Yazılım Proje Yönetimi: Şelale Modeli ve Çevik Yöntemlerin Karşılaştırılması

Cevriye GENCER[1], Ali KAYACAN[2]

[1]Endüstri Mühendisliği Bölümü, Gazi Üniversitesi Mühendislik Fakültesi, Ankara, Türkiye
[2]Yönetim Bilişim Sistemleri Anabilim Dalı, Gazi Üniversitesi Bilişim Enstitüsü, Ankara, Türkiye
ctemel@gazi.edu.tr, ali.kayacan@gazi.edu.tr
(Geliş/Received:04.05.2017; Kabul/Accepted:21.07.2017)
DOI: 10.17671/gazibtd.331054

Özet— Günümüzde yazılım, günlük eğlencemizden en karmaşık iş süreçlerine kadar hayatımızın her alanında yerini almış ve etkileri hissedilen bir olgudur. Şirketler, iş süreçlerinin yönetimi için etkin yazılım sistemlerine ihtiyaç duyarlar ve bu sistemleri bilgi sistem departmanları vasıtasıyla üretir ya da yazılım şirketlerinden tedarik ederler. Her iki durumda da yazılımın üretilmesi bir dizi mühendislik ve proje yönetim sürecini gerektirir. Bu süreçlerin metodoloji seçiminde, üretilecek yazılımın büyüklüğü, karmaşıklığı ve kullanım maksadı gibi karakteristik özellikleri, yazılımı üretecek organizasyon ve proje ekibinin yapısı ve kullanılabilecek kaynaklar önem arz eder. Bu çalışmada, şelale modeli ile çevik yöntemler arasında, proje yönetimi ve kontrolü, müşteriler, organizasyon yapısı, proje elemanları, ürün, gereksinimler, kaynaklar ve risk yönetimi açısından ortaya çıkan durumlar temel alınarak bir karşılaştırma ortaya konulmakta ve yazılım proje yönetimi metodolojisi seçimi konusunda önerilerde bulunulmaktadır.

Anahtar Kelimeler— Şelale Modeli, Çevik Yöntemler, Çevik Metodolojiler, Yazılım Proje Yönetimi

Software Project Management: A Comparison of Waterfall Model and Agile Methodologies

Abstract— In our time, software is a fact, the influence of which is strongly felt and has attained its place in our lives from daily entertainment to the most complex business processes. Corporations need efficient software systems in order to manage their business processes and they either produce these systems through the agency of their information systems departments or procure from software companies. In both situations, the production of software requires a series of engineering and project management processes. In selection of methodology for these processes, the characteristics of software such as size, complexity and purpose of use, the structure of the project team and organization which are to produce the software and available sources are important. In this study, a comparison of waterfall model and agile methodologies based on situations about Project Management and Control, Customers, Organizational Structure, Project Members, Product Requirements, Sources and Risk Management is presented and propsals about selecting a methodology for software project management are submitted.

Keywords— Waterfall Model, Agile Methodologies, Software Project Management

1. GİRİŞ (INTRODUCTION)

Günümüzde bulut bilişim, nesnelerin interneti, büyük veri, sosyal ağlar ve dağıtık sistemler gibi gelişerek yaygınlaşan kavramlar ve donanımsal teknolojilerdeki gelişmeler sayesinde gerek gömülü sistemler vasıtasıyla evimizden savunma sistemlerine, gerekse cep telefonu ve bilgisayarlar vasıtasıyla günlük eğlencemizden en karmaşık iş süreçlerine kadar, yazılım, hayatımızın her alanında yerini almış ve etkileri hissedilen bir olgudur. Şirketler, iş süreçlerinin yönetimi için etkin yazılım sistemlerine ihtiyaç duyarlar ve bu yazılımı bilgi sistem departmanları vasıtasıyla üretir ya da yazılım şirketlerinden tedarik ederler. Her iki durumda da yazılımın üretilmesi bir dizi mühendislik ve proje yönetim sürecini gerektirir. Bu süreçlerin metodoloji seçiminde, üretilecek yazılımın büyüklüğü, karmaşıklığı, kullanım maksadı gibi karakteristik özellikleri, yazılımı üretecek organizasyon ve proje ekibinin yapısı ve kullanılabilecek kaynaklar önem arz eder. Bu çalışmanın amacı, yazılım mühendisliği kavramının ortaya çıkmasından beri gelişerek kullanılmakta olan geleneksel plan tabanlı

yöntemlerin ve her türlü proje yönetimi metodolojisinin öncüsü konumundaki şelale modeli ile geleneksel plan tabanlı yöntemlerin yetersiz kaldığı durumlara bir alternatif olarak ortaya çıkan ve günümüzde çok önemli bir yer edinmiş olan çevik yöntemler arasında, projelere özgü durumlara uygun metodoloji seçimi konusunda rehberlik yapacak bir karşılaştırma sunmaktır.

Benzer konularda yapılan çalışmalar incelendiğinde genel olarak geleneksel plan tabanlı yöntemler ve çevik yöntemlerin içeriği ve uygulanması ile yöntemlerin belirli etkenler açısından ele alındığı çalışmalara rastlanmaktadır. Bu çalışmanın hazırlanmasında, elde edilen az miktarda karşılaştırma çalışmasının yanında, çevik yöntemlerin adaptasyonu, geleneksel plan tabanlı yöntemlerden çevik yöntemlere geçiş uygulamalarında elde edilen tecrübeler ve geleneksel plan tabanlı yöntemler ile çevik yöntemlerin birlikte kullanıldığı uygulamalarda elde edilen tecrübeler incelenmiştir. Ayrıca metodoloji kullanımı ve başarı oranları gibi istatistiki bilgi sunan araştırmaların mevcut en güncel verilerinden yararlanılarak genel resmin mümkün olduğunca sağlıklı bir şekilde ortaya konması amaçlanmıştır.

Çalışmanın ikinci ve üçüncü bölümünde sırasıyla şelale modeli ve çevik yöntemlerin kısa tarihçeleri, ilkeleri ve özellikleri ile fayda ve kısıtları ortaya konacak, dördüncü bölümde şelale modeli ve çevik yöntemlerin benzerlikleri ve farkları belirtilecek ve proje yönetimi ve kontrolü, müşteriler, organizasyon yapısı, proje elemanları, ürün, gereksinimler, kaynaklar ve risk yönetimi açısından ortaya çıkan durumlara göre karşılaştırılmaları yapılacaktır. Beşinci bölümde ise sonuçlar özetlenecek ve yazılım proje yönetimi metodolojisi seçimi konusunda önerilerde bulunulacaktır.

1.1. Yazılım ve Yazılım Mühendisliği (Software and Software Engineering)

Yazılım kelimesinin Türk Dil Kurumu Büyük Türkçe Sözlüğündeki karşılığı: "Bir bilgi işlem dizgesinin işleyişi ile ilgili bilgisayar izlencelerinin, yordamların, kuralların ve gerektiğinde belgelemenin tümü." olarak verilmiştir.[1] Yazılım sadece bir programlama dilinde yazılmış kod satırlarını değil, somut dokümantasyonu da içerir. Yazılım mühendisliği ise sistem tanımlamasının ilk safhalarından, yazılımın kullanım ve bakım safhalarına kadar yazılım üretiminin her aşamasında icra edilen faaliyetleri içeren mühendislik disiplinidir. Yazılım mühendisliği, yazılım geliştirme sürecinin sadece teknik bölümünü içermez, yazılım proje yönetimi ve yazılım üretiminin destekleyici unsurları olan metodoloji, kullanılan araçlar ve kavramların geliştirilmesi de yazılım mühendisliğinin kapsama alanına girer [2].

Yazılım mühendisliğinde, kullanılan metodoloji ne olursa olsun, dört temel faaliyetten meydana gelen sistematik bir yaklaşım uygulanır. Bu faaliyetler;

a.) Müşteriler ve geliştiriciler tarafından, üretilecek yazılımın gereksinim ve kısıtlarının belirlendiği *Sistem Tanımlaması.*

b.) Yazılımın tasarlandığı ve kodlamanın yapıldığı *Yazılım Geliştirme.*

c.) Yazılımın, müşterinin talep ettiği ürün olduğunun onaylandığı *Yazılım Doğrulama.*

ç.) Yazılımın, değişen müşteri ve pazar ihtiyaçları doğrultusunda güncellendiği *Yazılım Evrimi* [2].

Şelale modeli ile çevik yöntemlerin en temel farkı yazılım mühendisliğinin bu sistematik yaklaşımını uygulayış tarzıdır. Şelale modelinde bu faaliyetler birbirini takip eden adımlar şeklinde icra edilir ve bir adımın çıktısı bir sonraki adımın girdisini teşkil eder. Faaliyetlerin sona ermesi kesindir ve geriye dönüş yapılmaz. Proje sonunda elde edilen ürün müşteriye teslim edilir. Çevik yöntemlerde ise tekrarlı bir anlayış baz alınır. Bu adımların baştan sona uygulandığı süreçler ve bu süreçler sonunda elde edilen ürünün müşteriye teslimi sonucunda alınan geri besleme ile bir sonraki sürecin evrilerek nihai ürün ortaya çıkana kadar bu süreçlerin tekrarı ile proje sonucuna ulaşılması hedeflenir. Bu anlayış farkının avantaj ve dezavantajları mevcuttur ve uygulanması gereken şartlar ve ortamlar değişiklik göstermektedir. Bu çalışmanın asıl konusunu teşkil eden bu hususlar ikinci, üçüncü ve dördüncü bölümlerde ele alınacaktır.

1.2. Proje Yönetimi ve Yazılım Proje Yönetimi (Project Management vs. Software Project Management)

Proje, benzersiz bir ürün, hizmet ya da sonuç yaratmak için girişilen geçici bir çabadır[3]. Daha detaylı bir tanım yapmak gerekirse, bir probleme çözüm bulma ya da beliren bir fırsatı değerlendirmeye yönelik, bir ekibin, başlangıcı ve bitişi belirli bir süre ve sınırlı bir finansman dahilinde, birtakım kaynaklar kullanarak, müşteri memnuniyetini ve kaliteyi göz önünde bulundururken olası riskleri yönetmek şartıyla, tanımlanmış bir kapsama uygun amaç ve hedefler doğrultusunda özgün bir planı başlatma, yürütme, kontrol etme ve sonuca bağlama sürecidir [4]. Proje yönetimi ise, belirli bir projenin hedef ve amaçlarına ulaşıp bitirilmesi için kaynakların planlanması, organize edilmesi, tedarik edilmesi ve yönetilmesi disiplinidir [5]. IEEE tarafından hazırlanan Yazılım Mühendisliği Bilgi Tabanı (SWEBOK), yazılım mühendisliği yönetimini, "Yazılım ürünleri ve yazılım mühendisliği hizmetlerinin, etkin, verimli ve paydaşların fayda sağlayacağı şekilde ortaya konmasını sağlamak maksadıyla, planlama, koordinasyon, ölçme, izleme, kontrol ve raporlama gibi faaliyetlerin yönetimi" olarak tanımlamaktadır [6]. Proje Yönetimi Enstitüsü PMI ise Proje Yönetimi Bilgi Tabanı (PMBOK®) ve bu bilgi tabanının yazılım ekinde (SWX to PMBOK®) yazılım proje yönetimi faaliyetlerini, uygulanan metodolojiye bakılmaksızın yedi ana başlık altında toplamıştır. Bunlar, Başlatma ve Kapsam Tanımı, Yazılım Proje Planlaması,

Yazılım Proje Gerçeklemesi, Gözden Geçirme ve Değerlendirme, Kapanış, Yazılım Mühendisliği Ölçümlemesi ve Yazılım Mühendisliği Yönetim Araçlarıdır [3,7].

Yazılım projeleri özelleşmiş projelerdir ve proje yönetimi ile ilgili temel özellikleri taşırken, içeriği gereği özel yönetim teknikleri gerektirir. Yazılım projelerini diğer projelerden ayıran önemli farklar mevcuttur. Diğer mühendislik alanlarında meydana getirilen son ürünün fiziksel bir varlığı mevcutken, yazılım projelerinde ortaya çıkan ürün düşünsel süreçlerin bir tezahürü olduğu için soyuttur. Bu soyutluk hem proje yönetim süreçlerinin izlenebilirliğini sekteye uğratmakta, hem de müşterinin tam olarak ne istediğini anlatabilmesini zorlaştırdığı için proje başlangıcında tanımlanması gereken isterlerin doğru tanımlanmasını olumsuz etkilemekte ve projenin belirsizliğini arttırmaktadır. Genel proje yönetimine göre belirsizlikler daha fazla olduğu için yazılım projelerinde sarf edilecek iş gücünü ve maliyeti hesaplamak daha zordur ve başarısızlık ihtimali daha fazladır. Diğer mühendislik alanlarında icra edilen projeler zaman ve mekana göre çok farklılık göstermediği için proje yönetimi konusunda tecrübeler edinilmekte ve yeni projelerde bu tecrübelerden yararlanılmaktadır. Yazılım projelerinde ise bilgi teknolojilerinin hızlı gelişimi ve müşteri isterlerindeki değişimler, mevcut proje içerisinde dahi değişikliğe sebep olabileceği gibi önceki projelerde kazanılmış deneyimi neredeyse geçersiz kılmaktadır. Bu nedenlerden dolayı yazılım projelerinde başarı oranları düşüktür. Projelerin başarılı olup olmadığını belirleyen ölçütler, proje yönetimi üçgeni olarak da bilinen zaman - kaynak - kapsam öğeleridir. Standish Group International tarafından 1994 yılından beri her yıl yazılım projelerinin başarı oranları konusunda yayınlanan CHAOS raporlarına göre, 2011 - 2015 yılları arasında yazılım projelerinde elde edilen başarı oranları Tablo 1'de sunulmuştur.

