasında spiral metodoloji kullanımı uygulanabilmektedir. Bu şekilde kurgulanan projelerde, şelale metodolojisi ile iş takibi yapılabilmekte, değişiklik yönetimi uygulanabilmekte ve geliştirme aşaması iterasyonlar halinde yapılarak kullanıcı/müşteri tarafından ortaya çıkan yazılım parçacıkları gözlemlenebilmektedir. Burada bahsi geçen karma metodoloji kullanımı gibi V-Model ile yine spiral metodolojinin karma olarak kullanılabilmesi de mümkün olmaktadır.

Sonuç olarak, yazılım geliştirme projelerinde SDLC sürecinin işletileceği farklı metodolojilerin, projeler kurgulanırken hem projenin ihtiyaçlarının olgunluğu hem geliştirici ve kullanıcı//müşteri tarafının proje içerisindeki katılım durumu ve uyumu hem de projenin maliyet kalemlerinin direkt veya dolaylı olarak etkilendiği proje yönetim bacaklarındaki (zaman, kapsam, maliyet, kalite ve risk yönetimi) kısıtlar göz önünde bulundurulmalıdır.

KAYNAKLAR

1. Gencer, C., Kayacan, A. (2017). *Yazılım Proje Yönetimi: Şelale Modeli ve Çevik Yöntemlerin Karşılaştırılması,* Bilişm Teknolojileri Dergisi (Cilt:10, Sayı:3), 349-350

2. Çalışkan, D., Yavuz, A. F., Doğan, B., Çalış Uslu, B. (2021). *Türkiye'de Çevik ve Klasik Yazılım Geliştirme Metodolojilerine Dair Kapsamlı Bir Değerlendirme,* BEÜ Fen Bilimleri Dergisi (10-1), 150-151

3. İnternet: Çetin, Ö. H., Yazılım Yaşam Döngü Modelleri URL: https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742 , Son Erişim Tarihi: 05.06.2023.
4. İnternet: Yazılım Mühendisliği (Bilecik Üniversitesi), Ders-2: Yazılım Geliştirme, URL: http://w3.bilecik.edu.tr/bilgisayar/wp-content/uploads/sites/75/2019/02/ders-2.pdf, syf:17, Son Erişim Tarihi: 05.06.2023.