Bu makaleye atıfta bulunmak için/To cite this article:
KESKİNKILIÇ, M, KAHVECİ, F. (2019). Yazılım Mühendisliğinde Çevik Yöntemler Üzerine Kavramsal Bir İnceleme ve Sınıflandırma. Atatürk Üniversitesi Sosyal Bilimler Enstitüsü Dergisi, 23 (3) , 1067-1091.

Yazılım Mühendisliğinde Çevik Yöntemler Üzerine Kavramsal Bir İnceleme ve Sınıflandırma

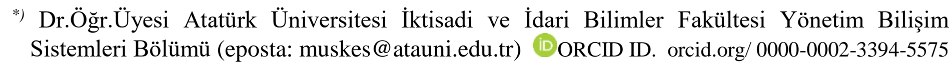
Mustafa KESKİNKILIÇ [*]
Ferhat KAHVECİ [**]

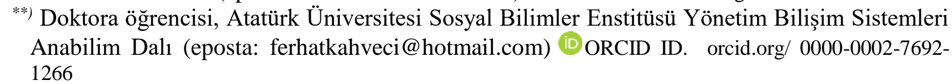
Öz: *Yönetim bilişim sistemleri disiplinini oluşturan alanlardan biri olan yazılım hızla gelişen bir endüstri alanıdır. Bu alanın ihtiyaçlarına cevap verebilmek için yazılım mühendisliğinde tanımlı çeşitli yöntembilimlerin yanı sıra çevik yöntembilimler de kullanılmaya başlanmıştır. Birbirine sıkı sıkıya bağlı yazılım, süreç ve proje kavramlarıyla ilgili çevik yöntembilimler başlığı altında birçok yöntem ve uygulama geliştirilmiştir. Bunların her birisinin kendine özgü yapıları olması yanında birçok ortak noktaları da vardır. Kavram karışıklıklarını engellemek amacıyla bu ortak noktaları vurgulayan Çevik Yazılım Geliştirme Manifestosu ilan edilmiştir. Bu çalışmada çevik yazılım geliştirme yaklaşımına dayalı uygulama ve süreçlerde öne çıkan bu ortak özellikler birer sınıflandırma kategorisi olarak tanımlanmıştır. Çevik yazılım geliştirme yaklaşımında, süreçler üzerinde analiz, yönetim, iyileştirme, gerçekleştirme yeniden tasarım ve değerlendirme gibi bazı faaliyetler öne çıkmaktadır. Çevik yöntembilimler başlığı altında tanımlı metodoloji, yöntem ve uygulamaların sınıflandırılması bu faaliyetlere göre yapılmıştır. Böylece hangi çevik metodoloji, yöntem ve uygulamanın hangi çevik faaliyet alanında daha etkin olduğu belirlenmiştir. Sınıflandırma çalışmasının okuyucuya daha yararlı olabilmesi için, yazılım proje yönetiminde kullanılan bazı kavram ve uygulamaların birbirleriyle ilgi ve farklılıkları kavramsal bir çerçevede ortaya konmaya çalışılmıştır.*

Anahtar Kelimeler: *Yönetim Bilişim Sistemleri, Yazılım Mühendisliği, Yazılım Proje Yönetimi, Yazılım Süreçleri, Çevik Yöntemler*

A Conceptual Study and Classification on Agile Methodologies at Software Engineering

Abstract: *Software, which is one of the fields of discipline of management information systems, is a rapidly developing industry field. In order to meet the needs of this field, a variety of methodologies as well as agile methodologies have been used in software engineering. Many methods and applications have been developed under the title of agile methodologies related to software, process and project concepts. Each of these has its own specific structures as well as many common points. Agile Software Development Manifest, which emphasizes these common points, has been declared to prevent concept confusion. In this study, these common features that are prominent in applications and processes based on agile software development approach are defined as classification categories. Some activities such as analysis, management, improvement, implementation, reconfiguring and evaluation in agile software development approach are prominent on the processes. Classification of*


[*] Dr.Öğr.Üyesi Atatürk Üniversitesi İktisadi ve İdari Bilimler Fakültesi Yönetim Bilişim Sistemleri Bölümü (eposta: muskes@atauni.edu.tr) ORCID ID. orcid.org/ 0000-0002-3394-5575


[**] Doktora öğrencisi, Atatürk Üniversitesi Sosyal Bilimler Enstitüsü Yönetim Bilişim Sistemleri Anabilim Dalı (eposta: ferhatkahveci@hotmail.com) ORCID ID. orcid.org/ 0000-0002-7692-1266

defined methodologies, methods and applications under title of agile methodologies is based on these activities. Thus, which agile methodology, method and application in which agile activity area is determined more effective. In order to make the classification work more useful to the reader, it has been tried to reveal the interests and differences of some concepts and applications used in software project management within a conceptual framework.

Keywords: *Management Information Systems, Software engineering, Software Project Management, Software Processes, Agile methodologies*

Makale Geliş Tarihi: *26.08.2019*

Makale Kabul Tarihi: *18.09.2019*

I. Giriş

Yazılımlara çeşitli prosedürler ve otomasyon gibi önemli teknik özellikler katılarak bilgi sistemleri oluşturulmaktadır. Yazılım mühendisliği yazılım konusuna geliştirilme, işletilme ve bakım konularında sistematik, disiplinli ve ölçülebilir bir yaklaşım uygulamaktadır. Yazılım geliştirme sadece bir programlama veya kodlama işi değildir. Mühendisin, yazılım ürününün iyi çalışması için gereksinimleri tam olarak tanımlayıp bu gereksinimleri karşılayacak bir tasarım üretmesini; böylece projenin zamanında ve bütçeyi aşmadan planlanmasını ve tamamlanıp teslim edilmesini gerekli kılar. (O'Regan, 2017: 4-5).

Yazılım mühendisliğinin yazılım projesi açısından hedefi; maliyet, süre, personel ve ürün kalitesi açısından olabilecek en iyi düzeyde projeyi tamamlamaktır. Yazılım mühendisliğinin işletme bağlamında hedefi; iş süreçlerini bilgisayar ortamında otomatikleştirilmiş bir takip süreci içinde gerçekleştirebilme ihtiyacını en verimli, güvenli ve fonksiyonel şekilde gidermektir. Yazılım mühendisliğinin yazılım geliştirme açısından amacı ise; sistematik bir yaklaşımla ihtiyaca uygun yazılım geliştirme yöntemi ve süreç modelinin seçiminde yol gösterip kaliteli bir yazılımın üretilmesini sağlamaktır. Sonra da yazılımı uygun biçimde işletmeyle bütünleştirerek kullanımda ortaya çıkacak bakım ihtiyaçlarını planlamak ve bu ihtiyaçları karşılamaktır.

Bu hedefler açısından bakınca yazılım mühendisliği karşımıza bir yöntembilim (metodoloji) olarak çıkmaktadır. Belli bir işi yapmak üzere birbiriyle bütünleşik olarak kullanılan yöntem, araç ve teknikler kümesine yöntembilim denmektedir. Yöntembilimlerin amacı; işlerin düşük maliyetle, zamanında ve başarıyla yapılmasını sağlamaktır. Uygun bir yöntembilim kullanmadan özellikle bir yazılım projesinin başarılı olması mümkün değildir. Bilişim sistem ve teknolojilerine ait bir proje yürütebilmek için yaygın kullanım alanına sahip özellikle yazılım geliştirme süreci yaşam çevrimini tanımlayan birçok yöntembilim ve bunların altında tanımlı yöntem, süreç, model, mimari, araç ve teknik anlamında uygulamalar vardır ki zaman içinde bunlar birleşerek birer uluslararası standart haline gelmişlerdir (Sarıdoğan, 2004:397). Bu yöntembilimlerden biri de çevik yöntembilimlerdir. Son zamanlarda yazılım geliştiriciler arasında çevik adı altındaki hafif yöntembilimlerin popülerliğinde büyük bir artış olmuştur (O'Regan, 2017: 12).

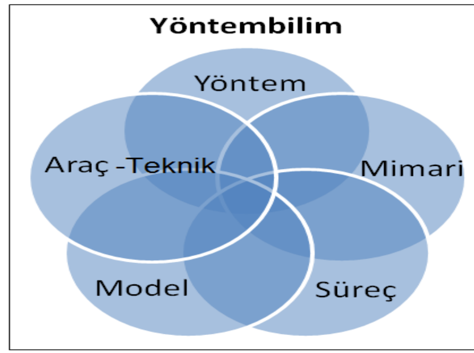
Bilgi sistemleri kurumsal yazılımlardır ve bunları geliştirme işinin profesyonelce yapılması gerekir. Bu profesyonellik yazılım mühendisliği ilkelerine sıkı sıkıya bağlı yazılım projeleri çerçevesinde iş yapmayı gerekli kılar. Yazılım mühendisliğinin içerdiği konular arasında yazılım proje yönetimi de bulunmaktadır. Bu yüzden bilgi sistemleri gibi profesyonel yazılımlar projelendirilerek geliştirilirken hem yazılım mühendisliği kavramları hem de proje yönetimi kavramları çoğu zaman bir arada kullanılmaktadır. Yazılım mühendisliği yöntembilimi yol gösterici bir ilkeler kılavuzu ve yapılacak işlerle ilgili bir yol haritası olarak değişik yöntem ve uygulamalar önerirken, proje yönetimi yöntembilimleri de benzer adı taşıyan kavramları içerebilmektedir. Yazılım mühendisliğinde, yazılım proje yönetiminde ve çevik yöntemler içerisinde karmaşık gibi görünen birbiriyle ilgili, benzer, hatta aynı adı taşıyan fakat farklı işler gören kavram, yöntem ve uygulamalar bulunmaktadır (TBA, 2006: 970). Bu yüzden yeni başlayanlar aynı adı taşıyan yazılım mühendisliğine ait kavramlar ile proje yönetimi kavramlarını birbiriyle karıştırmaktadırlar. Örneğin çevik (agile) ve scurum hem yazılım geliştirme işinde hem de proje yönetimi işinde ayrı ayrı birer yöntembilimdirler. Ayrıca süreç geliştirme ve yönetme için de birer araç içermektedirler. Profesyonel yazılım geliştirme işi yazılım, süreç ve proje kavramlarının her üçü ile de yakından ilgili olduğundan çevik, scurum ve diğer yöntembilimler her üç kavramla ilgili yöntem ve araçları içerecek şekilde tasarlanmışlardır. Bahsi geçen kavramlar arasındaki ilginin, benzerlik ya da farklılıkların durum ve derecesini belirtmek; bunları birbirlerine göre konumlarını ve öne çıkan özelliklerini ortaya koyacak şekilde tanımlamak; ayrıntıları inceleyip bir sınıflandırmada bulunmak ve konuya yeni başlayanlar ya da kafası karışık olanlar için kavramları yerli yerine oturtmaya çalışmak faydalı olacaktır. Çünkü Yazılım mühendisliği, yazılım üretimindeki karmaşıklıkları gidermeyi hedefler (Arifoğlu & Doğru, 2001: 9).