Tablo 1. 2011-2015 CHAOS Raporları Başarı Oranları[8]
(CHAOS Reports Success Rates Between 2011-2015)

	2011	**2012**	**2013**	**2014**	**2015**
Başarılı Projeler	% 29	% 27	% 31	% 28	% 29
Sorunlu Projeler	% 49	% 56	% 50	% 55	% 52
Başarısız Projeler	% 22	% 17	% 19	% 17	% 19

CHAOS raporlarında belirtilen başarılı projeler, planlanan zaman içerisinde, planlanan bütçe aşılmadan hedefe ulaşılan ve tatmin edici bir sonuç yaratan projelerdir. Sorunlu projeler, tamamlanan fakat zaman ve bütçe konusunda planlananın dışına çıkmış, ya da gereksinimler ve fonksiyonlar açısından hedeflerine ulaşamamış projelerdir. Başarısız projeler ise iptal edilmiş ya da tamamlanamamış projelerdir. CHAOS raporlarında önemle üzerinde durulması gereken bir diğer konu ise proje başarısına etki eden faktörlerdir. 2015 yılı için belirlenen faktörler ve proje başarısına etki puanları Tablo 2'de sunulmuştur.

Tablo 2.CHAOS Başarı Faktörleri[8]
(CHAOS FActors of Success)

Başarı Faktörü	**Proje Başarısına Etkisi %**
Üst Yönetim Desteği	15
Proje Çalışanlarının Duygusal Olgunluk Seviyesi	15
Müşteri/Kullanıcı Katılımı	15
Proje Optimizasyonu	15
Proje Çalışanlarının Yetenek Seviyesi	10
Projelerde uygulanan Yöntemlerin Standartlığı	8
Çevik Süreçlere Hakimiyet	7
Sade Yönetim	6
Proje Yönetimi Uzmanlığı	5
Açık ve Anlaşılır Hedefler	4

Tablo 2'de belirtilen faktörlerin hemen hepsi proje yönetimi ve uygulanan metodoloji ile doğrudan ilgilidir. Yazılım projelerinde karşılaşılan düşük başarı yüzdeleri ve başarı faktörlerinin metodolojiyle olan bağıntıları ise uygulanan metodoloji konusuna verilmesi gereken önemi vurgulamaktadır.

2. ŞELALE MODELİ (WATERFALL MODEL)

Şelale modeli, yazılım projelerinde uygulanan faaliyetlerin ardışık safhalar halinde icra edildiği, yazılım mühendisliğinin en eski ve temel modelidir. Günümüzde hükümetler ve büyük şirketler tarafından her türlü projenin yönetim standardı olarak kabul görmektedir [9].

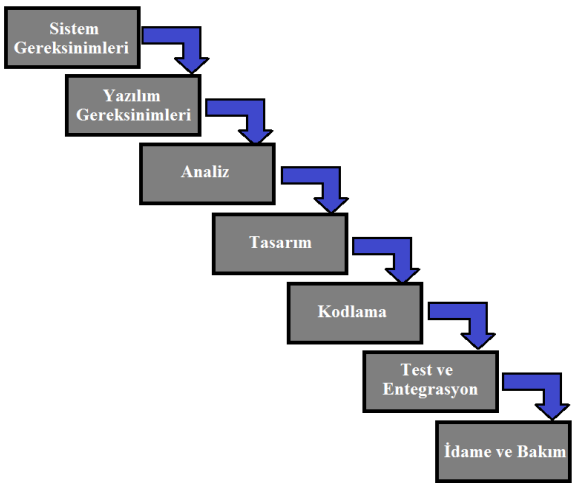
2.1. Tarihçe (History)

1950'li yıllarda uygulanmakta olan ilk yazılım geliştirme yöntemi "kodla ve düzelt" daha çok bilinen ismiyle "ad hoc ya da kovboy programlama", herhangi bir planlama veya tasarım safhası olmadan, doğrudan gereksinimlerin kodlanması ve süreç içi ya da ürün teslimi sonrası karşılaşılan hataların düzeltilmesi prensibine dayanmaktaydı. Zaman içinde yazılım projelerinin büyüyerek bir proje yönetim metodolojisi gerektirmesi, kritik sistemlerde hatalara tolerans gösterilemeyecek olması, teknolojinin gelişerek bilgisayar kapasitelerinin hızla artması ve kaliteli yazılım üretilememesi gibi nedenlerden dolayı 1960'lı yıllarda yaşanan yazılım krizi sonucu NATO Bilim Komitesi tarafından 1968 yılında Almanya'da Yazılım Mühendisliği Konferansı düzenlendi [10]. Bu konferansta üzerinde durulan temel konular yazılım mühendisliği kavramının tanımlanması ve yazılım

geliştirme süreçlerinde uygulanması gereken standartların tespit edilmesidir [10]. Yakın bir zaman sonra 1970 yılında Dr. Winston Royce tarafından yayınlanan bir bildiride Şelale Modeli tanımlandı [11]. Her ne kadar Dr.Royce tarafından önerilen metodoloji evrimsel ve yinelemeli süreçler ile geliştirme süresince müşterinin katılımı gibi yaklaşık 30 yıl sonra "çevik" olarak adlandırılacak uygulamalara vurgu yapsa da [12] Şelale yöntemi zaman içinde bugün bildiğimiz şeklini aldı ve yazılım geliştirme metodolojilerinin temeli haline geldi.

2.2. İlkeleri ve Özellikleri (Principles and Properties)

Şelale modelinde uygulanan safhalar Şekil 1'de gösterilmiştir.



Şekil 1. Şelale Modeli
(Waterfall Model)

Her safhanın sonunda genellikle dokümantasyondan oluşan bir çıktı elde edilir ve her çıktı kendini takip eden safhanın girdisini oluşturur [13]. Çıktı-girdi ilişkisi nedeniyle yaşanacak olası değişimlerin projeye maliyeti yüksek olacağı için safhalar arasında atlama veya büyük çapta geri dönüşlere izin verilmez. Şelale modelinin uygulandığı projelerde safha bitişleri net bir şekilde tanımlanmıştır ve her safha değişiminde katı incelemeler, yoğun dokümantasyon ve yönetim onayı gibi kontrollerle proje yoğun bir denetim altında icra edilir. Bu nedenle uygulanması katı bir disiplin gerektirir. Bu model, "önden büyük tasarım" felsefesini benimser ve kodlama safhasından önce büyük bir analiz ve tasarım yapılır [13]. Sistem ve yazılım gereksinimleri, gerekli her şeyin baştan bilindiği ve değişiklik olmayacağı faraziyeleri kabul edilerek en başta belirlenir [12]. Bu nedenle belirsizlik içermeyen tam anlaşılmış ve değişme ihtimali olmayan projeler için daha uygundur [2] çünkü şelale modelinin en büyük zayıflığını bu faraziyeler oluşturmaktadır. Müşteri, gereksinimler tanımlanırken projenin içindedir, analiz, tasarım ve kodlama safhalarında görünmez, test ve entegrasyon safhasında yeniden ortaya çıkar [12]. Şelale modelinde en çok planlama, zaman çizelgeleri, hedef tarihler, bütçe ve bütün sistemin bir defada kullanıma alınması konuları üzerinde durulur.

Bu yaklaşımın temel amacı proje süresince değişikliklere izin vermeyerek, kapsam, zaman ve kaynaklar gibi faktörleri baştan sabitlemek ve büyük bir risk faktörü olan değişimi etkisiz kılmaktır [14]. Şelale modelinin ortaya çıktığı zaman ve şartlar değerlendirildiğinde, iş süreçleri bilgi sistemlerine günümüzde olduğu kadar bağımlı değildi ve projelerin aceleyle bitirilmesi gerekmiyordu. Teknolojik gelişme ve iş yaşamındaki değişim günümüzde olduğu kadar hızlı değildi bu yüzden projeler günümüzde olduğu kadar hızlı bir tempoyla ilerlemiyordu. Teknoloji ve kaynak kısıtlarından dolayı bilgi sistemleri donanım merkezliydi ve yazılımın ana maksadı, depolama kapasitesi ve işlemci gücü gibi kısıtlı donanımsal kaynakları en optimum şekilde kullanabilmekti [15]. Bu nedenlerle ortaya çıktığı zamanlarda şelale modeli daha çok bir risk yönetim metodu olarak görülmekteydi ve bu riskin yönetilebilmesi için gerekli görülen ve daha sonra şelale modeli ile birlikte geleneksel proje yönetim dünyasının temeli olan hususlar şunlardı:

- Planlama ve bütçelemenin doğru yapılabilmesi için bütün gereksinimler erkenden tanımlanmalıydı.
- Gereksinimleri erkenden tanımlamak ve değişiklik olmayacak şekilde sabitlemek kapsam aşımını önleyecekti.
- Gelecekte referans oluşturması için bütün tasarım ve karar verme süreçleri belgelenmeliydi.
- Safhalar arası yoğun incelemeler bütün paydaşların süreçten haberdar olmasının yanında doğabilecek problemlerin erken tespit edilmesini sağlayacaktı.
- Mimarinin uygun olduğundan emin olunabilmesi için, tasarım faaliyetleri başlamadan önce bütün gereksinimler belgelenmeliydi.
- Tüm sistemin kabiliyetlerinin değerlendirildiğinden emin olunabilmesi için test işlemlerinden önce bütün kodlama işlemleri tamamlanmalıydı.
- Sistem uygulamaya koyulmadan önce bütün gereksinimler karşılanmalıydı [12].

2.3. Faydaları (Benefits)

- Projenin başlangıcında gereksinimlerin açık, anlaşılır ve tam olarak tanımlanması, daha doğru zaman ve maliyet tahmini yapılmasını sağlayarak projedeki belirsizlikleri azaltır.
- Çok kullanılan bir metot olduğu için genelde teoride iyi bilinir.
- Kimin ne zaman ne yapması gerektiği katı bir şekilde tanımlandığı ve takip edildiği için kendi başına iş yapma yeterliliği ve tecrübesi olmayan proje elemanları, proje yöneticileri ve performansı iniş çıkışlı proje ekipleri için idealdir.
- Bir safhaya başlanmadan önce bir önceki safha tamamlandığı için proje ilerleyişinin takibi

kolaydır. Ayrıca eş zamanlı ilerleyen safha olmaması kaynak takibini de kolaylaştırır.

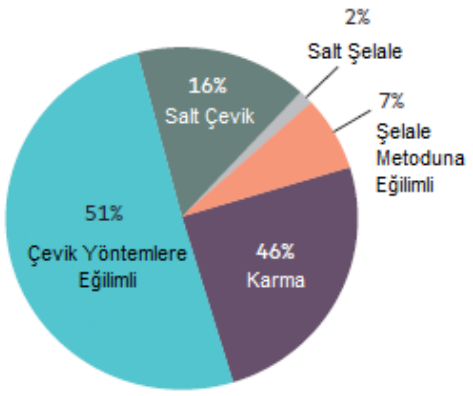
- Dokümantasyonun devamlı bir faaliyet olması proje sonunda dokümantasyon maksadıyla geçmişe dönme emeğini engellediği gibi projenin her hangi bir zamanında dokümantasyonun mevcut duruma göre tam olmasını sağlar ve bu dokümantasyon üzerinden projenin mevcut halinin değerlendirilebilmesine imkân verir.
- Şelale modeli disiplinli ve istikrarlı bir modeldir ve proje elemanlarına projenin iyi bir şekilde yönetildiği ve her şeyin kontrol altında olduğu hissiyatını verir.
- Müşterinin projeye ne zaman ve ne kadar katkı yapacağı bellidir ve proje boyunca zaman ayırması beklenmez.
- Fonksiyonel organizasyonlarda birden fazla projede görev alan proje elemanlarının işlerinin üst üste binmesini engeller. Örneğin analiz çalışması tamamlandıktan sonra başka bir projede görev almaya devam eden bir elemanın geri dönüşü gerekmez.
- Hazırlanan detaylı ve kapsamlı dokümantasyon bir iletişim aracı olarak kullanılabilir ve özellikle büyük projelerde ve bütün elemanların aynı ortamda bulunmadığı projelerde yüz yüze iletişim ihtiyacını azaltır. [9,13,16]

