DEÜ FMD 22(66), 741-748, 2020

Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi
Dokuz Eylul University Faculty of Engineering Journal of Science and Engineering

Basılı/Printed ISSN: 1302-9304. Elektronik/Online ISSN: 2547-958X

Bir Gelişimsel Yazılım Geliştirme Metodolojisi

A Progressive Software Development Methodology

Kökten Ulaş Birant [1*], Alp Kut [2]

[1, 2] Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği Bölümü, İzmir, TÜRKİYE
Sorumlu Yazar / Corresponding Author *: ulas@cs.deu.edu.tr

Geliş Tarihi / Received: 30.01.2020 *Araştırma Makalesi/Research Article*
Kabul Tarihi / Accepted: 21.04.2020 *DOI:10.21205/deufmd.2020226609*
Atıf şekli/ How to cite: BIRANT, K.U.,KUT, A.(2020). Bir Gelişimsel Yazılım Geliştirme Metodolojisi. DEUFMD 22(66), 741-748.

Öz

Yazılım Geliştirme Metodolojileri, farklı ürün yapılarına ve farklı çalışma ortamlarına uygun olarak, yazılım geliştirme ekiplerinin performanslarının arttırılması ve geliştirilen yazılım üzerinde daha etkin bakım adımları yürütülmesini sağlamak amacıyla hazırlanan süreçlerdir. Bu süreçlerin tanımlanması ve uygulanmasında, başarısı kanıtlanmış süreçlerin kullanılması tercih edilse de, çoğu zaman ekiplerin kendilerine özgü durumları bu süreçlerin başarılı sonuçlar elde etmesini imkansız hale getirir. Bu nedenle hem yazılım geliştirme ekibinin çalışma performansını yükseltecek, hem de gerçekçi bir süreç oluşturulması önemli bir fayda sağlayacaktır. Hazırlanan çalışma ile yazılım geliştirme ekiplerinin adım adım gelişimini sağlayacak ve uygulanabilirliği yüksek bir gelişimsel metodoloji açıklanmaktadır.

Anahtar Kelimeler: *Yazılım Mühendisliği, Yazılım Geliştirme Metodları, Süreç Yönetimi, Yazılım Yönetimi*

Abstract

Software Development .Methodologies, according to different product types and different working environments, are processes, which are prepared for increasing the performance of Software development teams and for providing more effective maintenance steps on developed software. Even though proven ones are chosen on defining and implementations of these processes, most of time particular situations of development teams cause unsuccessfull results. Therefore it will be very useful to prepare a process to increase the performance of developers and to be realistic. According to this study, a progressive and usable methodology, which provides a step-by-step improvement for company is explained.

Keywords: *Software Engineering, Software Development Methodology, Process Management, Software Management*

1. Giriş

Yazılım Geliştirme Metodolojileri, programcılıktan yazılım mühendisliğine geçiş sürecinin ana unsurlarından birisi olarak kabul edilir. Program yazmanın en büyük sorunları arasında gösterilen plansızlık ve kırılganlık (çok fazla sayıda parametreye bağlılık) sorunlarını çözme yolunda farklı çözümler [10] sunulmuştur.

Çalışmada tanımlanan model yardımıyla, düzenli geliştirme yapısına sahip olmayan firmaların adımlar yoluyla ilerleyerek daha düzenli, kontrollü ve geliştirilebilir bir model çerçevesinde kaliteli ürünler geliştirmeleri amacıyla bir metod tanımlanmaktadır.

2. Art Alan ve İlgili Çalışmalar

Literatürde tanımlanan çözümlerin ilki ve belki de ana tanımı ortaya koyan Dr. W. Royce Şelale (İng. Waterfall) yöntemini anlattığı çalışmasında, kapsamlı yazılım projelerinde çalışmanın sonrasında elde ettiği deneyim ile süreci öncelikle iki adıma ayırır; Analiz ve Kodlama. Sadece "Kodlama" olarak düşünülen geliştirme sürecini tanımlamaya yönelik ilk adım olarak geliştirme öncesindeki ihtiyaçların tanımlanması çalışmasını koyan Dr. Royce, ilgili yazısında detaylı bir açıklama ile "Şelale Yöntemi"ni ortaya koymuştur. Bugün de, tüm süreçler bu başlıkları esas alarak geliştirilmektedir. [9]