II. Kavramsal Çerçeve

A. Yazılım Mühendisliğinin Temel Kavramları

Yazılım mühendisliği söz konusu olduğunda; sırasıyla verilecek olursa metodoloji, metot, süreç modeli, yaşam döngüsü, mimari, araç-teknik, uygulama, analiz, tasarım, gerçekleştirme, paradigma, yaklaşım, platform, geliştirme, kalite, metrik, test, değerlendirme, geçerlileme, doğrulama, yerleştirme, entegrasyon, bakım-onarım, gibi kavramlara sıkça rastlanmaktadır. Kavramların profesyonelce kullanıldığı bir projede özellikle uygulamaya yöneldikçe; proje gelişimi ve ona bağlı olarak yazılım geliştirme çabaları izlenmeye başlandıkça; bu kavramların sistematik bir biçimde yazılım mühendisliğinin önerdiği yöntembilim başlığı altında girift bir şekilde yerli yerine yerleşerek daha anlamlı bir hale geldiği görülmektedir. Alan hakimiyeti olmadan yazılan literatürde, özellikle metodoloji teriminin yanlış kullanımı nedeniyle, kullanılan teknikler ve araçlar ile bunların nasıl konuşlandırılacağını belirleyen ilkeler arasındaki önemli kavramsal farklılıkların algılanması zorlaşmaktadır (McGregor & Murnane, 2010: 420). Bunun için başlangıçta temel kavramların genel kabul görmüş literatüre göre tanımlarının verilmesinde yarar vardır.

1. Yöntembilim (Metodoloji)-Yöntem (Metot): Yöntembilim mantığa, gerçekliğe ve bilgiye dayalı tüm araştırma ve uygulamaları kapsayan bir çerçevedir. Bir bilimsel veya teknik çalışmada izlenecek yol ve ilkelere yöntembilim dendiği gibi bu çalışmalarda kullanılan yöntem, teknik ve araçların toplamına ve nasıl konuşlanacaklarına dair yol haritasına da yöntembilim denmektedir. Ayrıca amaca ulaşmak için ne tür yöntem, araç ve teknikler kullanılması gerektiği konusunda geliştirilen kavramsal sisteme de yöntembilim denmektedir (Demir ve Acar. 1992: 387). Şekil 1'de yazılım mühendisliği yöntembiliminin içeriği ve bu içeriğin birbirleriyle ilişkileri kabaca gösterilmiştir.



Şekil 1: Yazılım mühendisliği yöntembiliminin içerdiği diğer kavramlar

Yöntem ise, bilimsel bir araştırmayı veya teknik bir çalışmayı gerçekleştirmek için gerekli teknik ve prosedürler olarak tanımlanmaktadır. Bu teknik ve prosedürler: örnekleme, veri toplama, analiz, tasarım, yorumlama, raporlama gibi çalışmalardan oluşmaktadır. Bundan başka amaç, araştırma sorusu, teori, kavramsal çerçeve, sınıflama, model gibi çalışmaların hepsi yöntembilim tarafından belirlenmektedir. (McGregor & Murnane, 2010: 420).

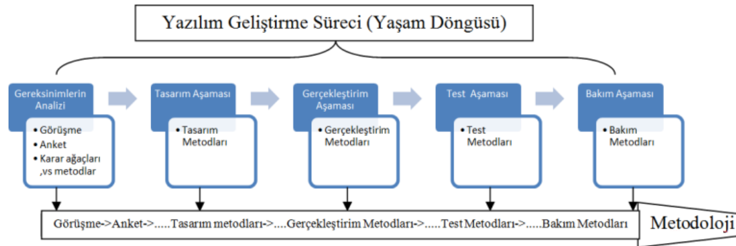
2. Yazılım Geliştirme Süreci-Yaşam Döngüsü-Süreç Modeli: Yazılım geliştirme işi bir süreçten oluşmaktadır. Bu süreç, bir yazılımın tasarım, üretim ve kullanım dahil geçirdiği tüm aşamaların toplamıdır.

1970'lerde çıkan ve yazılımın bir mühendislik etkinliği olarak uygulanmasında rol oynayan yazılım geliştirme süreçleri: kodlama ve geliştirme öncesi planlama ve analiz aşamasındaki yaklaşımları bir yönteme bağlamış; kodlama ve sonrası işlemlerinin bazı ilke ve standartlarla yürütülerek organize edilmesini sağlayarak yazılım geliştirme işine önemli katkıda bulunmuştur.

Geliştirme sürecini ilk olarak bir yaşam döngüsü şeklinde ifade eden Winn Royce olmuş ve önerdiği modele 'Şelale' (Waterfall) adı verilmiştir. Daha önce bir metodolojik yaklaşım eksikliği bulunan yazılım mühendisliği için şelale modeli tek yol olarak kabul görmüş ve çoğu zaman standart bir model olarak dayatılmış ve ilk olmanın verdiği kaçınılmaz hataları yüzünden çok eleştirilmiştir (Yazılım Geliştirme Süreci, 2018).

Yazılım geliştirilirken izlenen süreçte, işlemler çoğunlukla tekrarlı bir yol izlediğinden bir yaşam döngüsü oluşmaktadır. Bu yüzden yazılım geliştirme sürecine, yazılım yaşam döngüsü adı verilmektedir.

Yaşam döngüsü kavramı tüm yazılım projelerinde kullanılmaktadır (Acuña & Ferré, 2001: 237). Yaşam döngüsünün aşamaları: gereksinimlerin analizi, sistemin tasarımı, geliştirme ve test dahil gerçekleştirme, organizasyona yerleştirme ve teslimat sonrası bakımdır (Mujumdar, Masiwal & Chawan, 2012: 2015). Bunların her birine yazılım yaşam döngüsüne ait "Çekirdek Süreçler" denmektedir (Arifoğlu & Doğru, 2001: 17).



Şekil 2: Yazılım geliştirme sürecindeki yaşam döngüsü, metotlar ve metodoloji

Şekil 2'de yazılım geliştirme süreci ve bu sürece ait yaşam döngüsü yani yazılım süreç modeli ifade edilmiştir. Buradaki aşamaların farklı isimlendirilmesi veya farklı dizilimi ile yazılım geliştirme süreci içinde kullanılabilecek farklı "yazılım yaşam döngüleri" yani farklı "yazılım süreç modelleri" ortaya çıkar (Mahanti, Neogi & Bhattacherjee, 2012: 1254).

Yazılım geliştirme süreci dolayısıyla yazılım yaşam döngüsü veya diğer adıyla yazılım süreç modeli o kadar sıkı sıkıya ilişkilidirler ki neredeyse üç kavram da aynı şeyi ifade etmektedirler. Bir sürece ilişkin işlevleri yerine getirmek amacıyla kullanılan belirtim yöntemleri, akış şemaları, karar tabloları, veri sözlüğü gibi tanımlamalar yöntemlerdir (Arifoğlu & Doğru, 2001: 17). Bir süreç modeli ve belirtim yöntemleri birlikte ele alındığında yazılım geliştirme yaşam döngüsü boyunca kullanabilecek birbiriyle uyumlu metotları içeren "yöntembilim" meydana gelmektedir.

En sıkıntılı konu yazılım geliştirmenin sadece kodlama işi olarak düşünülmesidir. Halbuki yazılım geliştirme kodlamadan ibaret değildir (Camoğlu, 2008: 3). Yazılım geliştirme karmaşık ve çok boyutlu bir süreçtir ve bir yaşam döngüsüne sahiptir. Bu süreç veya döngünün sağlıklı işleyebilmesi için uygun bir yöntembilim kapsamında hem müşteriye uygun bir yöntemi hem de yazılım projesine uygun bir süreç modelini kullanmak gerekmektedir. Hatta yazılım süreçlerini geliştirecek ve yönetecek süreç geliştirme ve yönetim araçlarına ve yazılım kalitesini sağlamak için süreçleri iyileştirecek bir süreç iyileştirme modeline (CMMI, SPICE, AQAP vb.) ihtiyaç vardır.

Zaman içinde şelaleden başka diğer yazılım yaşam döngüleri (süreç modelleri) ortaya çıkmıştır. Günümüzde yazılım projesine ya da kuruluşa özel yaşam döngüleri veya yeni isimlendirme ile 'Süreç Modelleri' geliştirmek yaygın bir yöntem olmuştur. Doğrusal, Yinelemeli, Tahminsel, Çevik, RAD, RUP gibi süreç modelleri vardır.

Yazılım süreç modeli, bir diğer deyişle yazılım yaşam döngüsü, yazılım süreçlerinin nasıl yapılandırılacağı ve yönetileceği ile ilgili izlenecek yol haritasıdır. Bir yazılım projesinde kullanılacak yazılım süreç modelinin, sonradan da izlenebilecek kendine özgü adımları olduğu için, proje aşamasından sonra ortaya çıkabilecek hata veya eksiklikler, yazılım geliştiricilerden bağımsız olarak değerlendirilebilecektir. Bu yüzden bir yazılım projesinde kullanılacak süreç modeli, geliştirme yöntemi araç ve teknikleri yazılım geliştiricilerce kapsamlı bir şekilde bilinmelidir (Keskinkılıç & Özmen, 2018: 1).

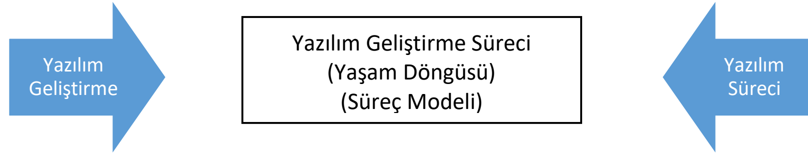
3. *Yazılım Geliştirme Yöntemi- Yazılım Geliştirme Yöntembilimi:* Yazılım geliştirme yöntemi daha çok yazılım ihtiyacı olan işletmenin yapısının ve kaynak durumunun dikkate alınarak ihtiyacının giderilmesiyle ilgilidir. Yani işletmenin ihtiyaç duyduğu yazılımı temin konusunda işletmenin para, zaman, personel, örgütsel yapı gibi kaynaklara ne oranda sahip olduğuna bakılarak yazılımın edinme ya da geliştirilme süreci planlanır. Bu faaliyetlerin içeriği işletmenin kaynak ve kapasite durumuna sıkı sıkıya bağlıdır. Bu bağlamda yazılım edinme veya geliştirme yöntemi olarak karşımıza genelde beş yöntem çıkmaktadır. Klasik Yaşam Döngüsü (Classic Waterfall), Prototipleme (Prototyping), Dış Kaynak Kullanımı (Outsourcing), Son Kullanıcı Geliştirmesi (End User Development), Paket Yazılım Kullanımı (Using Software Package). Her yazılım yönteminin birbirine göre üstün ve zayıf yönleri vardır.

Yazılım geliştirme yöntembilimi; yazılım geliştirirken kullanılacak yöntem, süreç, mimari, araç ve tekniklerin nasıl konuşlanacağını ve tüm bu faktörlerin nasıl bir ilişki içerisinde olacağının yol haritasını çizer. Geleneksel yazılım yöntembilimlerin dışındaki "Çevik Yöntemler" günümüzde dikkati en çok çeken yöntembilimlerdir.

4. Yazılım Mimarisi: Mimari, yazılımın üzerinde geliştirildiği ve çalıştırıldığı platformdur. Diğer bir deyişle uygulamanın çatısıdır (Arifoğlu & Doğru, 2001: 257).