2.4. Kısıtları (Limitations)

- Değişiklik, geliştirme veya düzeltme maksatlı geriye dönüşler şelalede akıntıya karşı yüzmeye benzetilir, zor ve maliyetlidir. Bu yüzden değişikliklere cevap veremez ve süreç içerisinde gelen istekler çoğunlukla kabul edilmez.
- Herhangi bir safhada tespit edilmemiş bir olumsuzluk sonraki bütün safhaları etkiler.
- Yapısal özelliği ve sıkı kontrolleri nedeniyle esnek olmayan, yavaş ve hantal bir doğası vardır.
- Gereksinimlerin proje başında tam ve anlaşılır bir şekilde tanımlanmasının beklenmesi çok gerçekçi olmayabilir ve müşterilerin ihtiyaçlarını tam ve doğru ifade edememesi proje sonunda eksik veya yanlış bir ürün elde edilmesine neden olabilir.
- Geliştiricilerin gereksinimleri müşteri veya kullanıcılarla yüz yüze görüşerek anlamaları yerine analiz ve tasarım safhalarında hazırlanan dokümanlardan okumaları, gereksinimlerin anlaşılmasını zorlaştırır, geliştiricilerin girdi yapmasına imkân vermez.
- Gereksinim tutarsızlıkları, eksik sistem bileşenleri ve beklenmeyen geliştirme ihtiyaçları, ancak tasarım veya kodlama aşamasında belirlenebilir.
- Sistem performansı tüm kodlama tamamlanana kadar ölçülemez ve tüm sistem testine kadar hatalar tespit edilemez.
- Yoğun ve güncel dokümantasyon zaman ve emek gerektiren bir faaliyettir ve genel olarak geliştiriciler tarafından gereksiz bir yük olarak görülür.
- Yazılı sistem tanımlamalarının kullanıcı veya müşteriler tarafından okunarak tam olarak anlaşılması ve uygulanması zordur.
- Bir safhanın tamamlanmadan sonraki safhanın başlamaması nedeniyle, tüm proje elemanlarını ilgilendirmeyen gecikmeler dahi bütün proje elemanları için zaman israfına neden olabilir.
- Yüksek yönetim giderleri küçük projeler ve küçük takımlar için gereksiz maliyetlerdir.
- Yazılım geliştirme faaliyetleri yaratıcılık gerektiren faaliyetlerdir ve şelale modelinin yüksek disiplinli yaklaşımı yaratıcılığı olumsuz etkiler. [9,13,17,18],

3. ÇEVİK YÖNTEMLER (AGILE METHODOLOGIES)

Çevik yazılım geliştirme, kendi kendini organize eden takımlar tarafından, son derece işbirlikçi bir anlayışla, "yeteri kadar" resmiyet içeren etkili bir yönetişim çerçevesinde, zamanında, uygun maliyetli ve paydaşların değişen ihtiyaçlarını karşılayan yüksek kalitede çözümler üreten yinelemeli ve artımlı (evrimsel) bir yaklaşımdır [19]. Çevik yöntemler, geleneksel yöntemlerin yetersiz kaldığı değerlendirilen konular için alternatif çözümler olarak ortaya çıkmıştır. HP'nin 2017 yılında yaptığı bir araştırmaya göre organizasyonlar yazılım proje yönetiminde daha çok çevik yöntemleri tercih etmektedir [20]. İlgili araştırma sonucu elde edilen veriler Şekil 2'de sunulmuştur. AgileTurkey tarafından yayınlanan Türkiye Çeviklik Raporuna göre de incelenen organizasyonların %72'si çevik yöntemleri kullanmaktadır [21].



Şekil 2. Organizasyonlarda uygulanan geliştirme yöntemleri [20]
(Development methods used in organizations)

3.1. Tarihçe (History)

Extreme Programlama (XP) [22], Scrum [23],eXtreme Test [24],Crystal Metodoloji Ailesi [25], Dinamik Sistem Geliştirme Metodu (DSDM) [26],Uyarlanır Yazılım Geliştirme (ASD) [27] ve Özelliğe Dayalı Geliştirme

(FDD) [28] gibi çevik yöntemleri meydana getiren başlıca metodolojiler, 1990'ların ikinci yarısından itibaren ortaya çıkmaya başladılar [14]. 2001 yılında, çevik metotların yaratıcısı ve önde gelen uygulayıcılarından on yedi yazılım mühendisi, dokümantasyona dayalı geleneksel yazılım geliştirme süreçlerine bir alternatif bulabilmek ve uyguladıkları metodolojilere dayalı ortak bir zeminde bulaşabilmek maksadıyla bir araya geldiler [29]. Kendilerine çevik ittifak adını veren bu uzmanlar, çalışmanın sonucunda çevik yöntemlerin temel ilke ve değerlerini kararlaştırarak Çevik Bildiriyi yayınladılar [30]. Yapılan araştırmalara göre çevik yöntemlerin benimsenmesi dünyada ve Türkiye'de 2010 yılından itibaren büyük bir yükselişe geçmiştir [20],[21].

3.2. İlkeleri ve Özellikleri (Principles and Properties)

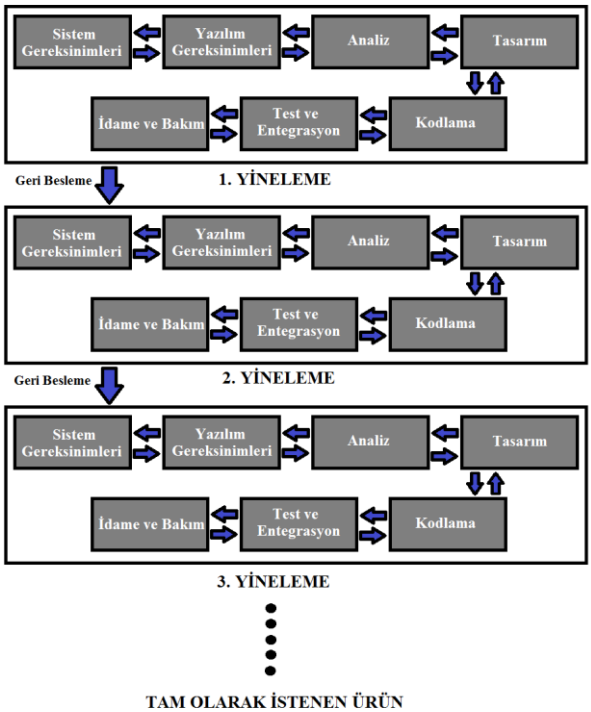
2001 yılında Çevik İttifak tarafından yayınlanan Çevik Bildiri felsefesine göre:

- Süreçler ve araçlardan ziyade bireyler ve etkileşimlere,
- Kapsamlı dokümantasyondan ziyade çalışan yazılıma,
- Sözleşme pazarlıklarından ziyade müşteri ile işbirliğine,
- Bir plana bağlı kalmaktan ziyade değişime karşılık vermeye değer verilir [30].

Yukarıda temel değerleri belirtilen çevik yöntemlerin, bu felsefen doğan on iki ilkesi vardır. Bu ilkeler:

- En önemli öncelik değerli yazılımın erken ve devamlı teslimini sağlayarak müşterileri memnun etmektir.
- Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir. Çevik süreçler değişimi müşterinin rekabet avantajı için kullanır.
- Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır.
- İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdırlar.
- Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır.
- Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüz yüze iletişimdir.
- Çalışan yazılım ilerlemenin birincil ölçüsüdür.
- Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.
- Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır.
- Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca arttırılması sanatı, olmazsa olmazlardandır.
- En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar.
- Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler [30].

Çevik yöntemlerde uygulanan yinelemeler, önceki yinelemelerde elde edilen tecrübeler ve tespit edilen aksaklıklar doğrultusunda şekillenir. Yapılması gereken görevler, taşıdıkları iş değerine göre önceliklendirilir. Yapılacak işin en iyi nasıl yapılacağını, mevcut kaynaklar ve kısıtlar çerçevesinde proje takımı kendi belirler. Takım belli görevleri belli sürelerde (yineleme süresi içinde) tamamlamalıdır. Yineleme sonunda teslim edilecek ürünü meydana getirmekten sorumlu olan güçlü ve zayıf yanlarıyla takımdır. Bu nedenle takım içi işbirliği önem arz eder. Bu kapsamda çevik yöntemlerin dayandığı temel esaslar, *deneysellik, önceliklendirme, kendi kendini örgütleme, zaman çerçevesi, iş birliği* şeklinde sıralanabilir [31]. Çevik yöntemlerde yazılım geliştirme faaliyetleri yinelemeli safhalar halinde uygulanır. Bu yinelemeler sonucunda kullanılabilir ürün ortaya çıkarılır ve müşteri veya kullanıcıların geri beslemeleri ışığında ve varsa değişen gereksinimler doğrultusunda geliştirme süreçleri tekrarlanır. Bu yinelemeli süreçler müşterinin tam olarak istediği ürün ortaya çıkana kadar sürdürülür. Çevik yöntemlerde uygulanan safhalar Şekil 3'te gösterilmiştir.



Şekil 3. Çevik Yöntemlerin Genel Uygulaması
(General Implementation of Agile Methods)

Başlıca çevik metodolojilere ait kullanım oranları Tablo 3'de sunulmuştur. Gerek dünya genelinde gerekse Türkiye'de en popüler metodoloji Scrum olmakla birlikte XP ve Kanban diğer metodolojilere göre öne çıkmaktadır. Scrum yönteminin Kanban ve XP ile beraber kullanıldığı ve organizasyonların kendi ihtiyaçlarına göre uyarladığı karma metodolojiler de çevik yöntemlerin başlıca kullanım şekillerindendir. VersionOne [32] araştırmasında katılımcılardan tek metodoloji belirtmeleri istenirken, AgileTurkey [21] ve Forrester [33] araştırmasında çoklu seçeneğe izin verilmiştir.

Tablo 3. Başlıca Çevik Yöntemlerin Uygulanma Oranları
(Implementation Rates of Primary Agile Methodologies)

	VersionOne [32]	**Forrester [33]**	**AgileTurkey [21]**
Scrum	% 58	% 86	% 65
XP	% 1	% 29	% 7
Kanban	% 5	% 57	% 32
Scrum/XP Karma	% 10	-	% 8
Scrum/Kanban Karma	% 8	-	% 7
Çoklu Karma Uyarlamalar	% 8	-	% 8

3.3. Faydaları(Benefits)

- Çevik yöntemlerin felsefesi gereği proje süresince değişikliklerin hoş karşılanması ve hatta teşvik edilmesi nedeniyle ve yinelemeli süreç yapısı sayesinde esneklik yüksektir ve değişen şartların yönetilmesi kolaydır.
- Ayaküstü kısa toplantılar ve yüz yüze iletişim felsefesi ile çevik yöntemlerde uygulanan insan ve iletişim merkezli süreçler sayesinde proje elemanlarının moral ve motivasyonu artar, değerli bilgi ve görüşlerin paylaşımı ile üretkenlik seviyeleri yükselir.
- Kısa süreli yinelemeler ve yineleme sonucunda teslim edilen çalışır ürün sayesinde projenin ilerlemesi daha sağlıklı bir şekilde gözlemlenir ve proje riski azalır, muhtemel hatalar daha erken tespit edilir, teslimat öngörülebilirliği artar ve ürün pazara daha çabuk ulaşır.
- Müşteri veya kullanıcıların proje sürecinde proje elemanları ile yüz yüze iletişim kurabilmeleri sayesinde iş süreçleri ile yazılım geliştirme süreçlerinin entegrasyon seviyesi yükselir, istenen özelliklerin tam olarak ifade edilebilmesiyle yazılımın fonksiyonel kalitesi artar ve müşteri memnuniyeti sağlanır.
- Çevik yöntemler, uygulanan safhalar arası geçişlere ve geri dönüşlere müsait olduğu için süreç devam ederken tespit edilen geriye dönük aksaklıklar kolaylıkla düzeltilebilir.
- Genel olarak üretilen yazılımın sadece ürün özellikleri dokümantasyona çevrilir ve diğer süreçler hariç tutulur. Hafif dokümantasyon iş yükünü azaltır. [32,33,34,35]

3.4. Kısıtları (Limitations)

- Çevik yöntemlerde süreçler iletişim merkezli olduğu için proje elemanlarının birbirleriyle veya müşteriyle aynı ortamda bulunmaması, proje gelişimini olumsuz etkiler. Video konferans gibi uzaktan iletişim kurulabilecek teknolojiler kullanılabilir fakat bu da gerekli altyapı ve teknoloji nedeniyle maliyetleri arttırır.
- Alt yüklenicinin teklif verebilmesi ve bir kontrat yapılabilmesi için istenenlerin kesin olması ve değişmemesi gerekir. Çevik yöntemlerin belirsiz ve değişken ortamı alt yüklenici uygulamalarını zorlaştırır.
- Fonksiyonel olarak bütünlük gerektiren büyük ve karmaşık sistemler yinelemeler için uygun parçalara bölünemeyebilir.
- Zaman ayıramama veya isteksiz olma gibi nedenlerle müşteri veya son kullanıcıların proje içerisinde olma oranı düşükse gereksinimler doğru biçimde tanımlanamayabilir.
- Değişime açık olması nedeniyle projenin kaynak ve kapsam planlaması zordur.
- "Gerektiği kadar" anlayışı ile hazırlanan dokümantasyonun yetersiz olması durumunda projeye sonradan katılan geliştiriciler ve bakım-idame safhaları olumsuz etkilenir.
- Takımlar kendi kendini organize ettiği için bireylerin yetenekli, tecrübeli ve çevik süreçlere hâkim olması gerekir. Yeteneksiz, tecrübesiz elemanlar takım performansını düşürerek proje gelişim sürecini olumsuz etkiler.
- Önceliklerini tam olarak belirleyemeyen ve yapılacaklar listesindeki öncelikleri sık sık değiştirmek isteyen müşteriler planlamaları olumsuz etkileyebilir.
- Çevik yöntemlerle, spesifik problemler veya fonksiyonlara göre tanımlanan gereksinimleri karşılayacak yeterlikte çözüm geliştirildiği için, ürünlerin diğer projelerde yeniden kullanılabilirliği olmaz.
- Yazılım hatasının insan hayatını tehlikeye attığı veya büyük maddi zararlar doğurduğu hata toleransı olmayan sistemler için çevik yöntemlerin test ve kalite kontrol yöntemleri yeterli olmayabilir.
- Çevik yöntemlerde müşteri faydası ön planda tutulur fakat bu genel olarak müşterinin istediği fonksiyonların karşılanmasıdır. Yalın geliştirmenin bir kalıtımı olarak basitlik felsefesi uygulandığı için kullanıcı ara yüzü ve kullanıcı deneyimi gibi kullanıcı tatmini ile ilgili konular göz ardı edilebilir. [17,34,36,37]

4. KARŞILAŞTIRMA (COMPARISON)

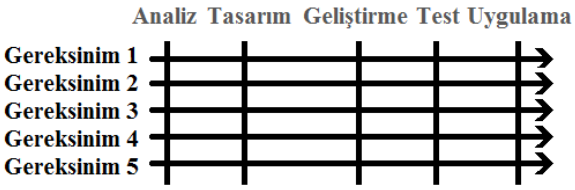
Geleneksel şelale modeli ile çevik yöntemler arasında yapılan işin doğası gereği birçok benzerlik mevcut iken felsefelerindeki fikir ayrılığı nedeniyle yaklaşım tarzı ve uygulanan tekniklerde farklılıklar bulunmaktadır. Projelerde şelale modeli ya da çevik yöntemlerin uygulanması, Tablo 2'de belirtilen faktörler açısından farklı sonuçlar doğurabilir. Projelerin başarı oranları uygulanan metodolojiye göre de farklılık göstermektedir. 2011-2015 yılları arasında 10000'in üzerinde yazılım projesinde, uygulanan metodolojiye göre elde edilen başarı oranları Tablo 4'te sunulmuştur. Elde edilen veriler ışığında çevik yöntemlerin başarı oranlarının şelale modelinin başarı oranlarının yaklaşık 4 katı, başarısızlık oranlarının ise 3'te 1'i olduğu görülmektedir. Buna rağmen çevik yöntemlerin başarı ihtimalinin şelale modeline göre her zaman daha fazla olacağı söylenemez. Proje başarısı için proje şartlarına uygun metodoloji seçilmelidir.