Yaşanan sorunlarda beklenen büyük çözümün gerçekleşmemesi sonucunda tanımlanan yeni süreçlerde en öne çıkan ise "Spiral Model" olur. B. Boehm, ilgili makalesinde örnek aldığı süreçlerde yaşanan sıkıntılar olarak risk tanımlamalarını değerlendirilmemesini (süreç dahilindeki analizlerin yetersiz olmasını), temel analiz çalışmalarının yetersiz olmasını, geliştirme sürecinde geri dönüşlerin fazla efor gerektirmesini (bu nedenle geri dönüşlerin hem yüksek maliyetli olmasını, hem de göz ardı edilebildiğini) göstermiş ve yeni modeli önermiştir. Önerilen model; kontrol adımlarının arttığı, kullanıcı ile iletişimin ve dolayısıyla geri dönüşlerin kolaylaştırıldığı ve raporlama ihtiyaçlarını arttırmak suretiyle disipline vurgu yapılan bir çalışmadır. [3]

Modellerin, uygulama ile uyumsuzluğunda en önemli neden olarak kullanıcı ihtiyaçlarının değişimi gösterilmektedir. İhtiyaçlardaki değişim (veya analiz sürecinde yetersiz veya eksik inceleme sonucunda hatalı ihtiyaç tanımlaması yapılması) neden olarak kabul edildiğinde ise yapılan geliştirmeler, "İhtiyaçların toplandığı ve işlendiği" adımlarda odaklanmıştır. Bu konuda gelişmeler incelendiğinde, "hızlı ve sarfedilebilir prototipleme" (İng. "Rapid Throwaway Prototyping") ile analizin teyidlenmesinin ve geribildirim sürecinin daha sağlıklı yürütülmesinin amaçlandığı görülmektedir. Sonrasında "Arttırımlı Geliştirme" (ing. Incremental Development) sürecinin geliştirilmesinde de, ihtiyaç analizi çalışmasının proje başlangıcında yeterli verimlilikte olmaması ve projenin gelişim sürecine yayılması görülmektedir. Son olarak "Evrimsel Prototipleme" süreci de hazırlanan prototip üzerinden geliştirmenin devam etmesini (buna bağlı olarak gelişimin sürekliliğini) önerir. [4]

Yakın dönemde ortaya çıkmış olan "Çevik Yazılım Geliştirme" felsefesine uygun metodolojiler ise, "anti-metodoloji" kavramı dahilinde değerlendirilse de, belirlenmiş format ve kurallara uygun bir geliştirmeyi önerir. Kendisinden önceki felsefeye karşıt olarak ise, kullanıcı ihtiyaçlarının toplanması ve bağlı çözümün üretilmesini sürecin bir adımı olmaktan çok sürecin kendisi haline getirir ve geliştirme de dahil olmak üzere her adımı kullanıcı ihtiyaçlarının anlaşılmasına yönelik bir fırsat olarak değerlendirir. Ayrıca önceki metodlardan farklılık olarak, geliştiriciye olan güven ve metoda değil, sonuca odaklılık görülmektedir. [5]

Tüm metodların çalışmaları değerlendirildiğinde ise, temel yaşam döngüsünde "İhtiyaçların Analizi", "Çözümün Tasarımı", "Yazılımın Geliştirilmesi (ve Sınanması)", "Bakım" adımlarının bulunması gerektiği, ancak bu adımların uygulanış şekillerinin ve sıralamalarının farklı parametrelere göre değerlendirilmesi gerektiği görülmektedir. Bu parametreler içinde geliştiricilerin özelliklerinden ürün ve ortam özelliklerine kadar çok farklı bilgiler bulunsa da, yazılım geliştirme için adımların belirlenmeye başladığı görülmektedir. [6]

Başka bir görüş de, yazılım geliştirme sürecinin tanımlanamaz adımlardan ve gelişme dahilinde netleştirilebilir bilgiden oluştuğunu savunarak, yazılım geliştirme sürecini sürekli ürünün gelişimine bağlı "Evrimleşme" olarak sunar. Bu şekilde bakıldığında tüm yazılım geliştirme adımlarından öte yazılım geliştirme sürecini ihtiyaç olan bilgilerin alınması ve çözümlerin geliştirilmesi amacında birleşerek gelişen bir evrim süreci olarak tanımlamak mümkündür. [8]

2015 yılında yapılan bir araştırma ile yazılım mühendisliği süreçlerinin seçiminde hangi kriterlerin değerli olduğuna yönelik çeşitli bulgular sunmuştur. Katılımcıların yarısından çoğunun proje lideri veya takım lideri olduğu ve genellikle 1 yıla yakın sürelerde yeni fonksiyon ve ürünler geliştirilmesine yönelik yazılım firmalarında yapılan çalışmada proje ekiplerinin beşte birinin Waterfall, beşte üçünün çeşitli çevik metodlar ve geri kalan projelerde de çeşitli farklı metodlar kullandıkları tespit edilmiştir.