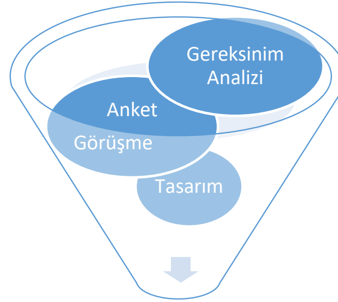
Yazılım mimarisi; Mimari Tanımlama Dili (ADL-Architecture Description Language), mimari bakış açısı ve mimari çerçeve ile şekillenmekte ve mimari modelin sunumu için gerekli prensipleri ve uygulamaları içermektedir. Yazılım mimarisi, yazılım geliştirirken özellikle işlevsel birimlere ve sistemin genel yapısına odaklanır (Reddy, Naidu & Govindarajulu, 2006: 36). Yazılım mimarisi için kullanılan belli başlı kavramlar vardır (Reddy ve ark., 2006: 33). Bu nedenle yazılımın mühendisliğinin diğer konularında karşılaşılan karışıklık yazılım mimarisinde pek görülmez. Başlıca yazılım mimarileri; bileşen, sentez, uzay, etmen tabanlı mimariler; nesneye, ilgiye yönelik mimariler; servis, ajan odaklı mimariler; model, olay güdümlü mimarilerdir. Bunlardan başka istemci-sunucu mimarisi, veri merkezli mimari, dağıtık mimari, iki katmanlı mimari, üç katmanlı mimari, kablosuz uygulama (WAP) mimarisi gibi çeşitli mimariler de bulunmaktadır (Oussalah, 2014, Arifoğlu, 2001: 257-275). Her bir mimari çatıya uygun biçimde yazılım geliştirmek için aynı adı taşıyan yazılım yöntembilimleri de vardır. Buradan şu anlaşılmalıdır ki; yazılım mimarisi kavramı da yazılım yöntembilimi kavramıyla sıkı sıkıya ve sistematik bir biçimde ilişkilidir. Ayrıca yazılım mimarileriyle ilgili internette yapılacak bir aramada bu ad altında okutulan bazı derslerin içeriğinde maalesef sadece yazılım yöntembilimleri ve yaşam döngülerinin ele alındığı, mimarilerden hiç bahsedilmediği görülmektedir. Yani yöntembilim veya yaşam döngüleri mimari konusuyla karıştırılmaktadır.

5. Kavramların Değerlendirilmesi-Araçlar ve Teknikler: Yukarıda verilen kavramların tanımları ışığında bakıldığında "yazılım geliştirme" ve "yazılım süreci" kavramları iç içe geçerek; "yazılım geliştirme süreci"ni yani "yaşam döngüsü"nü, bir diğer deyişle "süreç modeli"ni göstermektedirler. Şekil 3'te bu anlatılmaktadır.



Şekil 3. Yazılım Geliştirmede Süreç, Döngü ve Model Kavramları

Şu kadar var ki yaşam döngüsünün şeklini ve akışını süreç modeli belirlemektedir. Süreç modeline yazılım mühendisliği paradigması veya stratejisi de denmektedir (Pressman, 2001: 26). "Yazılım geliştirme yöntemi" çerçevesinde ise yazılım geliştirme sürecinde ve belirtim yöntemlerinde kullanılan bir takım "araç ve teknikler" söz konusudur. Şekil 4`te görüleceği gibi gereksinimlerin analizinde yapılan görüşmeler ve anketler "yöntem" içinde kullanılan birer araç ve teknik olarak tanımlanırlar.



Şekil 4. Yazılım Geliştirme Yönteminde Kullanılan Bazı Araç ve Teknikler

B. Çevik Yazılım Geliştirme

Geleneksel olarak kullanılan süreç modellerinden şelale, spiral, artırımsal, vb., ağır süreçler olarak nitelendirilmektedirler. Bu yaklaşımlarda birçok problem bulunmaktadır. Bunlar; zayıf iletişim, tahmini plan, belirsizliğe ve değişikliğe karşı yetersizlik, son kullanıcıyı hesaba katmama ve projenin sonuna kadar yüksek riskli görevler alınmasının projenin kapsam, zaman ve bütçesinde problemlere yol açmasıdır. Karşılaşılacak birçok problemin doğası gereği başlangıçta öngörülebilmesi zordur. Bu yüzden klasik seri üretim metotları bu koşullarda başarılı çözümlerden uzaktır ve hızla gelişen endüstrinin ihtiyaçlarına ve zamana karşı yarış sorununa cevap vermek için yazılım geliştirmede daha esnek metotlara ihtiyaç duyulmaktadır (Awad, 2005; Aybek, 2010; Ranniko, 2011: 3). Bu ihtiyaca binaen çevik (agile) paradigma ortaya çıkmıştır. Çevik paradigma: bir

proje içerisinde yazılım geliştirme faaliyetlerinin iteratif ve artırımsal bir yaşam döngüsü içinde yapılmasını, her döngü sonunda değer üretilmesini, paydaşlar ve son kullanıcıların bu döngüde söz sahibi olmasını gerekli gören bir yaklaşımdır. Çevik yöntembilimler çerçevesi içerisinde çevik yöntemler, çevik mimari ve çevik proje yönetimi birlikte barınmaktadır.

Yapılan araştırmalarda yazılım projeleri büyüdükçe karmaşıklık artmakta ve projelerin başarı oranları azalmaktadır. Diğer yandan projeleri parçalara ayıran tekrarlanan yazılım geliştirme yöntembilimleri, projelerin karmaşıklık ve riskini azaltmakta, verimliliği ve başarı oranlarını artırmaktadır (Aybek, 2010: 3). 1990'lı yıllarda geleneksel doküman merkezli ağır ve problemli yazılım geliştirme süreçlerine alternatif olarak çevik yazılım geliştirme süreçleri oluşturulmaya başlanmıştır (Sliger & Broderick, 2008: 10). Çevik yazılım geliştirme göreceli olarak yeni olsa da geçmişi benzer özellikler taşıyan "İteratif ve Artırımsal Geliştirme" (Iterative and Incremental Development-IID) kavramıyla 1930'lara kadar uzanmaktadır. NASA 1960 ve IBM 1970 yıllarında IID'yi projelerinde kullanmışlardır (Larman, 2003: 129). Çevik ile hedeflenen, süreçleri etkin kullanarak hızlandırmak ve gerektiğinde bunları dokümante etmektir (Gürcan, 2013: 1). Başta Uç Programlama (XP: Extreme Programming) ve Scrum olmak üzere Crystal, Dinamik Sistem Geliştirme Yöntemi (DSDM: Dynamic Systems Development Method), Lean, Kanban, Özellik Güdümlü Geliştirme (FDD: Feature Driven Development) ve Adaptif Yazılım Geliştirme (ASD: Adaptive Software Development) en çok kullanılan çevik yazılım geliştirme yöntembilimleridirler (Versionone, 2016). 2001 yılında çevik yazılım geliştiriciler, ortak bir paydada birleşmek amacıyla "Çevik Yazılım Geliştirme Manifestosu"nu yayımlamışlardır. Bu manifestoda yer alan esaslar şunlardır (Agilealliance, 2019a): Çevik yazılım geliştirme, çeşitli iteratif ve artırımsal yazılım geliştirme yöntembilimlerini kapsayan bir terimdir. Çevik yazılım geliştirmede; bireyler ve aralarındaki etkileşim, süreç ve kullanılan araçlardan daha önemlidir. Yazılımın çalışması ayrıntılı bir şekilde dokümantasyon yapmaktan daha önemlidir. Müşterinin yapacağı katkılar sözleşme ve anlaşmalardan daha önemlidir. Değişikliklere cevap verebilmek düz bir planı izlemekten daha önemlidir.

Çevik manifestoda on iki prensip üzerinde durulmaktadır (Agilealliance, 2019b): 1- Müşteri için önem arz eden yazılımın en erken ve sürekli teslimini sağlayarak müşteri memnuniyetini elde etmek ilk önceliktir. 2- Gecikmeler olsa dahi değişikliklere açık olmak. Çevik süreç bu değişikliklerle müşterinin avantajlı şekilde rekabet etmesini sağlar. 3- Kısa zamanda (2-6 hafta arası) çalışan yazılımı teslim etmek. 4- İş adamları ve yazılım geliştiriciler günlük olarak proje için bir araya gelmelidirler. 5- Projeleri bireyleri motive ederek uygulamak, onlara uygun ortamı sağlayarak ihtiyaçlarını karşılayıp işi bitireceklerine güvenmek önemlidir. 6- Takım içerisinde bilgi aktarımının en etkili ve kısa yolunun yüz yüze görüşmeler olduğunu bilmek önemlidir. 7- İşleyen yazılım, sürecin ilk göstergesidir. 8- Çevik süreç kararlı gelişim sunar. Sponsorlar, geliştiriciler ve kullanıcılar sürekli uyum içindedirler. 9- Teknik açıdan mükemmelliğe ve iyi tasarıma sürekli dikkat edilmesi çevikliği artırır. 10- Basitlik önemlidir. 11- En iyi mimariler, gereksinimler ve tasarımlar kendini organize edebilen takımlardan çıkar. 12- Takım, çalışmasının daha verimli nasıl olabileceğini inceler ve buna göre davranışlarını

iyileştirir. Manifesto değerlendirildiğinde, çevik yazılım geliştirmeye dayalı uygulama ve süreçlerde öne çıkan ve ortak özellikler olarak sınıflandırma işinde birer kategorik faktör olarak kullanılabilecek bazı faaliyetler göze çarpmaktadır. Birinci madde müşteri memnuniyetinin sağlanmasıyla ilgilidir. Bu da müşteri gereksinimlerinin en başta iyi belirlenmesi yani gereksinimlerin iyi "Analiz" edilmesi demektir. İkinci maddede vurgulanan konu değişikliklere cevaptır. Müşteri gereksinimlerindeki ya da çevre şartlarındaki değişiklikler, yazılımda yeni şartlara uyum sağlayacak düzenlemeler gerektirebilir. Bu ise yazılımda "İyileştirme" ile mümkündür. Üç, dört, beş, sekiz, on bir ve on ikinci maddeler sırasıyla; yazılımın kısa sürede tamamlanması, paydaşların sıklıkla toplanmaları, bireylerin motive edilmesi, sponsor, geliştirici ve kullanıcıların sürekli uyum içinde olması, takımların kendilerini organize edip verimli olmaları gerekliliğini vurgular. Yani bu maddeler sürenin etkin kullanımını, iletişimi, motivasyonu, uyumu ve kendi kendine organize olup verimli olmayı gerekli kıldığından hem yazılım süreci için hem de motivasyon, uyum ve verim için iyi bir "Yönetim" faaliyeti gerekliliği ortaya çıkmaktadır. Altıncı madde yazılım geliştiricilerin yüz yüze iletişimlerinin, "Gerçekleştirme" aşamasında yazılımın planlanan sürede ve amaca uygun olarak geliştirilmesine yardımcı olan bir konudur. Yedinci maddede yazılımın çalışmasının "Değerlendirme" açısından en iyi bir kriter olduğunu vurgulamaktadır. Dokuz ve onuncu maddelerde yazılımın iyileştirilerek mükemmelleştirilmesi ve kullanıcı açısından basitleştirilmesi köklü bir değişimin bir anlamda "Yeniden Tasarım" faaliyetinin gerçekleştirilmesine bağlıdır.

C. Çevik Yazılım Geliştirme Süreci Sınıflandırma Kategorileri

Çevik yazılım geliştirme manifestosunda yer alan maddelerin değerlendirilmesi sonucunda öne çıkan aşağıdaki faaliyetler sınıflandırmada birer faktör olarak kullanılmıştır.