Tablo 4. Uygulanan Metodolojiye Göre Yazılım Projeleri Başarı Oranları [8]
(Software Projects Success Rates According to the Methodology Implemented)

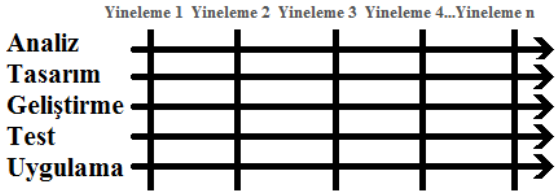
	Şelale Modeli	**Çevik Yöntemler**
Başarılı Projeler	% 11	% 39
Sorunlu Projeler	% 60	% 52
Başarısız Projeler	% 29	% 9

4.1. Şelale Modeli ve Çevik Yöntemler Arasındaki Benzerlikler ve Farklılıklar (Similarities and Differences Between Waterfall Model and Agile Methodologies)

Şelale modeli ile çevik yöntemler aynı amaç doğrultusunda ortaya çıkmış yaklaşımlardır. Her iki yaklaşımda da planlama önemli bir yer tutar. Şelale modelinde bütün süreci kapsayan uzun vadeli ve detaylı bir planlama yapılırken, çevik yöntemlerde değişen şartlara cevap verebilmek için uzun vadeli planlamadan ziyade kısa vadeli ve daha detaylı planlamalar yapılır. Bütün süreç sürüm planlarına, sürüm planları yineleme planlarına, yineleme planları ise günlük planlara bölünür. Böylece daha detaylı hedefler belirlenebilir. Her iki yaklaşımda da süreç içinde detaylı incelemelere önem verilir. Şelale modelinde bu incelemeler safhalar arası geçişte yapılan detaylı kontroller ve dokümantasyon vasıtasıyla yapılırken, çevik yöntemlerde yinelemeler sonucu üretilen çalışan ürünler ve müşterinin bütün sürecin içinde bulunması ile detaylı bir inceleme sağlanmış olur. İki metodolojide de yapılan faaliyetler ortaktır. Bu faaliyetlerin icra şekilleri farklıdır. Şelale modelinde bu faaliyetler her gereksinime uygulanan safhalar olarak karşımıza çıkar ve ana sürecin alt bileşenleridir. Çevik yöntemlerde ise her yinelemenin uygulandığı faaliyetler olarak karşımıza çıkarlar ve ana sürecin alt bileşeni yinelemeleri oluşturan faaliyetlerdir. Metodolojilere ait temel süreç akışları Şekil 4 ve Şekil 5'te gösterilmiştir.



Şekil 4. Şelale Modeli Süreç Akışı [12]
(Waterfall Model Process Flow)



Şekil 5. Çevik Yöntemler Süreç Akışı [12]
(Agile Methodologies Process Flow)

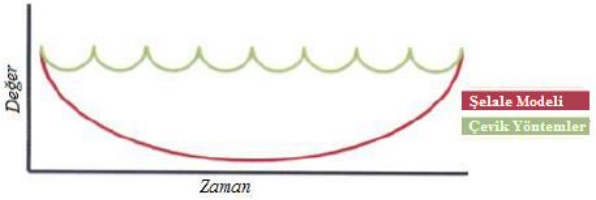
Şelale modelinde gereksinimlerden oluşan paket proje sürecinde safhalar üzerinde ilerlerken çevik yöntemlerde safhalardan oluşan paket yinelemeler üzerinde ilerler. Projenin başında tanımlanan gereksinimler ve önceliklere göre teslimat yapılması nedeniyle şelale modelinde devamlı geriye doğru bir bakış vardır. Çevik yöntemlerde ise evrilen gereksinim ve önceliklere göre teslimat yapılması nedeniyle devamlı ileriye doğru bir bakış vardır. Yapısal süreçlere dayalı olması nedeniyle şelale modeli daha çok bir mühendislik yaklaşımı sunarken kişilere ve iletişime dayalı olması nedeniyle çevik yöntemlerde yaratıcılık ön plandadır. [12,31,38]

Çalışmanın bundan sonraki bölümünde; Tablo 2'de belirtilen CHAOS raporları başarı faktörleri, literatür araştırmaları ve yazılım projelerinde edinilen kişisel tecrübeler ışığında proje başarısına etki ettiği değerlendirilen konularda, şelale modeli ve çevik yöntemler için karşılaştırma ve değerlendirmeler sunulacaktır.

4.2. Proje Yönetimi ve Kontrolü (Project Management and Control)

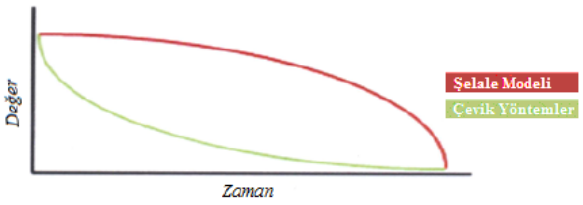
Yönetim ve Roller: Merkezi ve hiyerarşik yapılı yönetim sistemleri, şelale modeli ile uyumludur. Liderlik, dayanışma ve çalışanların işlerini kolaylaştırıcı anlayışta yönetim tarzı ise çevik yöntemler için uygun ortamlardır [41]. Roller açısından da iki metodoloji arasındaki en temel fark yöneticilerdir. Çevik yöntemlerde takımı motive eden, engelleri kaldıran, süreçleri kolaylaştıran bir koçluk ortaya koyulurken, şelale modelinde klasik komuta-kontrol yönetim tarzı uygulanır [45]. Kendi kendini organize bağlamında proje yönetimi faaliyetleri liderler, müşteriler ve takım elemanları arasında paylaşılır [46].

Görünürlük: Şelale modelinde proje görünürlüğü uygulanan safhalar arasında yapılan kontroller ve dokümantasyonla sağlanırken, çevik yöntemlerde yinelemeler sonunda ortaya çıkarılan çalışır ürünler ve ürün iş listeleri üzerinden sağlanır. Çevik yöntemler bu kapsamda müşteriden doğrudan çalışan ürünün onayını alarak proje ilerlemesini gerçekleştirdiği için şelale modeline göre avantaj sağlar. Şelale modeli ve çevik yöntemlerin görünürlük grafiği Şekil 6'da sunulmuştur. Çevik yöntemlerin görünürlük olarak avantajlı olduğu projeler, çalışan ürünün kullanıcı ara yüzlü ürün olduğu projelerdir. Kullanıcı tarafından değerlendirilemeyen ya da fonksiyonel olarak küçük parçalara ayrılamayan projelerde dokümantasyon daha iyi bir görünürlük sağlayacağı için şelale modeli daha uygundur [36].



Şekil 6. Görünürlük Karşılaştırması [48]
(Comparison of Visibility)

Karmaşıklık: Üretilecek yazılımın karmaşıklık seviyesi arttıkça analiz ve yönetim ihtiyacı da artar. Örneğin teknik ve yönetsel açıdan en karmaşık projeler olan gerçek zamanlı savunma sistemleri [39] için kodlama aşamasına kadar detaylı analizler yapılmış olmalıdır. Bu şartlarda şelale modeli, plan tabanlı bir yaklaşım olduğu için en iyi seçenektir. Çevik yöntemler en erken zamanda müşteri için değer üretmeyi amaçlar ve eldeki mevcut girdilerle kodlamaya geçilerek yinelemeler içerisinde durumun olgunlaşması amaçlanır. Bu nedenle çevik yöntemler daha az analiz gerektiren nispeten daha az karmaşık sistemler için daha uygundur. Detaylı analiz ve planlama şelale modelinin çevik yöntemlere göre avantajı iken, yineleme sonlarında yapılan çalışır ürün teslimleri vasıtasıyla yapılabilen gözden geçirme, alınan geri besleme ve değişim yapabilme imkânı, proje sürecinde karmaşıklık ile başa çıkabilme açısından çevik yöntemlerin şelale modeline göre avantajıdır. Şelale modeli ve çevik yöntemlerin proje sürecinde karmaşıklık grafiği Şekil 8'de sunulmuştur.



Şekil 8. Proje Sürecinde Karmaşıklık Karşılaştırması [48]
(Comparison of Complication During Project)

Büyüklük: Proje büyüklüğü ile ilgili genel kanı, büyük projelerin detaylı dokümantasyon ile daha iyi yönetilebileceği, küçük projelerin ise katılımcılar arasındaki doğrudan iletişim sayesinde daha etkin sürdürülebileceği, bu nedenle büyük projelerin şelale, küçük projelerin ise çevik yöntemler için daha uygun olduğu yönündedir [41]. Proje büyüdükçe çalışan sayısı ve bütçe artar. Bu da daha çok koordinasyon gerektirir. Şelale modeli, kapsamlı planlama, dokümantasyon ve büyük takımlar için daha iyi iletişim ve koordinasyon süreçleri sunarak bu ortamı destekler [17].

Proje büyüdükçe çevik yöntemlerin maruz kaldığı dezavantajlara karşı tedbirler alınmakta ve büyük ölçekli projelerde kullanılmak üzere özelleşmiş geniş kapsamlı çevik metodolojiler ortaya çıkmaktadır. Geniş kapsamlı çevik yöntemlerin en önde gelenleri Disciplined Agile Delivery (DAD), Large-Scale Scrum (LeSS), ve Scaled Agile Framework (SAFe) metodolojileridir [55].

2011-2015 yılları arasında yazılım projelerinde elde edilen başarı durumlarının uygulanan metodoloji göz önüne alınmadan proje büyüklüğüne göre dağılımı Tablo 5'te, proje büyüklüğü ve uygulanan metodolojiye göre başarı durumları Tablo 6'da sunulmuştur.

Tablo 5. Yazılım Projeleri Başarı Oranlarının Proje Büyüklüğüne Göre Dağılımı [8]
(The Resolution of Software Projects Success Rates by Project Size)

Proje Büyüklüğü	Başarılı	Sorunlu	Başarısız
Büyük	% 8	% 25	% 41
Orta	% 30	% 59	% 48
Küçük	% 62	% 16	% 11
Toplam	% 100	% 100	% 100

Tablo 6. Proje Büyüklüğüne ve Uygulanan Metodolojiye Göre Yazılım Projeleri Başarı Oranları [8]
(Software Projects Success Rates according to Methodology and Project Size)

Proje Büyüklüğü	Metodoloji	Başarılı	Sorunlu	Başarısız	Toplam
Büyük	**Çevik**	% 18	% 59	% 23	% 100
	Şelale	% 3	% 55	% 42	% 100
Orta	**Çevik**	% 27	% 62	% 11	% 100
	Şelale	% 7	% 68	% 25	% 100
Küçük	**Çevik**	% 58	% 38	% 4	% 100
	Şelale	% 44	% 45	% 11	% 100

Tablo 5'te görüleceği üzere başarılı yazılım projelerinin %92 gibi çok büyük bir oranını küçük ve orta ölçekli projeler oluştururken sorunlu projelerin %84, başarısız projelerin ise %89 unu orta ve büyük ölçekli projeler oluşturmaktadır. Proje boyutu büyüdükçe başarı şansı büyük oranda azalmaktadır. Çevik yöntemlerin

uygulandığı projelerdeki başarı oranının, şelale modelinin uygulandığı projelerdeki başarı oranından daha yüksek olduğu Tablo 4'te görülmekteydi. Karşılaştırmanın içine proje boyutu girdiğinde de sonucun değişmediği Tablo 6'da görülmektedir. Genel kanının [41] aksine elde edilen sonuçlar çevik yöntemlerin her türlü proje büyüklüğünde daha başarılı olduğunu göstermektedir.