Metodların seçimi ve geliştirilmesinde de takım büyüklüğü, proje bütçesi ve organizasyonel ihtiyaçların önemi görülmüş olup, yoğunlukla çevik metodları temel alan hibrit yöntemlerin öne çıkmaya başladığı raporlanmıştır. [15]

2016 yılında "Çevik Metodoloji"lerin adaptasyon özelliklerine yönelik yapılan bir araştırmada bu konudaki motivasyonlar üzerine de çalışılmıştır. Bu konudaki temel motivasyonların markete çıkış hızını arttırmak, değişen önceliklere hızlı reaksiyon verebilmek ve sonrasında da mühendislik disiplinine uygunluğu arttırmak olduğu görülmüştür. Burada özellikle yazılım ürününün temel faydalarının (ve temel sorunlarının) çevik yazılım metodolojilerine adaptasyonda önemli bir motivasyon kaynağı olduğu görülmüştür. (Bu noktada "mühendislik disiplini" vurgusu da, yapılan çalışmaya katılanların genellikle mühendislik disiplinine uygun olmayan süreçlerden geldiklerine işaret edebilmektedir.) [13] [14]

Bu noktada süreçlerin anlamlılığı veya faydalılığı tartışılabilir ki, alınan sonuçlar, metodların gelişimine neden olduğu gibi, duruma çok bağımlı, görece değerlendirmeler sunar. Ancak son dönemde gelişen bir tanımlama olan, "Olgunluk Modelleri" nesnel bir yorum kazandırabilir ki, bu modellerden en tanınmış olanı "Yapabilirlik Olgunluk Modeli" (İng. "Capability Maturity Model") ismiyle gelişen modeldir. Bu modelde farklı aşamalarda tanımlanan firmalar, aşamalarda belirlenmiş yetkinlikleri ve başarımları ile sınıflandırılabilirler. Bu durum aynı zamanda kendi durumuna en uygun modeli seçmiş firma için modelin de olgunluk başarımı olarak düşünülebilir. [7] [2]

Ayrıca yazılım geliştirme süreçleri literatüründe en çok tartışılan konulardan birisi de, mühendislik çalışmaları açısından zayıf oldukları algısı bulunan "Çevik Metodlar"ın, yönetimsel düzenliliği önemseyen "Olgunluk Modeli" ile değerlendirilmesidir ki, bu alanda yapılan yakın zamanlı araştırmalar, çevik süreçlerin temelde olgunluk açısından ciddi sıkıntıları olmadığını göstermektedir. [11]

3. Gelişimsel Süreç

"Yazılım Geliştirme Modelleri"nin gelişimi incelendiğinde görülmektedir ki, farklı geliştirme modellerinin tanımlanmasında durumları tanımlayan parametreler, bir başka deyişle sorunlar ve çözüm yöntemleri esas olmuştur. İlk sorun, yazılım geliştirmenin temel problemlerine çözüm üretmek iken, zamanla geliştirilen modellerin geliştirici bağımsızlığını ve yaratıcılığını arttırmasıyla bu temel problemlerin bir nebze çözüldüğü görülmüştür. Ancak bu şekilde bir yaratıcılık artışının beraberinde getirdiği sorunların başında düzensizliğin de artışı bulunmaktadır. Bu ana sorunun alt sorunları olarak da, kişiye bağımlılık, öngörülememe, zor bakım yapılabilirlik ve dolayısıyla plansızlık görülmektedir. Bir firmanın daha düzenli ve planlı geliştirme yapabilmesinin yolu da, geliştirme metodunu "ıslah" edebilmesinden geçer.

3.1. 4 Değerler

Çoğu çevik modelde olduğu gibi bu modelde de geliştirici grubunun belirli kurallara uyması, geliştirilen proje yapısının belli özelliklerde olması beklenmektedir. Bu kurallara uyulması ve şartların bu koşullara izin vermesi durumunda çalışmanın başarıya ulaşması olasılığından bahsedilebilir.