Analiz: Gereksinimlerin ve dolayısıyla sürecin iyi analizi; başta doğru yazılım geliştirme süreç modelinin seçiminde sonda ise müşteri memnuniyetinin sağlanmasında çok yarar sağlar. Ne kadar çok veri toplanır ve derğerlendirilirse projenin başarılı bir şekilde tamamlanma şansı o kadar yüksek olur. Tasarım ve sistemi geliştirme aşamaları için kullanıcı ihtiyaçları ne kadar önemliyse, yazılım geliştirme süreç modeli seçiminin temelini oluşturan bilgi ve düzenleyiciler de projenin başarı durumu için o kadar önemlidir. (Sasankar & Chavan, 2011: 391). Bir projeye başlarken planlanması ve yeri gelince yapılması gereken SWOT analizi, Reorganizasyon, Değişim Yönetimi, Kurum Kültürü ve Rengineering çalışmaları vardır. Bunlardan SWOT analizi ele alındığında, rekabet gücünü koruma stratejilerini değerlendirmeye ve geliştirmeye yarayan bir araç olduğu görülür. Böylece süreç modelinin güçlü, zayıf, avantajlı ve tehdit olan yanlarını ortaya koyar. Uygun süreç modelinin seçimi, sadece geniş teknik bilginin yanı sıra uzman yöneticilerin deneyimlerine de gereksinim duyan karmaşık ve zorlu bir görevdir (Sasankar & Chavan, 2011: 399).

Yönetim: Genel proje yönetimi anlamı yanında süreç yönetimi anlamını da içermektedir. Yazılım süreç yönetiminin ilk kuralı dokümantasyon gereksinimlerinin yerine getirilmesidir. Doküman, yönetim ve kontrolün belli alanlarına yönelik kuralların

belirtildiği yazılardır. Kaynakların verimli kullanımı, zaman çizelgesine uyma, çalışanları organize ve kontrol etme ve işlerin tamamlanma noktalarını belirlemeye yönelik faaliyetleri içerir (NASA, 1990: 1). Yani yönetim faaliyeti aynı zamanda hem proje içindeki teknik faktörleri hem de sosyal faktör olan çalışanları ve müşterileri de kapsamaktadır

İyileştirme: Birçok yazılım mühendisliği organizasyonu ürün kalitesini ve takım verimliliğini artırma, ayrıca ürün geliştirme süresini azaltma adına yazılım geliştirme sürecini iyileştirmeye çalışmaktadırlar. Yazılım geliştirme sürecini iyileştirmeye odaklı modellerden en çok bilinenleri şunlardır. Ideal Model, ISO/IEC-12207, ISO 15504 (SPICE), Capability Maturity Model Integration (CMMI), Software Process Improvement (SPI), Total Quality Management (TQM), Quality Function Deployment (QFD), Function Point Analysis (FPA), Defect Prevention Process (DPP), Software Quality Assurance (SWQA), Configuration Management (CM), Software Reliability Engineering (SRE), vb. (Paulish, 1993; Akyol, 2015: 12).

Gerçekleştirme: Büyük yazılım sistemleri geliştirme işi; projelendirmeyi, faaliyetlerin koordinasyonunu, proje kontrolünü ve bilgi paylaşımını sağlayan araçlara sahip bir takım çalışmasını gerektirir. İş birliğine dayalı yazılım geliştirmedeki mevcut iş paylaşım ortamları, çoğunlukla geliştirme sürecini tanımlamaya ve otomatikleştirmeye çalışırlar. Birebir iletişimin yanı sıra planlama, tanımlama, değişiklik ve iş birliğine dayalı faaliyetlerin yönetimini esas alan paylaşımlı yazılım geliştirme ortamının kurulması gerekir. Bu ortam özellikle; net bir şekilde tanımlanmış iş birliği modeline göre değişebilen, çeşitli aktivitelere ve iş süreçlerine yol gösteren, yüksek seviyeli dinamik geliştirme sürecine zemin oluşturmalıdır (Altmann & Weinreich, 1998: 27). Gerçekleştirme aşaması sadece bir geliştirme aşaması değil, doğrudan yazılımın test, doğrulama ve geçerlileme gibi iç değerlendirme aşamalarını da içerir.

Yeniden Tasarım (Refactoring): Çevik yöntemler sözlüğünde, yeniden düzenleme, harici davranışını korurken mevcut bir yazılımın kaynak kodunun iç yapısını iyileştirmek şeklinde açıklanmaktadır (Agilealliance, 2019c). Yazılım geliştirme sürecinde köklü bir iyileştirme amacıyla radikal bir yeniden tasarım çalışmasını ifade eder. Radikal bir yeniden tasarım, mevcut süreçlerden ve prosedürlerden sıyrılma ve yeni yollar keşfetmedir. Köklü bir iyileştirme, performansta quantum sıçraması anlamına gelmektedir (Yousuf, 2012: 19). Çevik terminolojide refactoring kavramı tercih edilse de genel yazılım mühendisliği ve proje yönetimi terminolojilerinde reengineering kavramı kullanılmaktadır ki iki kavram da süreçler konusunda yeniden yapılanmayı ve bir paradigma değişikliğini kastetmektedir.

Değerlendirme: Projenin genel değerlendirmesi yanında süreç değerlendirme anlamını da içermektedir ki yazılımı dolayısıyla ele aldığından dış değerlendirme de denir. Yazılım geliştirme süreçlerinin etkin bir şekilde hedeflerine ulaşıp ulaşmadığını inceler. Bir sürecin kapasitesi, o sürecin alternatifleri ile birlikte kullanıcı ihtiyaçlarını karşılama kapasitesi demektir. Ayrıca değerlendirme faaliyeti, hangi yazılım geliştirme sürecinin kullanıcı ihtiyaçlarını daha çok karşıladığını ölçer.

Süreç değerlendirme, organizasyonların mevcut süreçlerini geliştirmede yardımcı olmaktadır. Ayrıca süreçte karşılaşılan güçlü, zayıf ve riskli yönleri ortaya çıkarmaktadır. Süreç değerlendirme süreç kapasitesini belirleme ve süreç iyileştirmeye öncülük eder. Süreç kapasite belirleme, organizasyon içerisindeki yazılım süreçlerini analiz eden düzenli bir değerlendirmedir. Ayrıca bir sürecin kapasitesini ve taşıdığı riskleri belirler. Organizasyonu yazılım süreç iyileştirmeye yönlendirir. Süreç iyileştirme yazılım süreçlerinde yapılacak değişiklikler anlamındadır (Thakur, 2016: 2).

III. Benzer Çalışmalar

Fernandes ve Almeida (2010), XP ve Scrum yöntemlerini yazılım gereksinimleri, yazılım gerçekleştirme, yazılım testi ve yazılım mühendisliği yönetimi özelliklerini karşılaştırmaktadır. Karşılaştırmada uygun değil, kısmen uygun ve tamamen uygun olarak üç seviyede ağırlıklandırma yapılmıştır. Yine Fernandes ve Almeida (2010) başka bir makalede çevik yöntemlerin sınıflandırılması ve karşılaştırılması konusunda çalışma yapmışlardır. Çevik yöntemlerin karşılaştırılması için özellik ve niteliklere dayalı kullanılan tekniği ve araçları tanımlamaktadırlar. Kullanılan nitelikler, IEEE Software Engineering Body of Knowledge (SWEBOK) Knowledge Area (KAs) ve Çevik Manifesto'da yayımlanan çevik prensiplerdir. Sonuçta XP ve Scrum'un kapsama alanları ve ne düzeyde çevik metodolojye uygun olduğu ortaya konmuştur.

Taheri ve Sadjadi (2015: 1), çevik yazılım geliştirme için özellik tabanlı (Feature Based) araç seçimi yaptığı çalışmada, mevcut araçları değerlendirmek için en iyi araçları ve özelliklerini tanımlayarak geçerli bir sınıflandırma modeli ortaya koymuştur. Sınıflandırma kriterleri esneklik, kullanım kolaylığı, kategori, maliyet, duyarlılık ve özelliklerdir. Sonucunda projenin ve işletmenin farklı seviyelerine göre gerekli araçları gösteren sınıflandırma modeli hazırlanmıştır.

Iacovelli ve Souveyet (2008: 91)'in yaptıkları çalışmada çevik yöntemlerin sınıflandırılması ve gereksinim mühendisliği süreçlerine olan etkisinin ölçülmesi amaçlanmıştır. Çalışmada kullanılan birçok yaklaşım vardır. Bunlardan ilki metotların sınıflandırılması ve karşılaştırılması için bir çerçeve oluşturmaktır. Çevik yöntemler zamanlama, takım, müşteri memnuniyeti, iş birliği, test, kalite gibi kriterlere göre karşılaştırılmıştır. İkincisi ise diğer metotlarda çevikliği sağlamak amacıyla bileşen tabanlı yaklaşım ortaya koymaktır. Bileşen tabanlı mühendislik yaklaşımda çevik yöntem bileşenlerini kullanarak mevcut çevik yöntemlerin çevikliğini artırılabileceği sonucuna ulaşmaktadır.

Tekinerdoğan (2003: 3) çalışmasında, yazılım metotlarının sınıflandırılmasında yapay kullanım senaryoları, desen ve alan güdümlü sınıflandırma tanımlamaları yapmaktadır. Bu çalışma ile ilgili yayımlanmış kitabı bulunmaktadır.

IV. Yöntembilim

A. Araştırmanın Amacı, Önemi, Alana Katkısı ve Modeli

Bu çalışmada öncelikle, yazılım mühendisliği ve çevik yöntemlerle ilgili karıştırılan temel kavramların açıklanması amaçlanmıştır. Sonra yazılım geliştirmede kullanılan

yöntem ve uygulamaların hangilerinin çevik yöntem oldukları ve hangi çevik faaliyet kategorisi altında sınıflandırılabileceklerinin araştırılması hedeflenmiştir. Çevik yöntemlerin çevik faaliyet kategorilerine göre sınıflandırılması önemlidir. Böylece gerçekleştirilecek bir çevik yazılım projesinin süreç özelliklerine göre hangi çevik yöntemin kullanılacağı kararına destek sağlanmış olacaktır.

Araştırma modeli şu şekildedir. Önemli ve esas kabul edilen kaynaklar taranarak hangi yöntemlerin çevik oldukları belirlenmiştir. Sonra çevik süreçler üzerinde etkin olan ve çevik yazılım geliştirme yöntemlerini özetler nitelikte olan faaliyet alanları birer sınıflandırma kategorisi olarak belirlenmiştir. Çevik olduğu belirlenen yöntemlerin öne çıkan faaliyetleri dikkate alınarak hangi kategoriye girdikleri belirlenip bu kategorilere göre bir sınıflandırma yapılmıştır.

B. Araştırmanın Kısıtları, Verilerin Toplanması ve Yöntem

Araştırmada kullanılan veriler ve bilgiler literatür taraması ile elde edilmiştir. Öncelikle sınıflandırma kategorileri hakkında kullanılan uygulama özelliklerini ortaya koyan çalışmalar değerlendirilmiştir. Çevik manifestoda, çevik yazılım geliştirme aracı olan Versionone'da ve literatürde sıklıkla geçen çevik yazılım geliştirme yöntemleri araştırılmıştır. Çevik yöntemlerin farklı kaynaklarda ortak olarak öne çıkan özelliklerini tanımlayan ve açıklayan veri ve bilgiler kullanılmıştır. Böylece analiz, yönetim, iyileştirme, gerçekleştirme, yeniden tasarım ve değerlendirme faaliyetleri olmak üzere sınıflandırma kategorileri ve çevik olarak belirlenen yöntemler için kullanılan uygulamaların özellikleri belirlenmiştir. Buna göre analiz, yönetim, iyileştirme, gerçekleştirme, yeniden tasarım ve değerlendirme olmak üzere altı sınıflandırma kategorisi ortaya konmuştur ve çalışma bunlarla sınırlıdır. Her bir kategori ve kullanılan uygulamaların özelliklerine dair bilgiler derlenerek Tablo 1 oluşturulmuştur. Kategorilerin ve uygulamaların özelliklerinin belirtilmesinde incelenen yöntemlerin öne çıkan nitelikleri göz önünde bulundurulmuştur.