Altyüklenici Kullanımı: Altyüklenici kullanılacak durumlarda, yapılması istenen ürün veya ürün parçasının detaylı tanımlanması şarttır. Altyüklenici kendisinden istenen ürüne göre bir maliyet, zaman ve efor tahmini hazırlayarak bir teklif verir ve sözleşme yapılır [36]. Maliyetleri arttıracağı için ihtiyaçlarda doğabilecek değişiklikler kabul edilmez. Bir tedbir alınmadığı müddetçe ihtiyaç sahibi müşteri ve proje takımı ile alt yüklenici arasında herhangi bir iletişim olmaz. İletişim olmadan müşteri etkileşimi ve proje takımı ile dayanışmadan söz edilemez. Bu nedenlerle çevik yöntemler, altyüklenici kullanımına uygun değildir. Tanımlanmış isterler ve zaman kısıtları nedeniyle şelale modeli alt yüklenici kullanımına daha müsaittir.

4.3. Müşteriler(Customers)

Müşteri katılımının istenen seviyede sağlanabildiği durumlarda çevik yöntemler proje sonucunda elde edilecek yazılımın nitelikleri açısından şelale modeline göre daha avantajlıdır. Gereksinimlerin ve iş süreçlerinin sağlıklı bir şekilde tanımlanması, ara teslimatlar ve geri beslemeler sayesinde elde edilen ürünün tam olarak istenen ürün olması sağlanabilir. Bunun yanında proje sürecine müşteri katılımının sağlanabilmesi için müşteri açısından istek, zaman ve bilgi gerekir. Proje elemanlarıyla birlikte proje süresince çalışacak eleman bulundurmak müşteri için külfetli olabilir ve istenmeyebilir. Geleneksel proje yönetimine alışkın müşteriler proje başında ihtiyaçlarını tanımlayıp bir daha zaman ayırmak istemeyebilir [40] çünkü müşterilerin kendi günlük işleri her zaman daha önemlidir [56]. Aynı yerde ve zaman diliminde olmayan müşteriler ile yüz yüze iletişim kurulamaması faaliyetleri aksatabilir. Müşteri adına projeye katılan personel, iş süreçlerine hâkim olmayabilir. Böyle durumlarda gereksinimlerin sağlıklı tanımlanamaması ve teslim edilen ürünlerden geri besleme alınamaması muhtemeldir. Bu gibi nedenlerle müşteri katılımının sağlıklı olarak sağlanamayacağı ortamlar açısından şelale modeli daha uygundur çünkü çevik yöntemlerin en önemli dayanaklarından biri müşteri katılımı iken şelale modelinde proje başlangıcından sonra teslime kadar bir müşteri katılımı beklenmez.

4.4. Organizasyon Yapısı (Organizational Structure)

Önde gelen organizasyon teorisyenlerinden Henry Mintzberg oranizasyon çeşitlerini, Girişimci Organizasyon, Makine Organizasyon, Profesyonel Organizasyon, Farklılaşmış Organizasyon, Yenilikçi Organizasyon, Görev Organizasyonu ve Politik Organizasyon olarak yedi ana grup altında toplamıştır [52]. Yazılım geliştirme organizasyonları genel olarak girişimci, farklılaşmış ve yenilikçi organizasyonlardır fakat makine ve profesyonel organizasyonlarda da yazılım projeleri yürütülebilir [13]. Görev organizasyonu ve politik organizasyonlar yazılım proje yönetimi açısından karşılaşılacak organizasyon tipleri değildir Bu nedenle ilk beş organizasyon ele alınacaktır.

Girişimci Organizasyon: Bu tip organizasyonlar esnektir ve düşük hiyerarşiye sahiptir. Basit ve dinamik bir ortamları vardır. Genellikle kuruluş aşamasındadırlar. Çoğu girişimci organizasyon yüksek derecede tepkiseldir ve görev anlayışları yüksek seviyededir. Çoğu kez organizasyonun başı çalışanlarla birebir iletişim halindedir. En büyük olumsuzlukları dengesiz ve sağlamlaşmamış iş ortamlarıdır. İletişim ortamının müsait ve yapılarının esnek olması nedeniyle ve iş süreçlerinin değişen şartlara uyum sağlaması gerektiğinden bu tip organizasyonlarda çevik yöntemlerin yinelemeli ve değişime açık yaklaşımlarının uygulanması şelale modeline göre daha uygundur fakat genellikle yeni oluşumlar oldukları için yetenekli ve tecrübeli çalışanları istihdam etme konusunda sıkıntı yaşayabilirler. [13,53]

Makine Organizasyon: Bu tip organizasyonlar fabrikalar gibi hiyerarşik ve olgunlaşmış yapılardır ve değişime direnirler. Durağan ve stabil bir ortamları vardır. Merkezi bir bürokrasi ve formel süreçler işler. Çalışanlar fonksiyonel gruplara ayrılmıştır ve genelde hep aynı işi yaparlar. En büyük olumsuzlukları kontrol saplantısı nedeniyle çalışanlarla ilgili problemlere açık olmaları ve değişen şartlara uyum sağlayamamalarıdır. Bu tip organizasyonlar her yönüyle şelale modeli için biçilmiş kaftandır ve çevik yöntemlerin uygulanması neredeyse imkansızdır. [13,53]

Profesyonel Organizasyon: Bu tip organizasyonlar bürokratik fakat merkezi olmayan, hastane, üniversite benzeri organizasyonlardır. Otonom olarak çalışan profesyonel personel gruplarından oluşurlar. Genel stratejileri stabildir fakat gerekli spesifik detaylar sık sık değişebilir. Çalışanlar işlerini iyi şekilde bilirler ve yönetim kademesi iş detaylarına karışmaz. Profesyonel organizasyonlarda koordinasyon konusunda problemler yaşanabilir. Şelale modelinin hiyerarşik komuta-kontrol yapısı bu tip organizasyonlar için uygun değildir. Çalışanların profesyonel olması ve organizasyonun kişisel otonomiye önem vermesinden dolayı çevik yöntemler bu tip organizasyonlar için daha uygun yaklaşımlardır. [13,53]

Farklılaşmış Organizasyonlar: Farklılaşmış organizasyonlar, bünyelerinde yapılan işin farklılığına göre oluşturulmuş şirket benzeri birimler bulunduran, bu birimlerin insan kaynakları, finans, hukuk gibi ortak fonksiyonlarını da kendi çatısı altında tutan holding benzeri yapılardır. Bu tip organizasyonlar için hiyerarşik yapı ve uzmanlık alanlarındaki farklılıklar nedeniyle şelale modeli daha uygundur. Her birim kendisiyle ilgili

faaliyeti icra edecek ve biten işi diğer birime iletecektir. Çevik yöntemlerde gerekli iletişim ve koordinasyon ortamının bu tip dağıtık organizasyonlarda sağlanması zordur. Uzmanlık alanlarına göre personel ayrılarak matriks yapı benzeri proje grupları oluşturulması halinde çevik yöntemler uygulanabilir. [13,53]

Yenilikçi Organizasyon: Bu tip organizasyonlar merkezi olmayan ve hiyerarşi seviyesi düşük organizasyonlardır. Fonksiyonel uzmanlardan oluşur ve karşılıklı koordinasyona dayanır. Kimin ne görev yapacağı çok net belirlenmemiştir. Bir masanın etrafında bir problemi çözmek için toplanmış uzmanlar topluluğuna benzetilebilir. Tanımından da anlaşılacağı üzere uzmanlık, koordinasyon ve otonom yapısı nedeniyle çevik yöntemlerin felsefesi ile birebir örtüşen bir organizasyon tipidir. Komuta kontrol yönetim sistemine uymaması nedeniyle şelale modeli için uygun değildir. [13,53]

4.5. Proje Elemanları (Project Members)

Roller: Şelale modeli ile çevik yöntemlerdeki roller başta yönetici rolleri olmak üzere farklıdır. Şelale modelinde proje elemanları, temel yazılım geliştirme ve mühendislik faaliyetlerini geliştirici ve test mühendisi gibi belirlenmiş uzmanlık alan adları altında gerçekleştirirler. Gereksinimlerin toplanması, netleştirilmesi, efor hesaplamaları gibi yönetim faaliyetleri ise proje yöneticileri veya iş analistleri tarafından yapılır. İşler, proje yöneticileri tarafından proje elemanlarına paylaştırılır. Çevik yöntemlerde ise yönetim faaliyetleri lider rollerdeki personelle birlikte kendi kendini organize eden takımların da katılımı ile icra edilir. Otonom takımlar, yazılım geliştirmenin geleneksel sorumluluklarının yanında hesaplamalar, planlama, görev dağılımı, projenin izlenmesi gibi proje yönetim faaliyetlerini de yürütürler. Takımların bu serbestliğinin yanında proje yönetim faaliyetleri ve karar verme süreçlerinin içine girişi, karar verme otoritesini operasyonel seviyeye indirerek süreçlerde hızlanma ve problem çözümlerinde isabet kazandırır [45].

Tecrübe ve Yetenek: Çevik yöntemler, kendi kendini organize eden takımlara dayalı olması ve nispeten daha az detaylı tasarımı koda çevirmesi nedeniyle tecrübeli ve yetenekli yazılımcılara ihtiyaç duyar. Takımın kendi kendini ve projenin verimliliğini değerlendirebilmesi gerekir. Bu nedenle takım elemanları bir karşılaştırma yapabilmek için geçmiş projelerde başarı ve başarısızlığı tecrübe etmiş olmalıdır [36]. Proje elemanları tecrübesiz ise otonom bir ortamda kontrolün kaybedilmemesi için plan tabanlı bir yaklaşımla detaylı bir şekilde hazırlanan tasarımın kodlanması daha uygun bir hal tarzıdır.

Takım büyüklüğü: Takımlar büyüdükçe elemanların iletişim etkinliği azalmaktadır. N elemanlı bir grup içindeki tek yönlü iletişim kanalı sayısı n*(n-1) dir. 8 kişilik bir takımda 56 muhtemel iletişim kanalı bulunur [2]. Ayrıca büyük takımlarda, kendi kendini yönetme becerisi zayıflarken grubun yönetim ihtiyacı doğar. Bu nedenle grup büyüdükçe çevik yöntemler gibi iletişim ve koordinasyona dayalı metodolojilerin verimliliği düşecektir. Şelale modelinde yerleşmiş bir yönetim anlayışı vardır ve yapılan katı iş tanımları nedeniyle iletişim daha az öneme sahiptir. Ayrıca süreç gereği üretilen detaylı dokümantasyon iyi bir iletişim aracıdır. Bu nedenlerle büyük takımlar için şelale modeli daha uygun bir yaklaşımdır.

Alan Bilgisi: Şelale modelinde proje başlangıcında detaylı gereksinim analizi ve tasarım çalışması yapıldığı için yazılım geliştirme ekibinde çalışan personelin yazılımın kullanılacağı iş alanı ile ilgili bilgisi olmasına gerek yoktur. Geliştirme safhasında kullanılacak girdiler önceki safhaların sonucunda elde edilen çıktılar olduğu için yapılmış analiz ve tasarıma göre kod yazılır. Çevik yöntemlerde ise müşterinin ihtiyacı olan iş süreçlerinin takım tarafından bilinmesine ihtiyaç vardır. Gerek detaylı analiz ve tasarım eksikliği gerekse değişimlere cevap verme zorunluluğu nedeniyle alan bilgisi eksikliği çevik yöntemlerde problemlere neden olabilir. Müşteri katılımı az olduğunda alan bilgisi olmayan bir takımda kullanıcı hikayelerine değer atamak ve öncelik belirlemek büyük zorluk oluşturur [40].

İş birliği: Şelale modelinde görev tanımları nettir ve görevler proje yöneticileri tarafından verilir Bu nedenle proje elemanlarının kendi işlerini yapmaları yeterlidir. İşini iyi yapan personel de bundan dolayı ödüllendirilebilir. Çevik yöntemlerde ise esas olan takımdır ve görev tanımları net değildir. Bu yüzden takımın görevini yerine getirebilmesi için takım elemanlarını çapraz fonksiyonlu olmaları gerekir. Başarılı ya da başarısız olan takımdır bu nedenle bireysel ödül veya ceza olmaz. Şelale modelinde proje süreçlerinde icra edilen görevler proje elemanlarına paylaştırılır ve safha sonunda yapılan işler toparlanarak tamamlanır. Çevik yöntemlerde ise icra edilen faaliyetler takım bütünlüğü içinde icra edilir ve proje küçük parçalara ayrılarak bütün halinde çalışılır. Bu nedenle çevik yöntemlerde görev alan proje elemanlarının projenin bütünü ve genel hedefleri konusunda farkındalık seviyesi daha yüksektir [44].

Yerleşim: Büyük şirketlerin dünyaya yayılmış olmalarının yanında farklı ülkelerdeki işgücünün daha ucuz olması sebebiyle proje bazlı işe alımlar da yapılmaktadır. Proje elemanlarının coğrafi olarak geniş bir alana dağıldığı durumlarda çevik yöntemlerin temel dayanaklarından iletişim zarar görür. Proje elemanlarının çalışma saatleri, büyük zaman dilimi farklılıkları nedeniyle eşzamanlı olamayabileceği için önemli toplantılara katılım konusunda problemler yaşanır, koordinasyonda güçlük çekilir. Telekomünikasyon vasıtalarıyla sağlanan iletişimde yakınlık hissi yüz yüze iletişimdeki kadar yoğun olmaz ve dayanışma azalır. Şelale modelinde yerleşim çevik yöntemler kadar önemli değildir çünkü projenin belli parçaları belli ekiplere bölünebilir fakat çevik yöntemler bu tarz bölünmelere izin vermez [43].