İletişim: Geliştiriciler arasında sözlü iletişimin yüksek olduğu bir ortamda, özellikle sözlü iletişimin yüksek tutulabileceği bir proje yapısında çalışılması gerekmektedir. Özellikle yüzyüze yapılan sözlü iletişim, fikirlerin aktarılmasında kelime anlamlarından daha kritik öneme sahip olan jest, mimik, tonlama, vurgu gibi unsurların yok olmasını engellemektedir. Bu nedenle geliştirilecek proje ekibinin üye karakterleri, özellikleri, fiziksel konumları, büyüklüğü gibi tüm özelliklerinin sözlü iletişimi (mümkünse yüzyüze) mümkün kılacak şekilde olması gerekmektedir. Bu durum aynı zamanda yazılı iletişimin kaybettirdiği duygu aktarımlarını ve zamanı da telafi etmeye yönelik bir özellik olup, ekibin ve projenin bu duruma uygunluğu önem arz etmektedir.

Koordinasyon: Bağımsız çalışılan bir ortamdan çok, koordine çalışan yaratıcı bireylerden oluşan bir ekip planlaması ve bu koordinasyonu sağlayabilecek bir yöneticinin bulunabileceği bir geliştirici ekip olması gerekmektedir. Özellikle "Arttırımlı Geliştirme" [4] (İng. Incremental Development) modelinde önemli yer tutan; geliştirilecek projenin alt modüllere ayrılmak suretiyle mümkün olduğu kadar paralel geliştirilebilmesi ve modüllerin sürekli entegrasyonu detaylı bir analiz yeteneğinin yanında başarılı bir koordinasyon becerisi ve isteğini de zorunlu kılmaktadır. Özellikle

metodun başlangıç noktasının "kişilere bağımlı" ve görece düzensiz bir geliştirme süreci olduğunu düşünecek olursak, etkileşim ile sorunları azaltabilmenin ve hızlı adaptasyonun yolu ancak yüksek koordinasyon becerisine sahip yönetim ve ekip üyelerinden geçecektir.

Cesaret: Geliştirici bireylerin kullanıcı ile gelişmeleri paylaşacak, sürekli kontrolü sağlayacak ve ortaya çıkan düzeltmeleri yapacak bir anlayışa sahip olmaları gerekmektedir. Çoğu çevik yazılım geliştirme metodunda da tanımlandığı gibi, yazılım geliştirme sürecinin ana sorunu hata ve eksiklerin erken aşamalarda ortaya konamamasıdır. Buna neden olarak da, yazılım geliştiricilerin geri bildirim almak, hataları düzeltmek, diğer geliştiricilerle (eleştiri almak veya eleştirmek amacıyla) iletişim kurmak ve gerektiğinde daha önceki çalışmalardan vazgeçebilmek (ve yeni bir çözüme adım atabilmek) konularındaki cesaret eksikliği gösterilmektedir. Bu ve benzeri başlıklar ile ilgili olarak gerekli cesaretin gösterilmesi hataların hızlı (ve doğal olarak en az maliyetle) çözülebilmesi anlamına gelecektir. Çemberler sürecinde de, diğer çevik yazılım geliştirme süreçlerindeki gibi önemli olmasının yanında, geliştirme ekibinin hızlı öğrenebilmesi ve kendini yenileyebilmesi açılarından önemi büyük olacaktır.

Sürekli Kodlama: Yazılım geliştirme, "program yazma" teriminin ötesine geçme noktasında en önemli aşamayı, geliştirmenin bir sürece yayılması ile gösterir. Özellikle yazılım ürününün en büyük problemleri olarak görünen, hatalı ürün geliştirme (ve bu hataların düzeltilmesinde öngörü oluşturamama) ve hatalı değerlendirme (ilerleme tahminlemesi) sorunlarının ana çözümü olarak da, ihtiyaçların analizinden, çözümün tasarımına bir ürün geliştirmeyi süreç adımlarına bölmeyi ortaya koymuştur. Bu durum, yazılım üretimi ile ilgili sorunları genel olarak çözüyor gibi görünse de, ürünün diğer mühendislik ürünlerine yönelik avantajlarını (hızlı üretim ve yenileme) törpülemiş, aynı zamanda yazılım geliştiricilerin deneme – yanılma hatasına düşebilse de, hızlı geliştirme becerilerini azaltmıştır. Bunun da yaratıcılık ve müşteri ilişkilerinin geliştirilmesi açılarından dezavantajı görülmektedir. Bu nedenle Çemberler sürecindeki geliştirme ekibinin, tüm uygulamayı kodlama perspektifinden incelemesi, tüm adımlarda "kodlanacak" bir yazılım ürünü geliştirileceği odağıyla çalışarak adım sonuçlarına yönelik adımları çalışması beklenmektedir. (Bir başka deyişle, süreç adımlarının her anında yapılan çalışmanın kodlama ile test edilmesi veya sürece hızlı bir şekilde etkilerinin incelenmesi amaçlanır.)