Tablo 1. Sınıflandırma kategorileri için kullanılan uygulamaların özellikleri

Sınıflandırma Kategorileri	Kullanılan uygulamaların özellikleri
Yazılım süreci analizi	Kullanıcı ihtiyaçları, süreç modeli seçimi, modellerin güçlü, zayıf, avantajlı ve dezavantajlı yanları.
Yazılım süreç yönetimi	Dokümantasyon, kuralların belirtimi, süreç planı, tamamlanma noktaları, zamanlama.
Yazılım süreç iyileştirme	İyileştirme, ürün kalitesi ve takım verimliliğini artırma, ürün geliştirme süresini azaltma.
Yazılım süreç gerçekleştirme	İş birliği, koordinasyon, proje kontrolü, bilgi paylaşımı, kod yazımı, model kullanımı.
Yazılım süreci yeniden yapılanma	Köklü bir iyileştirme, radikal bir yeniden tasarım.

Yazılım süreç değerlendirme	Hedefe ulaşıp ulaşmama, kullanıcı ihtiyaçlarını karşılama kapasitesi, hangi sürecin ihtiyacı karşıladığının ölçümü, süreç geliştirmeye yardımcı, süreç iyileştirmeye yönlendirme.

C. Bulgular

Bu kısımda çevik özelliği taşıyan yöntemlerle ilgili temel bilgiler sunulmaktadır.

Gereksinim Güdümlü Uygulama Geliştirme (Requirements Driven Software Development-RDSD): Kullanıcı gereksinimlerinin ön plana çıktığı çeşitli araç ve yöntembilimler kullanılmaktadır. Kullanıcı senaryolarına dayanan, model güdümlü mimari üzerine inşa edilmiştir ve yeniden kullanımı en üst düzeye taşımayı hedeflemektedir. Soyutlanabilen yazılım durumları ile gereksinimlerden mimariye, mimariden tasarıma ve tasarımdan koda geçiş, araç seti tarafından otomatik ve/veya yarı-otomatik olarak gerçekleştirilmektedir. Temel hatları ile durum temelli çıkarım (case based reasoning) yaklaşımını; gereksinimlerden koda çevirebilecek şekilde ele alan, kapsamlı bir süreç sunmaktadır (Tüfekçi, Çokkeçeci & Çetin, 2015, s.2). RDSD için kullanılan bir diğer yöntemde ise başlangıç gereksinimlerinin kapsamını belirleyen Tropos adında yazılım geliştirme yöntembilimi oluşturulmuştur (Castro, Kolp & Mylopoulos, 2002, s.365). IRM-SA'da doğal dille yazılan sistem gereksinimlerinin sistem mimarisine sistematik bir şekilde çevrilmesini sağlar (Vinarek & Hnetynka, 2016: 59).

Rasyonel Birleştirilmiş Süreç (Rational Unified Process-RUP): IBM Rational tarafından geliştirilen bileşen, ürün ve belge temelli bir yazılım mühendisliği süreç önerisidir (Nizam, 2014: 446). İçindeki her aşamada yazılım mühendisliğinin tüm aktivitelerinin farklı oranlarda uygulanmakta ve her aşamada kodlama yapılabilmektedir. Süreç içindeki aşamaların amaçları öğrenildiği için her aşama sonunda oluşturulması gereken çıktılar sıralanabilir (Çatal, 2005). Birleştirilmiş Süreç modelinin bir uygulaması olan RUP (Kruchten, 2003: 1) dinamik, statik ve pratik olmak üzere üç farklı perspektife sahiptir. Dinamik perspektif, modelin zaman içerisinde başlama, detaylandırma, kurma ve geçiş safhalarını göstermektedir. Statik perspektif, süreç aktivitelerini göstermektedir. Pratik perspektif, süreçler esnasında kullanılacak örnekleri içermektedir. Rasyonel birleştirilmiş sürecin yeniliği, dinamik safhalar ile statik iş akışının birbirinden ayrılması ve yazılımın müşteri lokasyonunda kurulmasını sürecin bir parçası haline getirmesidir (Sommerville, 2011: 50).

Çevik Birleştirilmiş Süreç (Agile Unified Process-AUP): Scott Ambler tarafından geliştirilen IBM RUP'un sadeleştirilmiş sürümüdür. Zamanla oluşan artırımların kullanılabilmesi için her aşamada geniş yaşam döngüsüne ve iterasyonlara odaklanmaktadır (Edeki, 2013: 13). Yazılım yaşam döngüsünün başlangıcında mimariyi ve gereksinimleri belirlerken çevik unsurların tümünü göz önünde bulundurur (Baelen, 2011, s.23). Basit, kolay anlaşılır bir yaklaşımla verimliliği artırmada, çevik teknikler ve kavramlarla nasıl uygulamalar geliştirilebileceğini anlatır. Bu teknikleri kullanılırken çevik modelleme, değişiklik yönetimi ve Yeniden Tasarım'dan yararlanır (Kara, 2010a).

Temel Birleştirilmiş Süreç (Essential Unified Process-EssUP): Ivar Jacobson tarafından geliştirilmiştir. RUP tabanlı ve çevik yazılım geliştirme uygulamalarını benimser. En bilinen özellikleri uygulamaların ayrıştırılması ve basitleştirilmesidir (Hui, 2015: 472). RUP, CMMI ve Çevik geliştirmeden yola çıkılarak kullanım senaryoları, iteratif geliştirme, mimari bazlı geliştirme, takım uygulamaları ve süreç uygulamaları gibi pratikler tanımlanmıştır. Amaç, bunlardan doğru olan uygulamayı seçip sürece entegre etmektir. EssUp, her biri bir uygulamayı belirten bir oyun kartları topluluğu olarak düşünülebilir (Kara, 2010b).

Yazılım Yaşam Döngüsü (Application Lifecycle Management-ALM): Bir yazılım geliştirme projesinin yönetiminde tüm yönleri ile takımlara yardımcı olacak yazılım piyasasının ihtiyaçlarına uygun entegre araçları tanımlar. Özelleştirilmiş bir dizi araçlar sayesinde özelleştirilmiş kuralların yerine getirilmesini sağlayan bir dizi roller olarak tanımlanır (IBM, 2008: 4). Yönetim, geliştirme ve faaliyetler olmak üzere üç yönü vardır. Kapsamı yazılım geliştirme yaşam döngüsünden daha fazladır. Çevik yazılım geliştirme sürecini kontrol etmektedir. Çevik yazılım geliştirme ALM'nin bir parçasıdır (Chappell, 2008: 4). İçerisinde önemli yapı taşlarını barındırmaktadır. Örneğin; taleplerin yönetilmesi, ürünün yazılım anlamda mimarisinin çıkarılması, tasarlanması, kodlanması, test edilmesi, yaşayan süreçlerin incelenmesi, inceleme sonucunda ürünün iyileştirilmesi, bu iyileştirmeler sonucunda da yeni sürümlerin çıkarılması gibi kavramların yönetilmesini kolaylaştıran metodolojilere sahiptir (Eryılmaz, 2014).

Scrum: Projede izlenmesi gereken adımları basit ama önemli birkaç kuralla belirterek esnek yönetim sunmaktadır (Acm, 2016a). Gereksinimler ve uygulamalardan ziyade bir dizi proje yönetim değerlerini ve çalışmalarını benimsemektedir ve bunlar diğer yöntembilimlerle birleştirilebilir ya da tamamlayıcı olarak kullanılabilir. Taahhüt, odak, açıklık, saygı ve cesaret olmak üzere beş değer üzerine kuruludur: (Larman, 2003: 174; Ranniko, 2011: 24). En önemli çıktı sürecin şeffaf hale getirilerek süreç içerisinde aksayan noktaların açığa vurulmasıdır. Böylece aksaklıkları çözümleyerek sürekli iyileştirme yönünde proje ekibini motive eder (Acm, 2016a).

Test Otomasyonu (Test Automation-TA): Testler, üç kalite boyutu olan güvenilirlik, işlevsellik, uygulama performansı ve sistem performansı ile yürütülür. Bu üç kalite boyutunun her biri için; süreç, planlama, tasarım, gerçekleştirim, uygulama ve değerlendirmenin yaşam döngüsü boyunca ne zaman ve nasıl otomasyonun yapılacağını tanımlar. Özellikle iteratif yaklaşımlar için her iterasyonun ve de ürünün yeni sürümünün sonunda regresyon testinin yapılmasını sağlar (Rational, 2003: 4). Testin uygulanması ve sürekliliği için bu önemlidir. Test güdümlü yazılım geliştirme test otomasyonundan fazlasını içerir. Uygulamalarından birçoğu test güdümlü yazılım geliştirme odaklı değildir. Örneğin, gerçekleştirimden sonra kabul betiği yazmak için kullanılan Fitness gibi kabul otomasyon aracı kullanımı (Stamelos & Sfetsos, 2007: 209).

İşlerin Ayrışımı (Separation of Concerns-SoC): Yazılımın işlev ve özelliklerine göre kodlarını (sınıf, fonksiyon vb.) ayırma veya soyutlamadır. Farklı görevleri bir yerde yığmaktansa, farklı yerlerde tanımlanması gerektiğini öngörmektedir. Sistem herhangi bir aşamada işlerin değişik bakış açılarına göre bölünmektedir. Farklı bakış açısındaki

öncelikli amaç, benzer işlerin birlikte gruplaması ve farklı açılardaki işlerin daha az bağlantıya sahip olmasıdır (Hofmeister ve ark., 2007: 110). Bu bakış açıları bağımsız olarak modellenebilir ve birleşme düzeni tüm sistem modelini temsil eder. Farklı işler farklı modeller ve farklı teknikler gerektirir (Chen, Liu, Ravn, Stolz & Zhan 2009: 169).

Açık Birleştirilmiş Süreç (Open Unified Process-OpenUP): Yapılandırılmış yaşama döngüsü içerisinde iteratif ve artırımsal yaklaşımları uygulayan yalın birleştirilmiş bir süreçtir. Yazılım geliştirmede iş birliğine dayalı çevik yaklaşımları benimser. Değişik proje türlerinde kullanabilen bir süreçtir (EPF, 2012). Eclipse Vakfı tarafından açık kaynak kodlu süreç çerçevesi olarak geliştirmiş olan Eclipse Süreç Çerçevesi (EPF-Eclipse Process Framework)'nin bir parçasıdır. RUP aşamalarının ve iterasyonlarının isimlerini kullanmaktadır. OpenUP disiplinler, çalışma çıktıları, roller ve yaşam döngüsü süreçleri sayesinde yüksek bir düzenlenme seviyesine sahiptir. Bu hiyerarşinin temelini oluşturmaktadır ve sonraki aşamalarda süreç tanımlamalarına kolayca ulaşmayı ve de kullanmayı sağlayan dosya ve kategorilere ayrılır (IBM, 2008: 544).