Ekip Çalışması: Ekip çalışmasının kalitesini belirleyen alt başlıklar, iletişim, koordinasyon, üye katkısı dengesi, karşılıklı destek, çaba ve uyumdur [50]. Ekip çalışmasının şelale modeli ve çevik yöntemlerdeki kalitesinin karşılaştırılması için bu alt maddelerin karşılaştırılması gerekir. İletişim, şelale modelinde formeldir ve daha çok proje yöneticisine yazılı durum raporları şeklindedir. Çevik yöntemlerde ise enformel bir iletişim mevcuttur ve ayaküstü toplantılar, kısa sohbetler, ekran karşısında anlatım şeklinde gerçekleşir. Koordinasyon şelale modelinde proje yöneticisinin kontrolü altındadır ve karar verme, tahminlerin yapılması, önceliklerin belirlenmesi ve görev dağılımı proje yöneticisi tarafından icra edilir. Çevik yöntemlerde bu uygulamaların hepsi otonom takımlar tarafından gerçekleştirilir. Şelale modelinde üye katkısının dengeli olması söz konusu değildir. Proje yöneticisi kimin ne iş yapacağına karar verir ve proje elemanları kendilerine verilen görevleri yapmakla mükelleftirler. Çevik yöntemlerde çapraz fonksiyonlu takımlar içinde bütün takım elemanlarının katkısı beklenir ve bu husus günlük toplantılarda koordine edilir. Karşılıklı destek açısından bakıldığında şelale modelinin hiyerarşik yapısı takım üyelerinin arasında karşılıklı bir desteği zorlaştırırken çevik yöntemlerde kodun bütün takıma ait olması, günlük toplantılar, retrospektif incelemeler karşılıklı desteği ve dayanışmayı canlandırır. Şelale modelinde çaba, bireylerin kendilerine verilen görevleri tamamlamaları için harcanır. Çevik yöntemlerde ise çaba takıma aittir ve bireysel iş yoktur. Şelale modelindeki hiyerarşik yapı ve formel iletişim, takım içi uyumu olumsuz etkilerken çevik yöntemlerde genellikle aynı fiziksel alanda bulunan takım elemanları arasındaki etkileşim ile daha yüksek bir uyum yakalanır. [51] Bu sebeplerle ekip çalışması bakımından çevik yöntemler şelale modeline göre daha kaliteli bir yaklaşım ortaya koyar ve ekip çalışmasının önemli olduğu projeler için çevik yöntemler daha uygundur.

4.6. Ürün (Product)

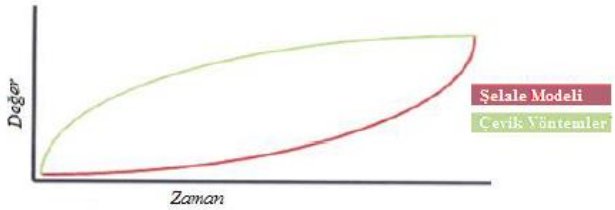
Nitelikler: Yazılımdan beklenen nitelikler, yazılımın uygulama alanına göre değişiklik gösterir. Örneğin bir banka sisteminden güvenli olması, iletişim ağlarının devamlılığının olması, interaktif bir oyunun hassas bir tepkisellik sunması beklenir. Ürün nitelikleri aşağıdaki şekilde genellenebilir:

- Sürdürülebilirlik*:* Yazılım müşterilerin değişen ihtiyaçlarına göre evrilebilecek şekilde üretilmelidir.
- Güvenilebilirlik ve Emniyet: Güvenilir yazılım, bir hata halinde fiziksel veya maddi hasar oluşturmamalıdır. Kötü niyetli kullanıcılar sisteme erişip zarar verememelidir.
- Verimlilik: Yazılım, işlemci ve RAM gibi sistem kaynaklarını boşa harcamamalıdır.
- Kabul Edilirlik: Yazılım, hitaben üretildiği kullanıcı tarafından kabul edilmelidir. Anlaşılır, kullanılabilir ve diğer sistemlerle uyumlu olmalıdır [2].

Bu niteliklere metodolojiler açısından bakılacak olursa, sürdürülebilirlik ve kabul edilirlik nitelikleri açısından çevik yöntemler şelale modeline göre daha uygundur. Şelale modeli değişimi kabul etmezken çevik yöntemlerde değişim hoş karşılanır. Çevik yöntemlerde yinelemeler sonunda yapılan ürün teslimleri ve alınan geri beslemeler ile müşterinin proje sürecinin tamamına katılımı, teslim safhasında kabul edilirlik açısından şelale modeline göre avantaj sağlar. Güvenilebilirlik ve emniyet ile verimlilik nitelikleri açısından ise şelale modeli ön plana çıkmaktadır. Şelale modelinde uygulanan katı süreçler ile test yöntemleri, çevik yöntemlerin test ve kalite yöntemlerine göre daha güvenilirdir. Ayrıca ürünün sık ve çabuk teslim edilmesi ve müşteri gereksinimlerinin işlevselliğine odaklanılması, çevik yöntemlerde güvenlik, verimlilik gibi konuların göz ardı edilmesine sebep olabilir.

Kalite Güvence ve Hata Toleransı: Çevik yöntemlerde eşli programlama, önce test yaklaşımı, gözden geçirme toplantıları, ürün değerlendirmeleri gibi tekniklerle sürekli inceleme ve ürünün geliştirme boyunca iyileştirilmesi sağlanmaya çalışılsa da, bu önlemlerin hata toleransı olmayan kritik sistemlerde istenen kalite için yeterli olmayacağı değerlendirilmektedir [36]. Bu sistemlerde arzu edilen kalitenin sağlanabilmesi için icra edilen analiz, geliştirme ve test faaliyetleri daha detaylı, resmi ve katı uygulamalar içermelidir [41]. Çevik yöntemlerin başlıca avantajları olan erken değer üretimi, sürekli geri besleme ve değişen şartlara adapte olabilme yeteneği, ilk ve en büyük önceliği hataya karşı sıfır tolerans olan bu sistemler için bir önem ifade etmez. Bu nedenle kalite güvence anlayışı açısından şelale modeli, çevik yöntemlere göre daha uygun bir yöntemdir.

Değer Üretimi: Gerek şelale modeli gerekse çevik yöntemler yazılımın üreteceği değere önem verir. Kazanılmış değer analizi, yazılım projelerinin önemli bir yönetim alanıdır. Çevik yöntemlerde müşterinin istediği yazılım değeri proje süreci boyunca öğrenilir ve üretilir. Şelale modelinde ise proje başlangıcında anlaşılması ve proje sonunda üretilmesi öngörülür [47]. Şelale modelinde proje sonunda ürün teslimi yapıldığı için değer üretimi proje sonunda gerçekleşir. Çevik yöntemlerde ise yinelemeler sonunda teslim edilen her çalışan ürün müşteri için bir fayda sağlar. Yazılımdan beklenen değerin iyi anlaşılması için, müşterinin isteklerini iyi ifade edebilmesi gerekir. Şelale modelinden çevik yöntemlere geçen bir şirketin bir proje yöneticisi, gereksinimlerin son kullanıcı tarafından değil de kullanıcı olmayan bir müdür tarafından tanımlandığı için yaptıkları işlerin % 80'inin kullanıcı için bir değer üretmediğini gördüklerini belirtmiştir [38]. Şelale modeli ve çevik yöntemlerin değer üretimi grafiği Şekil 10'da sunulmuştur.



Şekil 10. Değer Üretimi Karşılaştırması [48]
(Comparison of Providing Value)

Kullanıcı Deneyimi: Şelale modelinde kullanıcı deneyimi ve etkileşimi için detaylı bir tasarım ve geniş bir zaman ayrılır. Çevik yöntemlerde ise uzun vadeli plan ve detaylı tasarım yapılmaz ve kısa zamanlı yinelemeler, bütünsel bir yaklaşım gerektiren kullanıcı deneyimi ve etkileşimi için yapılacak çalışmalar açısından uygun değildir [43]. Bunun yanında çevik yöntemlerde teslimatlar sonrası alınan geri beslemelerle kullanıcı deneyimi iyileştirilebilir.

Dokümantasyon ve Modelleme: Üretilen yazılımın kullanılması öngörülen sürenin uzun ve kullanıcıların değişken olduğu durumlarda dokümantasyon bir başvuru kaynağı olarak önem arz edecektir. Geliştirme ekibinin dağınık olduğu veya bir altyüklenici kullanılan durumlarda da dokümantasyon önemli bir iletişim kanalı olarak kullanılabilir [2]. Dokümantasyon ve modelleme, müşteri için bir değer üretmediği durumlarda bile geliştiriciler için faydalı olabilir. Evrilen büyük sistemlerin geliştirilmesinde başvurulacak mevcut modelleme yoksa, geliştiriciler sistemi anlamak ve yapılacak değişimin etkisini belirleyebilmek için çalışan kodu analiz etmek zorunda kalırlar [36]. Dokümantasyon açısından uyguladıkları felsefe nedeniyle böyle durumlarda çevik yöntemlerden ziyade şelale modeli ile yazılım projesinin yönetilmesi daha faydalı olacaktır.

Teslimatlar: Jenerik yazılım üreterek pazara sunan firmalar, sürümden belli bir süre önce ürünün pazarlaması için çalışmalara başlarlar. Şelale modelinde proje sonunda tek bir teslimat yapılması, bu felsefeye uygundur fakat çevik yöntemlerdeki kısa süreli yinelemeler sonucu verilen ürün versiyonları ve gereksinim önceliklerinin belirlenmesindeki esnek yaklaşım, pazarlama departmanlarına hangi özellikteki ürünü ne zaman pazarlayacakları konusunda net bir görüş sağlayamaz [43]. Bu nedenle jenerik yazılım üreten firmalar için uzun vadeli planlamaları gereği şelale modeli daha uygun bir yaklaşımdır. Bunun yanında, pazar şartlarının ve iş süreçlerinin hızlı değiştiği, değişikliklere çabuk reaksiyon gösterilmesi gereken ortamlar ve sürekli değer üretimi beklenen projeler için, yinelemeler sonunda ürün tesliminin yapılması, çevik yöntemleri avantajlı kılar. Teslim edilen ürünlerden alınacak somut geri beslemeler sayesinde hatalar ayıklanır, düzeltmeler yapılır ve proje süreci daha sağlıklı ilerler.

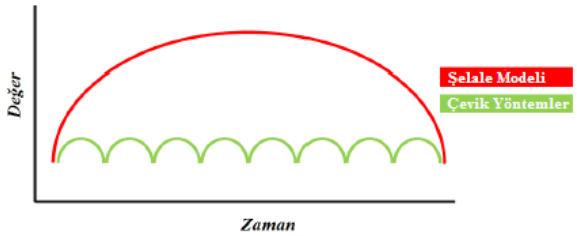
Yeniden Kullanılabilirlik: Çevik yöntemlerde basitlik prensibiyle hareket edildiği için, temel ihtiyacı karşılamak için çaba harcanır. Ürünün hiç gerçekleşmeyebilecek durumlar için genelleştirilmesine ve kapsamın genişletilmesine izin verilmez. Yeniden kullanılabilir bir ürün için bu şartların tanımlandığı dokümantasyon da gerekecektir ve bu da iş yükünü gereksiz yere arttıracaktır [36]. Bu nedenle üretilen ürünler daha spesifiktir. Dokümantasyon ve modelleme ihtiyaç duyulmadığı sürece detaylı olarak üretilmediği için ürünlerin yeniden kullanılabilirliği çok zayıftır. Genel maksatlı, kullanımı genişletilebilecek ya da farklı projelerde yararlanılabilecek yazılım için, dokümantasyon ve modelleme imkânı nedeniyle şelale modeli daha uygundur.

Kullanım Onayı: Üretilen yazılımın, kullanılacağı alan gereği bir otorite ya da düzenleyici kuruluş tarafından onaylanması gerekiyorsa (örneğin havacılık düzenlemeleri, adli düzenlemeler,) bu onaylama sırasında incelenecek dokümantasyonun detaylı olması gerekecektir ve söz konusu onay son ürüne verileceği için ara ürün teslimatı yapılmasının hiçbir önemi yoktur [2]. Bu tarz yazılımlar için şelale modeli daha uygun bir yaklaşımdır.

4.7. Gereksinimler (Requirements)

Belirsizlik: Şelale modeli, plan tabanlı ve önceden tanımlanan süreçler olduğu için belirsizlik toleransı yoktur. Çevik yöntemler ise planların değişeceği varsayımıyla önceden detaylı plan yapılmaması ve planlamadan ziyade ürün ortaya koyularak zamanın harcanması felsefesini benimsediği için belirsiz ortamlar açısından şelale modeline göre daha uygundur [41].

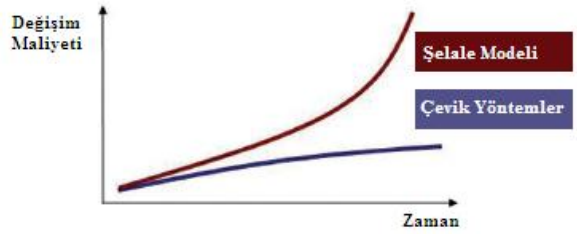
İhtiyaca Yabancılaşma: Bir projedeki kullanıcı tarafından belirtilen ihtiyaçlar sırasıyla gereksinimlere, tasarım modeline, yazılım koduna ve son olarak ihtiyacı karşıladığı değerlendirilen ürüne dönüşmektedir. Şelale modelinde projenin başındaki gereksinim analizi asıl ihtiyaç ile etkileşim halinde iken müteakip aşamalarda bir önceki aşamanın çıktısı ile işlem yapılır. Projenin tamamlanma süreci göz önüne alındığında gereksinim analizinden kabul dönemine kadar ihtiyaca yabancılaşma söz konusu olur. Çevik yöntemlerde ise her yineleme başlangıcında gereksinimler gözden geçirilir. Müşteri ise her yineleme sonucu teslim edilen ürünün ihtiyacını karşılayıp karşılamadığını görür. Bu nedenle ihtiyaca yabancılaşma etkisi bir yineleme süresi ve boyutu ile sınırlı kalmaktadır. [48] Şelale modeli ve çevik yöntemlerin ihtiyaca yabancılaşma grafiği Şekil 11'de sunulmuştur.