3.2. Adımlar

"Çemberler" metodu, adından da anlaşılacağı gibi, bir firmanın yeterlilikleri ve ihtiyaçlarına bağlı olarak içiçe geçmiş 3 adet sürecin uygulanması ile hayata geçirilen bir firma yetkinleşme sürecidir. Yazılım ekibinin başlangıç durumundaki kıt kaynakları ve uygulama ihtiyaçlarına bakılarak hazırlanmış bir iç çembere uyumlanması ile başlayan süreç, gelişmiş ve yetkinleşmiş bir yazılım firmasının elindeki kaynaklar ile büyümenin getirdiği ihtiyaçlarını gidermesini sağlayacak bir dış çember ile sonuçlanır. Söz konusu 3 çember, adımları ve raporlama ihtiyaçları açılarından "1. Başlangıç Çemberi", "2. Gelişme Çemberi", "3. Olgunluk Çemberi" olarak isimlendirilir.

Çemberler metodunda ana amaç; gelişimsel bir yöntem kullanarak, yaratıcılığı ve hızlı reaksiyon becerisini zirveye çıkartmak amacıyla düzenli çalışma yöntemini geriye itmiş olan bir firmanın hem geliştirdiği avantajları korumasını, hem de yaşadığı dezavantajları adım adım düzeltmesini sağlamaktır.

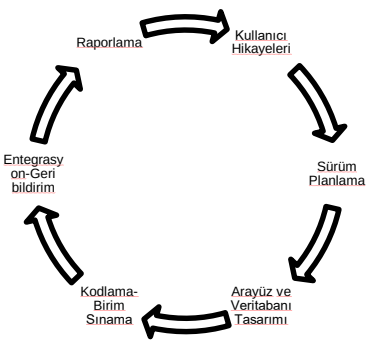
Genel olarak "Çevik Manifesto"dan yola çıkarak kurulan temel çatı, geliştirici ekibinin CMMI değerlendirmesine uygun olarak gelişimini esas almaktadır. Bir başka deyişle, ulaşılacak nokta olarak, firmanın CMMI derecelendirmesinde üst basamaklara tırmanmasını planlamaktadır. Bu amaçla, ekip üyelerine farklı çemberlerde farklı roller tanımlanmış ve farklı seviyelerde raporlama ihtiyaçları oluşturularak firmanın olgunluk seviyesi arttırılmaya çalışılmıştır.

3.2.1. Başlangıç Çemberi

Başlangıç Çemberi çalışmalarına yeni başlamış firmalar için anlamlı olacaktır. Çalışmalarına yeni başlamış veya uzun süredir çalışmakta olmasına rağmen olgunluğunu ve dayanıklılığını arttırma çabasındaki firmalar bu çemberin hedef kitlesinde bulunmaktadır. Özellikle yeni başlamış ve olgunluk seviyesi düşük firmalar incelendiğinde avantajlı yönler olarak firma çalışanlarının çözüm odaklı davranabilmeleri, müşteri ihtiyaçlarındaki değişikliklere (veya ihtiyaç tanımlama sürecinde yapılan hataların

düzeltilmesine) en başarılı çözümü hızla üretebilmeleri, iş bölümünü hızla değiştirebilmeleri gösterilebilir. Özellikle "hız" ile değerlendirilebilecek bu avantajlı yönlere karşılık olarak da, personel değişikliklerine yönelik kırılganlıkları, proje parametrelerindeki değer artışlarında (personel sayısı, proje sayısı, farklı lokasyon sayısı, vb.) bocalamaları ve gelişmelere karşı tutunamama riskleri önemli dezavantajlar olarak değerlendirilebilir ki, bu başlıkları da "olgunluk" terimi altında değerlendirebiliriz.

Başlangıç çemberinin ilerleyişindeki amaç da, "hız" sağlayan düzensizliğin gelişim karşısında çıkardığı sorunların en az kayıp ile çıkarılabilecek en yüksek "olgunluk" seviyesine çıkarmaktır. Burada yapılması planlanan; geliştiricilerin sahaya yakın olmaktan gelen becerilerini düzenli yönetime yaklaştırırken çalışmalarını da sekteye uğratmamaktır.