Takım Yazılım Süreci (Team Software Process-TSP): Takım üyelerinin ortak kelimelerle iletişim kurmasında, işlerin planlanmasında ve izlenmesinde ortak ölçü ve standartların takip edilebileceği süreç oluşturma veya benimseme çerçevesi sağlar. Bireylerin iş kalitesini ve etkinliklerini geliştirebilecekleri şekilde tanımlanmış ve yapılandırılmış süreç önceliğini temel almaktadır (Humphrey, Chick & Nicolas, 2010: 1). Yazılım projeleri bireyler tarafından geliştirildiğinden bireylerin yetenek ve çalışma disiplinleri bir projenin başarısında doğrudan etkilidir. Yazılım mühendisliği alanında bugüne kadar tasarım, test ve yönetim alanlarında birçok çalışmalar yapılmakla beraber yazılım geliştirmede bireysel disiplinin üzerine fazla gidilmemiştir. Bu alandaki eksiklik Watt Humprey tarafından PSP (Personal Software Process) ve TSP (Team Software Process) çalışmaları ile giderilmiştir. PSP bireysel, TSP ise takım içinde yer alan disiplini belirtir. Açıkça anlaşılabileceği gibi TSP, PSP'yi kapsar. Fakat şu gerçek unutulmamalı ki takımlar bireylerden oluşur ve PSP'nin önemi burada ortaya çıkar. PSP'nin amacı takım çalışmasını başarıya eriştirmektir (Güçlü, 2004). TSP yazılım takımlarına özgü bir süreçtir ve takımlardan yüksek performans elde etmeyi amaçlamaktadır (Over, 2010: 8).

Microsoft Çözüm Çerçevesi (Microsoft Solutions Framework-MSF): Teknoloji projelerinde tanımlanmış ilkelere, modellere, disiplinlere, kavramlara, kurallara ve Microsoft tarafından kullanılmış uygulamalara dayalı bilinçli ve disiplinli bir yaklaşımdır. 1993'te ilk sürümü, 2005'te dördüncü sürümü çıkmıştır. Son sürümü ile de çevik geliştirme kullanımı, küçük takımlar ve birleştirilen roller öne çıkmaktadır. MSF kullanımı bir kuruma CMMI üçüncü seviye sertifika alma imkânı sağlar (Nizam, 2014: 438). Belli sebeplerden dolayı yöntembilim yerine çerçeve olarak adlandırılmaktadır. Metodolojinin bakış açısına karşılık, planlama, oluşturma ve iş tabanlı teknoloji kurulumu için herhangi bir proje (karmaşıklık boyutu ne olursa olsun) ihtiyaçlarını karşılamak için uyarlanabilen esnek ve ölçeklenebilir çerçeve sunmaktadır (Microsoft, 2003: 4). Başarılı bilişim çözümleri için insanların nasıl organize edileceği, projelerin nasıl planlanacağı, nasıl bir süreç yapısı üzerinden geçileceği, risklerin değerlendirilip kurulumun nasıl tamamlanacağıyla ilgili bir rehberdir (Microsoft, 2016: 1).

Test Güdümlü Yazılım (Test Driven Development-TDD): Diğer bir adlarıyla Test First Development ve Test Driven Design olarak anılmaktadır. Extreme Programming yazılım sürecinin oluşturucusu Kent Beck tarafından ortaya atılmıştır. Birçok çevik sürecin kodlama bakımından bel kemiğini oluşturur (Altuntaş, 2007: 1). Testi geçen gerçek program kodundan önce bir birim testi yazılır. Test, gerçek işlevsel kodu çalıştıran test kodu tarafından otomatikleştirilir. Geliştiriciler ilk testi gerçekleştirebilecekleri kısa bir döngüyü uygular ve test hata verdiğinde sadece testi geçecek işlevsel kodu yazarlar. Bu yaklaşım sistem için kapsamlı bir dizi testin oluşturulmasını ve kaliteli kod elde edilmesini sağlar (Ranniko, 2011: 31).

Hızlı Uygulama Geliştirme (Rapid Application Development-RAD): Geliştirilmesi planlanan yazılımın, tasarım ve geliştirme aşamalarının hızlı ve kaliteli bir şekilde gerçekleştirilmesi için CASE (Computer Aided Software Engineering-Bilgisayar Destekli Yazılım Mühendisliği) araçlarının kullanılması gerekliliğini savunmaktadır (Alkan, 2004). Özellikle üç aydan kısa sürede geliştirilebilecek modüler parçalara bölünebilen sistemler için uygun olan RAD yaklaşımı, teknik risklerin yüksek olduğu ve modüler parçalara bölünemeyen sistemler için uygun değildir (Pressman, 2001: 34).

Uç Programlama (Extreme Programming-XP): Takımlar en iyiye ulaşmada iyi ve denenmiş mühendislik uygulamalarını kullanırlar (Ranniko, 2011: 23; Sliger & Broderick, 2008: 296). Amaç, kaliteli yazılım, ustaca ve sürdürülebilir yazılım geliştirme teknikleri ve değişim esnekliği ile müşteri memnuniyetine hızlıca ulaşmaktır (Larman, 2003, s.174; Ranniko, 2011: 23). İletişim, basitlik, geribildirim ve cesaret olan dört değeri önemsemektedir. Beş temel prensibi ise; Hızlı geribildirim, basitlik, artırımsal değişim, değişime açık olma ve kaliteli iş çıkarmaktır (Cohn, 2004; Ranniko, 2011: 23).

Nesne Yönelimli Programlama (Object Oriented Programming-OOP): Problemin, temelindeki kavramsal çerçeve içerisinde nasıl çözüleceği düşüncesini tanımlama yoludur. Önemli avantajları; gerçek dünyayı kavrayışı, kararlılık, tasarım ve uygulamaların yeniden kullanılabilirliğidir. Gerçek dünyanın doğal algısına çok yakındır (Reddy ve ark., 2006; Krogdahl ve Olsen, 1986). Karmaşıklığı ya da boyutu artan yazılımların kolayca ve kısa sürede geliştirilebilmesi için tüm yazılım projelerinde kullanılmaktadır (Gürsoy, 2011).

Servis Odaklı Mimari (Service Oriented Architecture-SOA): 2000 Yılı sonrasında mobil donanım teknolojilerinin gelişmesi ile kullanılmaya başlanmıştır. Yazılım geliştirme platformlarının gelişmesiyle uygulamalarını zenginleştirmiştir. Farklı servislerin, bir harmoni oluşturan karmaşık yapılar içerisinde beraber çalışılabilmesi yaklaşımıdır. Dolayısıyla bu yaklaşım ile tasarlanmış bir sistemin alt sistemi ise doğal olarak servisin kendisidir. Servisler ise birbirleri ile iletişime geçebilen yapılardır. Başka bir deyişle servis bir bağlantının diğer tarafındaki dinleyicisidir (Yazıcıoğlu, 2011).

Bileşen Yönelimli Yazılım Mühendisliği (Component Based Development-CBD): Temel bileşenlerin birleşimi ile sistemin oluşturulması bakış açısından doğmuştur (Reddy vd., 2006: 33). Süreç, sistem özelliklerinin ayrımı ile başlayıp, ayrımı yapılan sistemin yazılım bileşenlerine dönüştürülmesi ile devam edip, bileşenlerin entegre edilip

hedeflenen sistemin oluşturulması ile sonuçlanmaktadır (Bayar, 2016: 1). Örneğin, bir otomobil, aynı ya da farklı üreticilerden edinilen parçaların bir araya getirilmesi ile oluşturulabilir. Günümüzde otomobil üretimi büyük ölçüde bu biçimde yapılmaktadır. Donanım üreticilerinin bu üretim tarzını yazılım üretimine uygulamaya çalıştığımızda "Bileşen Tabanlı" yazılım mühendisliği ortaya çıkar (Arifoğlu & Doğru, 2001: 272).

İlgiye Yönelik Programlama (Aspect Oriented Programing-AOP): Yazılım projeleri her geçen gün büyümektedir. Mümkün olduğunca OOP prensipleri uygulansa da, yeniden kullanılabilirlik arttırılmaya çalışılsa da bir süre sonra yazılan kodların anlaşılma düzeyi ve bakım yapılabilirliği düşmektedir. AOP bu noktada yardımcı olmaktadır. En büyük hedefi modülerliği arttırmak ve Cross-Cutting Concern'ler arasındaki ayrımı daha net sağlamaktır -Separation of Cross-Cutting Concerns- (İrgin, 2013: 1). Bu kavram yazılımın uygun izolasyonunu, uyumunu ve kodların yeniden kullanılabilirliğini ifade eder (Kiczales ve ark., 1997: 3).

Model Güdümlü Mimari (Model Driven Architecture-MDA): Aynı zamanda bir yazılım geliştirme yöntemidir. Yazılım geliştirme sürecinin modeller üzerinden yürütülmesidir. Burada modeller yazılım geliştirme işine soyut bir bakış açısı kazandırmış ve otomatik kod üretimi ile süreç hızlandırılmıştır. Soyutlama ile sistem tasarımı üzerinde çalışanların çözüm uzayı (kod uzayı) yerine probleme yoğunlaşmaları sağlanmıştır. Böylelikle sorunlar daha anlaşılır ve daha etkili çözümler üretilebilir hale getirilir (Akgül, 2010: 1). Gerçek program kodu modelleme ile derlenebilir. Bu yöntem yazılım sistemlerinde farklı seviyelerdeki soyutlamalar için geçerli bir çerçeve sunmaktadır. Nesne tabanlı yaklaşıma yakındır ve ayrıca servis tabanlı yaklaşımla da uygulanabilmektedir (Koskela, Rahikainen & Wan, 2007: 14).

Sürekli Entegrasyon (Continuous Integration-CI): Bir yazılım takımının ürettiği kodun sürekli ve sorunsuz olarak sisteme entegre edilmesini amaçlayan bir yaklaşımdır. Bu entegrasyon süreci, çeşitli araçlar ile otomatik hale getirilmekte ve belirlenen sürelerde çeşitli betikler ve işlemler çalıştırılarak kodun sürüme uygunluğu sürekli olarak test edilmektedir. Böylece entegrasyon hataları mümkün olan en kısa sürede bulunabilmektedir. Ayrıca, havuzda bulunan kodların belirli aralıklarla test senaryolarının koşturulması ile sistem tutarlılığı sağlanmakta, geliştirilen yerleştirme otomasyonu ile yerleştirme süreçlerinde, insan faktöründen kaynaklanabilecek hatalar en aza indirgenmektedir. Bir görevin işi tamamlanır tamamlanmaz sistemin tamamına entegre edilmektedir. Bu entegre işleminden sonra sistemdeki tüm birim testlerinin başarılı olması gerekir (Acm, 2016b; Mujumdar ve ark., 2012: 2019).

Yetenek Olgunluk Modeli (Capability Maturity Model-CMMI): Yazılım mühendisliği, sistem mühendisliği ve hizmet işletmeleri için kalite sağlayıcı üç ayrı süreç iyileştirme modeli sunar. Etkin bir yazılım sürecinin anahtar elemanlarını tanımlayan bir çerçeve modeldir. Olgun olmayan bir süreçten olgun ve disiplinli bir sürece giden bir yol çizer. Temel prensipler olarak yeteneğin olgunluğa bağlı olduğunu ve olgunluğun da zamanla gelişerek ölçülebileceğini belirtir (Akyol, 2015: 29). Yazılım geliştirme süreçlerinin ne kadar olgun olduğunu değerlendiren ve aynı işi yapan diğer organizasyonlarla karşılaştırma yapabilmeyi sağlayan bir araçtır (Ülgen, 2016: 2).

İşletmelerin projelerinde başarıyı sürdürmek için iyi tanımlanmış ve iyi dokümante edilmiş yazılım geliştirme süreçlerinin bir standardı olmuştur (Awad, 2005: 33).

ISO 15504 (SPICE): 1993'te Uluslararası Standartlar Örgütü (ISO), tarafından başlatılan bir çalışmanın ürünüdür. Süreç iyileştirme veya yetenek belirleme amaçlarıyla yazılım süreç değerlendirmesi için bir çerçeve sunar. Süreç ve yetenek olmak üzere iki boyutlu bir modeldir (Akyol, 2015: 23). Süreç iyileştirme, süreç kültürüne sahip olmak isteyen her kuruluş için ilk adımdır. Bütün kuruluşlar değişim çabasındadır ve süreç iyileştirme çalışmaları mutlaka planlaması gereken ve temel bazı verilere dayandırılması gereken bir çalışmadır. Aksi halde başarısız olma olasılığı yüksektir. ISO 15504, daha çok bilinen adı ile SPICE, bu konudaki en kapsamlı süreç standartlarından biridir (TÜRCERT, 2016: 1).