Şekil 11. İhtiyaca Yabancılaşma Karşılaştırması [48]

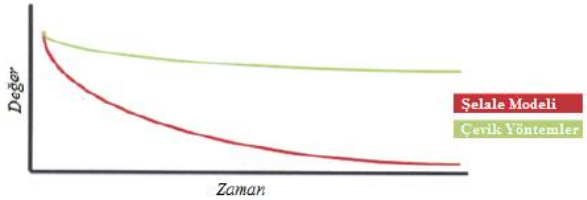
(Comparison of Divergence from Needs)

Değişime Açıklık: Çevik yöntemlerin ortaya çıkmasının en büyük sebeplerinden ve çevik yöntemlerin en büyük avantajlarından biri gereksinimlerdeki değişikliklerin proje boyunca kabul edilmesidir. Şelale modelinde gereksinim analizi safhası tamamlandıktan sonra projenin ilerleyen safhalarında gereksinimlerde yapılacak değişiklikler hem maliyeti arttıracağı hem de önceki safhalara dönülmesini gerektireceği için kabul edilmez. Çevik yöntemlerde değişim maliyetleri şelale modeline göre nispeten daha düşüktür. Şelale modeli ve çevik yöntemlerin değişim maliyeti karşılaştırması Şekil 12'de gösterilmiştir.



Şekil 12. Değişiklik Maliyeti Karşılaştırması [49]
(Comparison of Cost of Change)

Bu kapsamda, uzun şartname hazırlıkları, bütçe-ödenek ve ihale süreçleri ile proje süreleri göz önüne alındığında bazen şelale modeli ile üretilen yazılımlar yıllar önceki ihtiyaçlara cevap vermek durumunda kalmaktadır [12]. Şelale modeli ve çevik yöntemlerin değişime açıklık grafiği Şekil 13'te sunulmuştur.



Şekil 13. Değişime Açıklık Karşılaştırması [48]
(Comparison of Opennes to Change)

Anlaşılırlık, Uygulanabilirlik, Güvenilirlik: Geliştirme faaliyetine başlanmadan önce detaylı bir gereksinim tanımlama ve tasarım ihtiyacı var ise ayrılan zaman ve süreçler nedeniyle şelale modeli daha uygun bir yaklaşımdır [2]. Fakat bu safhaların başarıyla aşılabilmesi için proje başlangıcında bütün ihtiyaçların biliniyor olması ve bu ihtiyaçların müşteri tarafından doğru ve uygulanabilir bir şekilde ifade edilebilmesi gerekir. Şelale modelinde, gereksinimler tam anlaşılmadığında müşteriyi temsilen kimse bulunmadığı için geliştiriciler iş süreçleri ve gereksinimler konusunda varsayımlarda bulunmak zorunda kalırlar [40]. Geri dönüşler de kabul edilmediği için bu konulardaki aksaklıkların yaratacağı sıkıntılar proje boyunca hissedilecektir. Gereksinimlerin anlaşılırlığı, uygulanabilirliği ve güvenilirliği ile ilgili bir belirsizlik veya tereddüt varsa esnek yapısı nedeniyle çevik yöntemler daha uygun bir yaklaşım sunabilir

Gereksinim Yönetimi: Şelale modelinde detaylı hazırlanan gereksinimlerin tasarım ile birleşerek kodlanacak görevlerin oluşturulmasıyla gerek proje yöneticileri gerekse dokümantasyon vasıtasıyla iyi bir gereksinim yönetimi sağlanmış olur. Çevik yöntemlerde ise gereksinimlerin müşteriler ile birlikte belirlenip önceliklendirilmesi ve geliştirme takımının bu sürece dahil olması küçük projelerde bir sorun yaratmasa da gereksinimlerin birden fazla grup tarafından tanımlandığı ve geliştirme takımlarının hepsiyle irtibatının olmadığı büyük projelerde gereksinim yönetimi eksikliği yaşanabilir [43].

4.8. Kaynaklar (Resources)

Şelale modelinde çoğu zaman tercih sebebi olan daha detaylı analiz, dokümantasyon ve testlerin yapılabilmesi için daha çok bütçe, zaman ve işgücü gerekir [41]. Çevik yöntemler analiz, dokümantasyon ve test ihtiyaçlarının daha az olması sebebiyle kullanılan kaynaklar konusunda şelale modeline göre avantaj sağlar.

Zaman: Şelale modelinde safha bitimi için bütün işlerin bitmesi gerekir ve henüz bitirilmemiş işler yüzünden safha ilerlemediğinde işini bitiren personel zaman kaybeder. Proje elemanlarının hep beraber takım olarak çalışması kaynakların daha etkin kullanımını sağlar [44]. Bu nedenle şelale modelinde zaman israfı yaşanması mümkün iken çevik yöntemlerde kısa süreli yinelemeler sayesinde zaman en etkin şekilde kullanılır.

İş gücü: Yazılım geliştirme sürecinde yapılan faaliyetlerin aynı olduğu, bu faaliyetlerin sıralamasında ve takibinde farklılıklar olduğu göz önünde bulundurularak, şelale modeli ile çevik yöntemlerin, adam-saat işgücü açısından karşılaştırılması için belirleyici iki faktör mevcuttur. Bunlar şelale modelinde detaylı bir şekilde yapılan analiz ve gereksinim tanımlamaları ile çevik yöntemlerde yapılmış olan işlerde ortaya çıkan değişiklikler yüzünden tekrar yapılan işlerdir. Çevik yöntemlerde gereksinimlerin tanımlanması konusunda tasarruf edilen zaman, değişiklikler yüzünden yeniden yapılan işlere harcanan zamandan fazla olduğu takdirde, adam-saat karşılaştırılması açısından çevik yöntemler daha avantajlıdır. Değişiklik nedeniyle yapılmış olan işlere tekrar harcanan işgücü gereksinim tanımlamasında tasarruf edilen işgücünden fazla ise şelale modeli daha avantajlıdır [42].

4.9. Risk Yönetimi (Risk Management)

Risk yönetimi, risk faktörlerinin tanımlanması ve her risk faktörünün gerçekleşme ihtimali ile potansiyel etkisinin analizi, risk faktörlerinin önceliklendirilmesi, gerçekleşme ihtimallerin düşürülebilmesi için risk azaltma stratejilerinin belirlenmesi ve probleme dönüşen risk faktörlerinin olumsuz etkilerinin en aza indirgenmesi

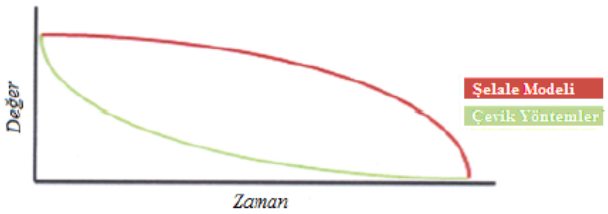
faaliyetlerini kapsar [6]. Tablo 7'de genel olarak risk türleri ve kısaca açıklamaları verilmiştir.

Söz konusu risk türleri, şelale modeli ve çevik yöntemler açısından değerlendirilecek olursa; zaman riskleri ve bütçe riskleri açısından şelale modelinde proje başlangıcında yapılan planlamaların detaylı ve gerçekçi bir şekilde yapılması önem arz etmektedir. Çevik yöntemlerde ise yinelemeli geliştirme anlayışı ve her yineleme sonunda yapılan teslimatlar bu risklerin izlenmesi ve yönetilmesini kolaylaştırmaktadır. Bunun yanında çevik yöntemlerin değişime açık yapısı, zaman ve bütçe risklerinin yönetimi açısından dezavantaj oluşturmaktadır. Şelale modeli ile çevik yöntemlerin yönetim anlayışındaki farklar nedeniyle yönetim risklerinin ele alınması da farklı olmalıdır. Şelale modelinde kontrol, otorite ve yönetim konuları, çevik yöntemlerde ise iletişim, organizasyon ve iş birliği konuları üzerinde durulmalıdır. Teknik riskler ve program risklerinin yönetimi açısından çevik yöntemlerin yineleme ve teslimatları izlenebilirliği kolaylaştırıp avantaj sağlarken, şelale modelinde kapsam konusunda değişikliğe kapalı olunması olası problemleri indirgemektedir. Çevik yöntemlerin personel bilgi ve tecrübesine dayalı yapısı nedeniyle personel risklerinin yönetimi konusu, şelale modeline göre daha fazla önem arz etmektedir.

Tablo 7. Risk Türleri ve Açıklamaları [54]
(Risk Types and Definitions)

Risk Türü	Açıklama
Zaman Riskleri	Projenin, yanlış görev ve malzeme paylaşımından dolayı beklenen süre içerisinde gerçekleşmesine veya tamamlanmasına engel olan risk çeşididir.
Bütçe (Maliyet) Riskleri	Gerçekçi olmayan bütçe tahminleri sonucu finansal sorunlara yol açan risklerdir. Bu risklerin gerçekleşmesi durumunda Tablolar değişmekte, maliyetler artmaktadır.
Yönetim Riskleri	Amaçların net olmayışı, planlama eksikliği, yönetim tecrübesi ve eğitim eksikliği, iletişim sorunları, örgütsel sorunlar, otorite eksikliği ve kontrol problemlerini kapsamaktadır.
Teknik Riskler	Genelde fonksiyonların yanlış olmasından kaynaklanır. Müşteri taleplerinin sürekli değişmesi, gelişmiş tekniklerin kullanılmaması ve geliştirilecek olan projenin zor faaliyetler içermesi gibi sebeplerden kaynaklanmaktadır.
Program Riskleri	Proje kapsamının dışına çıkan, kontrol dışı durumlardan veya önceliklerin sürekli değişmesinden doğan risklerdir.
Sözleşme ve Yasal Riskler	Değişen ihtiyaçları, pazar odaklı programları, sağlık ve güvenlik sorunları, hükümet düzenlemeleri ve ürün garantisi konularını içerir.
Personel Riskleri	Personel duraklamaları, deneyim ve eğitim sorunları, etik ve ahlak konularını, personel çatışmalarını ve verimlilik sorunlarını içermekedir.
Diğer Kaynaklı Riskler	Mevcut olmayan veya geç teslim edilen ekipman ve sarf malzemeleri, yetersiz aracı, yetersiz tesisleri, dağıtılan bölgeleri, bilgisayar kaynaklarının olmayışı ve yavaş tepki sürelerini kapsamaktadır.

Şelale modeli ve çevik yöntemlerin zaman içinde taşıdıkları risk grafiği Şekil 14'te sunulmuştur.



Şekil 14. Risk Karşılaştırması [48]
(Comparison of Risk)

Şelale modeli, proje başlangıcından itibaren hem üretim hem de kabul risklerini barındırmaktadır. Süreç içerisinde uygulanan önlemler ile risk bir miktar azalsa da kabul aşamasına kadar ana hatlarıyla devam etmektedir. Çevik yöntemlerde ise tamamlanan her yinelemede üretim riskleri, her teslimde ise müşteri katılımı sağlandığı için [14] kabul riskleri azalmaktadır [48].

5. SONUÇ(CONCLUSION)

Şelale modeli yazılım mühendisliğinin en eski ve temel modelidir. Günümüzde kamu kuruluşları ve büyük şirketler tarafından her türlü projenin yönetim standardı olarak kabul görmektedir [19]. Çevik yöntemler ise geleneksel yöntemlerin yetersiz kaldığı değerlendirilen konular için alternatif çözümler olarak ortaya çıkmıştır. Günümüzde gerek Türkiye gerekse dünya çapında yazılım proje yönetiminde daha çok çevik yöntemler tercih edilmektedir [30,31].

Çalışma kapsamında, çevik yöntemler ile şelale modelinin proje başarısına etki ettiği değerlendirilen faktörler açısından durumsal incelemeleri ve karşılaştırmaları yapılmıştır. Bu karşılaştırmalarda, 13 durumda çevik yöntemlerin; 11 durumda şelale modelinin avantajlı olduğu değerlendirilmiştir. 9 durumda ise her iki metodolojinin çeşitli avantajları mevcuttur. Genel karşılaştırma özeti Tablo 8'de sunulmuştur.

Şelale modeli ve çevik yöntemler güçlü ve zayıf yönleriyle uygunluk açısından proje bazında değerlendirilmesi gereken metodolojilerdir. Şelale modeli projenin geçmiş safhalarına bağımlı olarak yürütüldüğü için anlayış olarak geriye dönüktür ve bütün proje elemanlarının kendisi için tanımlanan işi yapmasının

beklendiği bir mühendislik düzenidir. Bu düzenden faydalanan şelale modeli daha çok, gereksinimlerin ve hedeflerin açık ve net olduğu, çözümlerin ve yapılacak işin bilindiği, gereksinimlerin değişmeyeceği, üretilecek yazılımın hata toleransının olmadığı ve fonksiyonel bütünlüğe sahip projeler için uygundur. Bu özellikleri nedeniyle en iyi kullanım alanı olarak savunma sistemleri ve gömülü sistemler verilebilir. Çevik yöntemler ise değişimi ve geri beslemeyi teşvik eden yapısı nedeniyle ileri dönük bir anlayışa sahiptir. Otonom takımların her işi kendi kendine yapması ve bu süreçte çapraz fonksiyonellikten ve deneycilikten faydalanmaları beklenir. Bu da düzenden çok bir yaratıcılık yaklaşımı gerektirir. Çevik yöntemler gereksinimlerin ve hedeflerin belirsiz ve bilinmez olduğu, değişikliğin sık, değer üretiminin kısa vadede arzu edildiği projeler için daha uygundur. Bu nedenle çevik yöntemler için en iyi kullanım alanı web tabanlı iş akış yönetimi sistemleridir.