Yeniden Tasarım (Refactoring): Yazılımı daha basit, anlaşılır, değiştirmesi kolay bir hale getirmek için içyapısında yapılan ve dış davranışını etkilemeyen değişikliklerdir (Bilge, 2013: 1). Yeniden tasarıma, derleme, çalıştırma ve işlev yapabilecek küçük ayrık değişiklikler yapılarak başlanır. Mümkün olduğunca yapılan değişiklikler mevcut koda eklenir ve test edilir. Bir sonraki küçük ayrık değişikliğe geçilir. Aynı şekilde mevcut kodlamaya eklenir ve tekrar test edilir. Yeniden tasarım tamamlandığında ve tüm testler başarılı olduğunda geriye dönülür ve yerine yenisi yazılan kod mevcut kodlamadan çıkarılır. Yapılan en son testler de başarılı olduğunda yeniden tasarım tamamlanmış olur (Versionone, 2016).

Yalın Yazılım Geliştirme (Lean Software Developmen-LSD): LSD, Lean üretimi, Toyota üretim sistemi ve Bob Charette'nin Lean geliştiriminin birleşiminden elde edilmiştir (Sliger ve Broderick 2008: 297). Müşteri memnuniyetini artırma, boşa harcamaları ortadan kaldırma (Turan, 2011: 1), değer akışını optimize etme, insanları harekete geçirme ve sürekli gelişim üzerine odaklanmaktadır (Ebert, Abrahamsson & Oza, 2012: 1). Lean endüstriyel kuramların yazılım geliştirmeye uygulanması hakkında sağlıklı bilgi sunmaktadır. Özellikle projelerine engeller yerine değer eklemek isteyen proje yöneticileri, takım liderleri ve bilgi işlem yöneticileri için bir araç sunar. Mümkün olduğunca fikirleri geciktirir, çünkü fikir karara dönüşmediği sürece değiştirmesi kolaydır. Her türlü değişime uyum sağlayabilen dayanıklı, değişime açık tasarım geliştirmeyi esas almaktadır (Poppendieck & Poppendieck, 2003: 52).

V. Sonuç

Çevik manifesto ve ilgili literatür esas alınarak, çevik metodolojilerde süreçler üzerinde önde gelen altı faaliyet alanına göre kategorik sınıflar oluşturulmuştur. Bulgularda ele alınan çevik metodoloji, yöntem ve uygulamaların öne çıkan özellikleri incelenmiş ve buna göre hangi kategorik faaliyet sınıfına girdikleri Tablo 2'de verilerek bir sınıflandırma yapılmıştır.

Çevik yazılım "Süreç Analizi Kategorisi"nin kullanıcı ihtiyaçları, süreç modeli seçimi, modellerin güçlü, zayıf, avantajlı ve dezavantajlı yanlarını ortaya çıkarma özellikleri dikkate alındığında; RDSD kullanıcı odaklı olması ve gereksinimlere önem vermesi, RUP süreç modeli seçimi, AUP modellerin avantajlı yanlarını belirlemesi,

EssUP süreç modeli seçimi ve ALM ise süreç analizi yönüyle "Süreç Analizi Sınıfı"na girmektedirler. Yani bu çevik yöntemler analiz faaliyetini ön plana çıkarmaktadırlar.

Çevik yazılım "Süreç Yönetimi Kategorisi"nin dokümantasyon, kuralların belirtimi, süreç planı, tamamlanma noktaları ve zamanlama faaliyetleri odak noktasıdır. Bunlar dikkate alındığında; RUP süreç aşamaları, AUP uygulama planı, EssUP planlama, ALM süreç kontrolü, SCRUM yönetim kuralları, TA zamanlama, SoC süreç planlama, OpenUP ve TSP kuralların belirtimi ve MSF proje planlanma faaliyetlerini ön plana çıkarmaları nedeniyle "Süreç Yönetimi Sınıfı"na girmektedirler.

Çevik yazılım "Süreç İyileştirme Kategorisi"nde ise iyileştirme, ürün kalitesi, takım verimliliği, ürün gelişim süresinin azaltılması önemlidir. Buna göre başlıca süreç iyileştirme uygulamaları olan CMMI ve SPICE ile yazılım mimarilerinin temelini olan OOP, SOA, CBD ve AOP bu faaliyetlere yönelik olduklarından "Süreç İyileştirme Sınıfı"na girmektedirler.

Çevik yazılım "Süreç Gerçekleştirme Kategorisi"nde yer alan iş birliği, koordinasyon, proje kontrolü, bilgi paylaşımı, kod yazımı ve model kullanımı gibi çalışmaların yoğunlukta olduğu görülmektedir. Buna göre OpenUP süreçlerin uyumunu sağlama, TSP takım başarısı, MSF çalışanları organize etme, TDD test ve işlevsel kod yazımı, RAD koordinasyon, CI kodların entegrasyonu, XP iletişim ve değişime açık olmasıyla "Süreç Gerçekleştirme Sınıfı"na girmektedirler.

Çevik yazılım "Yeniden Yapılanma Kategorisi" için köklü bir iyileştirme ve radikal bir şekilde tasarımın yeniden gerçekleştirilmesi gerektiği görülmektedir. LSD kendi içinde refactoring, kod güncelleme ve boşa harcamaların ortadan kaldırılması gibi yeniden yapılanma faaliyetlerini barındırması sebebiyle "Yeniden Yapılanma Sınıfı"na girmektedir.

Çevik yazılımın "Süreç Değerlendirme Kategorisi"nde yer alan faaliyetler; hedefe ulaşıp ulaşmamaya, kullanıcı ihtiyaçlarını karşılama kapasitesine, süreçlerin ihtiyacı ne kadar karşıladığını değerlendirmeye, süreç geliştirmeye ve süreç iyileştirmeye yönlendirmeye yoğunlaşmaktadır. Buna göre ALM süreç inceleme, SCRUM aksayan noktaları belirleme ve XP geribildirim faaliyetleri ile "Süreç Değerlendirme Sınıfı"na girmektedirler.

Tablo 2. Kategorilere Göre Sınıflandırılmış Çevik Metodoloji, Yöntem ve Uygulamalar

Sınıflandırma Kategorileri	Sınıflandırılan Çevik Metodoloji, Yöntem ve Uygulamalar
Çevik Süreci Analizi	RSDS, RUP, AUP, EssUp, ALM
Çevik Süreç Yönetimi	RUP, AUP, EssUP, ALM, SCURUM, TA, SoC, OpenUP, TSP, MSF
Çevik Süreç İyileştirme	CMMI, SPICE, OOP, SOA, CBD, AOP
Çevik Süreç Gerçekleştirme	OpenUP, TSP, MSF, TDD, RAD, CI, XP

Çevik Süreç Yeniden Yapılanma	LSD
Çevik Süreç Değerlendirme	ALM, SCURUM, XP

Kaynaklar

Acm. (2016a). Mühendislik Pratikleri. İstanbul: Acm Software. Erişim Tarihi: 16 Mart 2016. http://www.acm-software.com/muhendislik-pratikleri/

Acm. (2016b). Scrum. İstanbul: Acm Software. Erişim Tarihi: 16 Mart 2016. http://www.acm-software.com/scrum/

Acuña, S. T. & Ferré, X. (2001). "Software Process Modelling". *World Multiconference on Systemics, Cybernetics and Informatics (ISAS-SCI),* 22-25 Temmuz, (ss. 237-242). ABD: Florida, Orlando. ISBN: 980-07-7541-2.

Agilealliance. (2019a). Agile Manifesto. Corryton, Tennessee, ABD: Agile Alliance. Erişim Tarihi: 24 Nisan 2019. https://www.agilealliance.org/agile101/the-agile-manifesto/

Agilealliance. Agile Manifesto. (2019b). Corryton, Tennessee, ABD: Agile Alliance. Erişim Tarihi: 24 Nisan 2019. https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

Agilealliance. (2019c). Agile Glossary and Terminology. Erişim Tarihi:13.10.2019. https://www.agilealliance.org/agile101/agile-glossary/#r

Akgül, Ö. (2010). "Model Güdümlü Yazılım Geliştirme" [Elektronik Versiyon]. *Bilişim Dergi*. 16, 1-4.

Akyol, M. (2015). Yazılım Kalitesi ve Standartlar [Sunum]. Erişim Tarihi: 20 Mart 2016. http://docplayer.biz.tr/5825934-Yazilim-kalitesi-ve-standartlar.html

Alkan, M. (2004). Rapid Application Development (RAD). Erişim Tarihi: 16 Mart 2016. http://www.csharpnedir.com/articles/read/?id=315

Altmann, J. & Weinreich, R. (1998). "An environment for cooperative software development realization and implications". *Thirty-First Hawaii International Conference*, 9 Haziran, (ss. 27-37). ABD: Hawaii, Kohala Coast. DOI: 10.1.1.615.166.

Altuntaş, C. Test Driven Development. (2007). Erişim Tarihi: 16 Mart 2016. http://www.cihataltuntas.com/test-driven-development/

Arifoğlu, A. & Doğru, A. (2001). *Yazılım Mühendisliği : yöntemler, metodolojiler, CASE ortamları, günün teknolojisi.* 1. Baskı. Ankara: SAS Bilişim Yayınları. ISBN: 975-97197-2-X.

Awad, M. A. (2005). *A Comparison between Agile and Traditional Software Development Methodologies.* Western Australia University. DOI: 10.1.1.464.6090.

Aybek, A. (2010). *Çevik Yazılım Geliştirme "Agile".* İstanbul: Acm Software. Erişim Tarihi: 16 Mart 2019. http://www.acm-software.com/Pdf/AboutAgile.pdf

Baelen, H. V. (2011). "Agile (Unified Process)", [Elektronik Versiyon]. *Agile Record*, 6, 22-24. ISSN 2191-1320

Bayar, V. (2016). Bilesen Yönelimli Yazılım Geliştirme için Süreç Modeli. Erişim Tarihi: 16 Mart 2016. http://www.emo.org.tr/ekler/de8edbbe1547786_ek.pdf

Bilge, İ. (2013). Refactoring Nedir? Erişim Tarihi: 16 Mart 2016. http://ibrahimbilge.com/refactoring

Camoğlu, K. (2008). Yazılım Geliştirme Süreci. Erişim Tarihi: 5 Nisan 2016 http://www.chip.com.tr/blog/kadircamoglu/Yazilim-Gelistirme-Sureci_524.html

Castro, J., Kolp, M. & Mylopoulos, J. (2002). "Towards requirements-driven information systems engineering the Tropos project". *Information Systems* 27(6), 365-389. DOI: 10.1016/S0306-4379(02)00012-1.

Çatal, Ç. (2005). RUP ve XP Süreçlerinin Karşılaştırılması. Erişim Tarihi: 16 Mart 2016. http://www.csharpnedir.com/articles/read/?id=458

Chappell, D. (2008). What is Application Lifecycle Management?. Kaliforniya: Chappell & Associates. Erişim Tarihi: 16 Mart 2019. www.davidchappell.com/WhatIsALM--Chappell.pdf

Chen, Z., Liu, Z., Ravn, A. P., Stolz, V. & Zhan, N. (2009). "Refinement and verification in component-based model-driven design". *Science of Computer Programming*, 74(4), 168-196. DOI: 10.1016/j.scico.2008.08.003.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development.* Usa: Addison Wesley. ISBN: 978-0321205681.