İki yöntemin karışımı olarak ortaya çıkan metodolojiler de günümüzde popülerlik kazanmaya devam etmektedir. Örneğin birçok organizasyon, yazılım projesinin kodlama ve test safhalarının scrum metodu ile, planlama, tasarım ve teslimat gibi geri kalan safhaların da şelale modelinin prensipleri ile gerçekleştirildiği Water-Scrum-Fall metodolojisini uygulamaktadır [57].

Tablo 8. Karşılaştırma Özeti
(Summary of Comparisons)

Faktörler	**Şelale Modeli**	**Çevik Yöntemler**
Proje Yönetimi ve Yönetim Rolleri	✓	✓
Proje Görünürlüğü	×	✓
Proje Karmaşıklığı	✓	✓
Proje Büyüklüğü	×	✓
Altyüklenici Kullanımı	✓	×
Müşteri Katılımı	×	✓
Girişimci Organizasyon	×	✓
Makine Organizasyon	✓	×
Profesyonel Organizasyon	×	✓
Farklılaşmış Organizasyonlar	✓	✓
Yenilikçi Organizasyon	×	✓
Proje Elemanlarının Rolleri	✓	✓
Proje Elemanlarının Tecrübe ve Yetenekleri	✓	×
Takım büyüklüğü	✓	×
Proje Elemanlarının Alan Bilgisi	✓	×
İş birliği	×	✓
Proje Elemanlarının Yerleşimi	✓	×
Ekip Çalışması	×	✓
Yazılım Nitelikleri	✓	✓
Kalite Güvence ve Hata Toleransı	✓	×
Değer Üretimi	×	✓
Kullanıcı Deneyimi	✓	✓
Dokümantasyon ve Modelleme	✓	×
Teslimatlar	✓	✓
Yeniden Kullanılabilirlik	✓	×
Kullanım Onayı	✓	×
İhtiyaca Yabancılaşma	×	✓
Değişime Açıklık	×	✓
Gereksinimlerin Anlaşılırlığı, Uygulanabilirliği ve Güvenilirliği	×	✓
Gereksinim Yönetimi	✓	×
Zaman	×	✓
İş gücü	✓	✓
Risk Yönetimi	✓	✓

Günümüzde teknolojideki hızlı gelişim, bütün iş süreçlerini, pazar dinamiklerini ve bu doğrultuda yazılıma ihtiyaç duyan müşterilerin gereksinimlerini etkilemekte ve geçmişe nazaran daha değişken bir ortam yaratmaktadır. Bu değişken ortamda çevik yöntemler şelale modeline göre avantajlı olsa da bazı şartlarda şelale modelinin sağladığı güven ve düzeni sağlayamamaktadır. Bu nedenle başarılı bir yazılım proje yöneticisi hem şelale modeli hem de çevik yöntemlerin anlayışına, avantaj ve dezavantajlarına hâkim olmalı ve projenin ihtiyaçlarına göre uygun bir metodoloji benimsemelidir. Bu çalışma yazılım proje yönetiminde karşılaşılabilecek durumlar için şelale modeli ve çevik yöntemlerin avantaj ve dezavantajları açısından bir öneri niteliği taşımaktadır. Ne yazık ki her projeye uygun bir "gümüş kurşun" metodolojisi olmadığı için mevcut şartlara göre bir metodoloji seçimi yapılmalı ya da ihtiyaçlara göre metodolojilerin arzu edilen teknikleri birleştirilerek karma bir metodoloji oluşturulmalıdır.

KAYNAKLAR (REFERENCES)

[1] Büyük Türkçe Sözlük, http://www.tdk.gov.tr , 10.04.2017.

[2] I. Sommerville, **Software Engineering**, Pearson Education Inc., A.B.D., 2011.

[3] Project Management Institute-PMI, **A Guide to the Project Management Body of Knowledge (PMBOK® Guide)**, Project Management Institute-PMI, A.B.D., 2013.

[4] Proje, https://tr.wikipedia.org/wiki/Proje , 10.04.2017.

[5] Proje Yönetimi, https://tr.wikipedia.org/wiki/Proje_y%C3%B6netimi, 10.04.2017.

[6] IEEE Computer Society, **SWEBOK v3.0 - Guide to the Software Engineering Body of Knowledge**, P.Bourque, R.E.Fairley, IEEE Computer Society Products and Services, A.B.D., 2014.

[7] Project Management Institute-PMI, **Software Extension to the PMBOK® Guide Fifth Edition**, Project Management Institute-PMI, A.B.D., 2013.

[8] Standish Group International, **CHAOS Report**, A.B.D, 2015.

[9] N.M.A.Munassar, A.Govardhan, "A Comparison Between Five Models Of Software Engineering", International Journal of Computer Science Issues - IJCSI, Vol 7, Issue 5, 94-101,2010.

[10] NATO Science Committee, **Report on Software Engineering Conference**, P. Naur, B.Randell, Almanya, 1968.

[11] W.W.Royce, "Managing The Development of Large Software Systems", **The Proceedings of the WESCON**, San Francisco, A.B.D, 328-339, 1970.

[12] M.S.Palmquist, M.A.Lapham, S.Miller, T.Chick, I.Ozkaya, **Parallel Worlds: Agile and Waterfall Differences And Similarities**, Software Engineering Institute, Carneige Mellon University, A.B.D, 2013.

[13] A.Farrell, **Selecting a Software Development Methodology Based On Organizational Characteristics**, Yüksek Lisans Tezi, Athabasca University, School of Computing and Information Systems, 2007.

[14] P.Vohra, A.Singh, "A Contrast and Comparison of Modern Software Process Models", **International Conference on Advances in Management and Technology**, Patiala, Hindistan, 23-27,2013.

[15] A.Endres, "A Synopsis of Software Engineering History: The Industrial Perspective", **Position Papers for Dagstuhl Seminar 9635 on History of Software Engineering**, 1996, 20-24

[16] S.Balaji, M.S.Murugaiyan, "Waterfall vs. V-Model vs Agile: A Comparative Study on SDLC", International Journal of Information Technology and Business Management, Vol.2, No.1, 26-30, 2012

[17] M.A.Awad, **A Comparison Between Agile and Traditional Software Development Methodologies**, The University of Western Australia, Avustralya, 2005.

[18] The Standard Waterfall Model for Systems Development, http://web.archive.org/web/20040403211247/http://asd-www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm, 15.04.2017.

[19] Disciplined Agile Software Development: Definition, http://www.agilemodeling.com/essays/agileSoftwareDevelopment.htm, 20.04.2017.

[20] HP Enterprise, Agile is the new normal, A.B.D, 2017.

[21] AgileTurkey, **5th Annual Agility Report**, Türkiye, 2016.

[22] K.Beck, "Embracing Change with Extreme Programming", IEEE Computer, Vol. 32, 10 (10), 70-77, 1999.

[23] K.Schwaber, "Scrum Development Process", The proceedings of the OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag, Texas, A.B.D, 117-134, 1995.

[24] R.E.Jeffries, "eXtreme Testing: Why Aggressive Software Development Calls for Radical Testing Efforts", Software Testing & Quality Engineering, Vol. March/April, 23-26, 1999.

[25] A. Cockburn, **Surviving Object-Oriented Projects**, Addison Wesley, A.B.D., 1998.

[26] J.Stapleton, **DSDM: Business Focused Development**, Addison Wesley, İngiltere, 2003.

[27] J.A. Highsmith, **Adaptive Software Development: A Collaborative Approach to Managing Complex Systems**, Dorset House Publishing, A.B.D., 2000.

[28] P. Coad, J. De Luca, E. LeFebvre, **Java Modeling In Color With UML: Enterprise Components and Process**, Prentice Hall, A.B.D., 1999.

[29] History: The Agile Manifesto, http://agilemanifesto.org/history.html , 20.04.2017.

[30] Çevik Yazılım Geliştirme Manifestosu, http://agilemanifesto.org/iso/tr/manifesto.html, 20.04.2017.

[31] J.Fair, "Agile versus Waterfall:approach is right for my ERP Project?", **PMI® Global Congress 2012 EMEA** , Marsilya, Fransa,2012.

[32] VersionOne Inc., 11th Annual State of Agile Report, A.B.D., 2017.

[33] Forrester Research Inc., **The 2015 State of Agile Development**, A.B.D., 2015.

[34] F.Harleen, S.Chande, "A Systematic Study on Agile Software Development Methodologies and Practices", International Journal of Computer Science and Information Technologies, Vol. 5(3), 3626-3637, 2014.

[35] M.E.Lapham, R.Williams, C.Hammons, D.Burton, A.Schenker, **Considerations for Using Agile in DoD Acquisition**, Software Engineering Institute, Carneige Mellon University, A.B.D, 2013.

[36] D.Turk, R.France, B.Rumpe, "Assumptions Underlying Agile Software Processes", Journal of Database Management, Vol. 16, No.4, 62-87, 2005.

[37] D.Turk, R.France, B.Rumpe, "Limitations of Agile Software Processes", 3rd International Confrence on Extreme Programming and Flexible Processes in Software Engineering- XP 2002, Alghero, İtalya, 43-46, 2002.

[38] H.Wells, D.Dalcher, H.Smyth, "The adoption of agile management practices in a traditional Project environment: An IT/IS CaseStudy ", 48th Hawaii International Conference on System Sciences, Hawaii, A.B.D., 4446-4453, 2015.

[39] M.E.Sarıdoğan, **Yazılım Mühendisliği**, Papatya, İstanbul, Türkiye, 2004.

[40] G.v.Waardenburg, H.v.Vliet, "When agile meets the enterprise", Information and Software Technology, 55, 2154-2171, 2013.

[41] D.Edberg, P.Ivanova, W.Kuechler, "Methodology Mashups:An Exploration of Processes Used to Maintain Software", Journal of Management Information Systems, 28, 4, 271-304, 2012.

[42] S.Komai, H.Saidi, H.Nakanishi, "Man-Hour Comparison Between Two Methods of Agile and Waterfall in IT System Development", Proceedings of the 13th International Conference on Innovation and Management, Wuhan, Çin, 829-836, 2016.

[43] K.Dikert, M.Paasivaara, C.Lassenius, "Challenges and success factors for large-scale agile transformations: A systematic literatüre review", The Journal of Systems and Software, 119, 87-108, 2016.

[44] A.J.Sohi, M.Hertogh, M. Bosch-Rekveldt, R.Blom, "Does lean & agile project management help coping with project complexity?", **29th World Congress International Project Management Association**, Westin Playa Bonita, Panama, 252-259, 2015.

[45] R.Hoda, L.K.Murugesan, "Multi-level agile project management challenges: A self-organizing team perspective", The Journal of Systems and Software, 117, 245-257, 2016.

[46] R.Hoda, J.Noble, S.Marshall, "The impact of inadequate customer collaboration on self-organizing agile teams", Information and Software Technology, 53(5), 521-534, 2011.

[47] T.Dingsøyr, C.Lassenius, "Emerging themes in agile software development: Introduction to the special section on contionuous value delivery", Information and Software Technology, 77, 56-60, 2016.

[48] Y.Macit, E.Tüzün, "Uygulama Yaşam Döngüsü Yönetimi Karşılaştırmalı Süreç İncelemesi", **9uncu Ulusal Yazılım Mühendisliği Sempozyumu**, İzmir, Türkiye, 122-133, 2015.

[49] S.Canditt, D.Rauh,M.Wittmann, "Brückenschlag: Das V-Model XT mit Scrum inside", OBJEKTSpektrum, 5, 2010.

[50] M.Hoegl, H.G.Gemuenden, "Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence", Organization Science, 12 (4), 435-449, 2001.

[51] Y.Lindsjørn, D.I.K.Sjøberg, T.Dingsøyr, G.R.Bergersen, "Teamwork quality and project success in software development: A survey of agile deveopment teams", The Journal of Systems and Software, 122, 274-286, 2016.

[52] H.Mintzberg, **Mintzberg on Management**, The Free Press, A.B.D., 1989

[53] M.Bulu, Nasil Bir Organizasyon Yapisina Ihtiyaciniz Var?, http://www.competitiveturkey.org/trial/melih_bulu_yazi/NasilBirOrganizasyonYapisinaihtiyacinizVar.pdf, 01.05.2017.

[54] M.H.Calp, M.A.Akcayol, "Yazılım Projelerinde Karşılaşılan Risk Faktörleri ve Risk Yönetim Süreci", Marmara Fen Bilimleri Dergisi,1, 1-13, 2015.

[55] A.Vaidya, "Does DAD Know Best, Is it Better to do LeSS or Just be SAFe? Adapting Scaling Agile Practices into the Enterprise", **PNSQC 2014 Proceedings**, Oregon, A.B.D., 21-38, 2014.

[56] K.Katz, "In Defense of Waterfall: Deconstructing the Agile Manifesto", **Better Software Confrence & Expo**, Las Vegas, A.B.D., 2009.

[57] D.West, M.Gilpin, T.Grant, A.Anderson, "Water-Scrum-fall is the reality of agile for most organizations today", 2011.