Demir, Ö., & Acar, M. (1992). *Sosyal Bilimler Sözlüğü*. İstanbul: Ağaç Yayıncılık. ISBN: 9752500102.

Ebert, C., Abrahamsson, P. & Oza, N. (2012). "Lean Software Development". *IEEE Software*, (5), 22-25. DOI: 10.1109/ICSECOMPANION.2007.46.

Edeki, C. (2013). "Agile Unified Process". International Journal of *Computer Science and Mobile Aplications*, 1(3), 13-17. ISSN: 2321-8363.

EPF. (2012). *What is OpenUP?* Ottawa, Ontario, Kanada: The Eclipse Foundation. Erişim Tarihi: 16 Mart, 2019. https://www.eclipse.org/epf/general/OpenUP.pdf

Eryılmaz, Ö. (2014). *Application Lifecycle Management Nedir?* Erişim Tarihi: 16 Mart 2016. http://www.omereryilmaz.com/alm-application-lifecycle-management-nedir/

Fernandes, J. M. & Almeida, M. (2010). "A Technique to Classify and Compare Agile Methods". *Departamento de Informática, Universidade do Minho,* (48), 385-386. ISBN: 978-3-642-13053-3

Fernandes, J. M. & Almeida, M. (2010). "Classification and Comparison of Agile Methods". *7th International Conference Information and Communications Technolog,* 29 Eylül-2 Ekim, (ss. 391-396). Portekiz: Oporto. DOI: 10.1109/QUATIC.2010.71.

Güçlü, T. (2004). Yazılım Geliştirmede Disiplin ve PSP Yöntemi. Erişim Tarihi: 16 Mart 2016. http://www.csharpnedir.com/articles/read/?id=321

Gürcan, V. B. (2013). Agile Software Development (Çevik Yazılım Geliştirme). Erişim Tarihi: 16 Mart 2016. http://www.barbarosgurcan.com/post/Agile-Software-Development

Gürsoy, İ. (2011). *Nesne Yönelimli Programlama (OOP) Nedir?* Erişim Tarihi: 16 Mart 2016. http://www.ismailgursoy.com.tr/nesne-yonelimli-programlama-oop-nedir/

Hofmeister, C., Kruchten, P., Nord, R. N., Obbink, H., Ran, A. & America, P. (2007). "A General Model of Software Architecture Design", [Elektronik Versiyon]. *Systems and Software* (80), 106-126. DOI: 10.1016/j.jss.2006.05.024.

Hui, Y. (2015). "Compare Essential Unified Process (EssUP) with Rational Unified Process (RUP)". *Industrial Electronics and Applications (ICIEA)*, 15-17 Haziran, (ss. 472-476). New Zealand: Auckland. DOI:10.1109/ICIEA.2015.7334159.

Humphrey, W. S., Chick, T. A. & Nicolas, W. (2010). "Team Software Process Body of Knowledge Technical Report (Rapor No: CMU/SEI-2010-TR-020)". Hanscom: Software Engineering Institute Carnegie Mellon University.

Iacovelli, A. & Souveyet, C. (2008). "Framework for Agile Methods Classification". *Proceedings of Model Driven Information Systems Engineering (MoDISE-EUS)*, 16-17 Haziran, (ss. 91-102). France: Montpellier. ISSN: 1613-0073.

IBM (2008). "Collaborative Application Lifecycle Management with IBM Rational Products (Rapor No: SG24-7622-00)". New York: International Business Machines Corporation (IBM).

İrgin, D. (2013). Aspect-Oriented Programming Kavramı. Erişim Tarihi: 16 Mart 2016. http://www.denizirgin.com/post/2013/10/07/Aspect-Oriented-Programming

Kara, B. (2010a). Agile Unified Process.. Erişim Tarihi: 16 Mart 2016. İş Analizi Nedir: http://www.isanalizinedir.com

Kara, B. (2010b). Essential Unified Process. Erişim Tarihi: 16 Mart 2016. http://www.isanalizinedir.com

Keskinkılıç, M., Özmen, E. (2018). "Yazılım Projelerinde Yazılım Geliştiricilerin Yazılım Süreç Modellerini Kullanım Farkındalıkları". *Akademi Sosyal Bilimler Dergisi*, 5 (15), 61-78.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M. (1997). "Aspect-Oriented Programming". *ECOOP'97—Object-oriented*

programming, 9-13 Haziran, (ss. 220-242). Finland: Jyväskylä. ISBN 978-3-540-63089-0

Koskela, M., Rahikainen, M. & Wan, T. (2007). Software Development Methods: SOA vs. CBD, OO and AOP. DOI: 10.1.1.111.9582.

Kruchten, P. (2003). "The Rational Unified Process White Paper (TP026B, Rev 11/01)". Boston: Addison-Wesley.

Larman, C. (2003). *Agile and Iterative Development.* Boston Massachusetts, ABD: Addison Wesley. ISBN: 0-13-111155-8.

Mahanti, R., Neogi, M. S. & Bhattacherjee, V. (2012). "Factors Affecting the Choice of Software Life Cycle Models in the Software Industry". *Computer Science* 8(8), 1253-1262. ISSN: 1549-3636.

McGregor, S. L. & Murnane, J. A. (2010). "Paradigm, methodology and method intellectual integrity in consumer scholarship". *International Journal of Consumer Studies*, (34), 419-427. DOI: 10.1111/j.1470-6431.2010.00883.x.

Microsoft (2003). "Microsoft Solutions Framework version 3.0 Overview White Paper (MSF_V3)". Microsoft Corporation.

MSF. (2016). Microsoft Solutions Framework Overview. Erişim Tarihi: 16 Mart 2016. https://msdn.microsoft.com/tr-tr/library/jj161047(v=vs.120).aspx

Mujumdar, A., Masiwal, G. & Chawan, P. M. (2012). "Analysis of various Software Process Models. *Engineering Research and Applications"*, 2(3), 2015-2021. ISSN: 2248-9622.

NASA. (1990). "Manager's Handbook for Software Development (No: SEL-84-101)" . Greenbelt: Goddard Space.

Nizam, A. (2014). Yazılım Proje Yönetimi. 1. Basım. İstanbul: Papatya Yayıncılık. ISBN 978-605-4220-74-4.

O'Regan G. (2017). *Concise Guide to Software Engineering from Fundamentals to Application Methods (Undergraduate Topics in Computer Science)*, Springer International Publishing AG 2017. ISBN 978-3-319-57749-4.

Oussalah, M. C. (2014). *Software Architecture 1.* Somerset, US: Wiley-ISTE. ISBN: 9781848216006.

Over, J. (2010). *Introduction to the Team Software Process.* Pittsburgh: Software Engineering Institute Carnegie Mellon University.

Paulish, D. J. (1993). Paulish, D. J. (1993). "Software process improvement methods technical report (No: CMU/SEI-92-TR-026)". Carnegie-Mellon Univ. Pittsburgh Pa Software Engineering Inst.

Poppendieck, M. & Poppendieck, T. (2003). *Lean Software Development.* Boston: Addison Wesley. ISBN: 978-0321150783.

Pressman, R. S. (2001). *Software Engineering: A practitioner's approach(5'th Edition).* Usa: McGraw-Hill Eduactıon. ISBN: 0073655783.

Ranniko, P. (2011). *User-Centered Design in Agile Software Development*. (Yayımlanmış Master Tezi). University of Tampere School of Information Sciences.

Rational, S. (2003). *The Rational Unified Process: An Introduction* (3rd Ed.). Lexington: Rational Software Development. ISBN: 0-321-19770-4.

Reddy, A. R., Naidu, M. M. ve Govindarajulu, P. (2006). "Programming Methodologies and Software Architecture". *Computer Science and Network Security*, 6(11), 29-39. ISSN: 1738-7906

Royce, W. W. (1987). "Managing the Development of Large Software Systems : Concepts and techniques ". *9th international conference on Software Engineering,* 30 Mart-2 Nisan, (ss. 328-338). Kalifornya: IEEE Computer Society. ISBN: 0-89791-216-0

Sarıdoğan, M, Erhan. (2004). Yazılım Mühendisliği, 1. Basım. İstanbul: Papatya Yayıncılık. ISBN 975-6797-57-6.

Sasankar, A. B. & Chavan, V. (2011). "SWOT Analysis of Software Development Process Models". *International Journal of Computer Science*, 8(5), 390-399. ISSN: 1694-0814.

Sliger, M. & Broderick, S. (2008). *The Software Project Manager`s.* Usa: Adison Wesley. ISBN: 978-0321502759

Sommerville, I. (2011). *Software Engineering* (9'th Ed.). Usa: Pearson Education Inc. ISBN: 978-0137035151.

Stamelos, I. G. & Sfetsos, P. (2007). *Agile Software Development Quality Assurance.* ISBN: 978-1599042169.

Taheri, M. & Sadjadi, S. M. (2015). "Tool-Selection Classification Software Development". *The 27th International Conference on Software Engineering and Knowledge Engineering Conference*, 6-8 Haziran, (ss. 700-704). ABD: Pittsburgh, Wyndham Pittsburgh University Center. DOI: 10.18293/SEKE2015-234

Tekinerdoğan, B. (2003). *Classifying Software Architecture Design Methods*. Enschede: Twente Research and Education on Software Engineering.

Thakur, D. Software Process Assessment. (2016). Erişim Tarihi: 16 Mart 2016. ecomputernotes.com/software-engineering/software-process-assessment

Tüfekçi, Ö., Çokkeçeci, İ. & Çetin, S. (2015). Gereksinim Güdümlü Uygulama Geliştirme. Erişim Tarihi: 16 Mart 2016. www.cs.com.tr/TR/themes/touch/images/haberler/

Türkiye Bilişim Ansiklopedisi (2006). İstanbul: Papatya Yayıncılık. ISBN 975-6797-38-X

Turan, O. (2011). Lean Düşüncesinin Doğuşu. Erişim Tarihi: 16 Mart 2016. www.prince2turkey.com/cevik-yontemler/33-lean-yazilim-gelistirme.html

Türcert. (2016). ISO 15504 Yazılım Süreç Değerlendirme Sistemi. Erişim Tarihi: 16 Mart 2016. http://www.turcert.com/belgelendirme/sistem-belgelendirme/

Ülgen, S. (2016). Yazılım Geliştirme Süreçleri. Erişim Tarihi: 16 Mart 2016. http://www.sulc3.com/surecler.html

Versionone. (2016). What Is Agile Methodology? Erişim Tarihi: 4 Ağustos 2016. https://www.versionone.com/agile-101/agile-methodologies

Vinarek, J. & Hnetynka, P. (2016). "Towards an Automated Requirements-driven Development of Smart Cyber-Physical Systems". *International Workshop on Formal Engineering approaches to Software Components & Architectures*, 3 Nisan, (ss. 59-68). Netherlands: Eindhoven. DOI: 10.4204/EPTCS.205.5.

Yazıcıoğlu, S. (2011). *SOA Nedir?* Erişim Tarihi: 16 Mart 2016. http://blog.serkanyazicioglu.com/2011/12/soa-nedir/

Yazılım Geliştirme Süreci. (t.y). Erişim Tarihi: 27.05.2018, https://forum.java.com.tr/yazilim-gelistirme-sureci/

Yousuf, F. (2012).Reengineering Software Development. Erişim Tarihi: 17 Mart 2016. slideshare.net/faizayousuf/reengineering-software-development-process