Şekil 1. Başlangıç Çemberi

Şekilde de görüldüğü gibi, Başlangıç Çemberi sadece 6 adımdan oluşur. Bu adımlar temel yazılım geliştirme sürecinde, en hızlı ve alan tecrübesine sahip ekiplerin hali hazırda kullandıkları adımları tanımlamaktadır. Fark olarak sadece adımların isimlendirilmesi ve son adımda bulunan "Raporlama" çalışması bulunmaktadır.

İlk adım; "Kullanıcı Hikayeleri" tanımlama yoluyla yapılan Analiz sürecidir. Başta "Ekstrem Programlama" olmak üzere, "Çevik Yazılım Geliştirme Modelleri"nin analiz çalışmalarında kullanılan "Kullanıcı Hikayeleri", kullanıcılarının ürün üzerine fikirlerini en kısa haliyle, her seferinde tek fonksiyonu kısaca açıklayacak şekilde yazmaları ile oluşturulur. Yazılan "Kullanıcı Hikayeleri", temizlenme ve düzenlenme sürecinde numaralandırılır ve önceliklendirilir. Bu sayede kullanıcıya en yakın haliyle ihtiyaçların tespidi ve kayıt altına alınması sağlanır.

İkinci Adım; "Versiyonlama" ile yapılan Belirtimleme ve Planlama sürecidir. Hazırlanan Kullanıcı Hikayeleri geliştirme ve kullanım önemine uygun olarak sıralanır. Yaklaşık 2 hafta ile 1 ay içerisinde tamamlanacak versiyonların iteratif geliştirmeye uygun olarak kullanıcıya teslim edilmesine dayanacak versiyonlama ile proje sürecine dair planlamaların etkinliği ve başarımının arttırılması amaçlanmaktadır.

Üçüncü adım; "Arayüz-Veritabanı Tasarımı" ile tasarımın temellerin oluşturulması şeklindedir. Tasarım adımına pratikte aşina olmayan ekibin yazılımın altyapısının temelini oluşturan veritabanını ve ön yüzünün (ve fonksiyonel ihtiyaçlarının giderilmesinin) temeli olan arayüzünün tasarımı ile bu konularda incelemenin yapılabilmesi amaçlanmaktadır. Ayrıca bu sayede en temel noktalarda çözümün arşivlenmesi de sağlanmış olacaktır.

Dördüncü adım; "Kodlama – Birim Testleri" ile genel tanımı yapılmış olan ürünün modüllerinin geliştirilmesi beklenmektedir. Hali hazırda kapsamlı bir ürün geliştirilmediği varsayımıyla (Ürün kapsamının genişlemesi veya ekibin büyümesi durumunda firmanın zaten düzensiz bir yapıda başarılı sonuçlar almış olması beklenmeyecektir.) ürünün ilk tasarımına uygun olarak modüllerinin kodlanması, kodlama sürecinde gerekli görülebilecek detaylı ihtiyaç tanımlamasının yapılması ve modüllerin birim olarak sınamalarının yapılması planlanmaktadır. Bu adımlarda tutulan notlar sonraki adımlarda raporlama amacıyla da kullanılacağından planlı bir uyarı olmasa da, mümkün olan en düzenli şekilde saklanması faydalıdır.

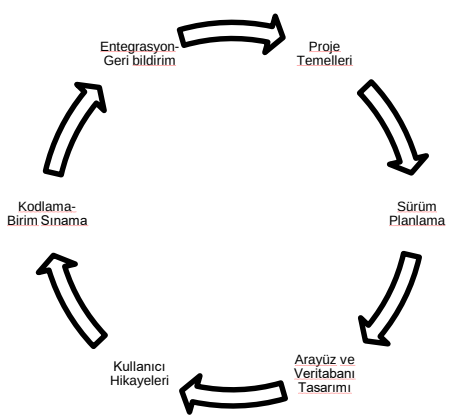
Beşinci adım; "Entegrasyon – Geri bildirim" ile geliştirilen modüllerin bir araya getirilmesi ve onaylama amaçlı olarak kullanıcı geri bildirimlerinin alınması planlanmaktadır. Dördüncü ve Beşinci adımlar arasında geri dönüşlü bir yapının olması ile entegrasyon veya onaylama sınamaları sırasında kodlama ile düzeltilebilecek hataların farkedilmesi durumunda bu adıma dönüş yapılarak düzeltme yapılması amaçlanmıştır. (Yaşanan sorunun kodlama aşamasında düzeltilememesi durumunda bir önceki adım olan "Arayüz – Veritabanı Tasarımı" adımına da dönüş sağlanabilir.)

Başlangıç Çemberi'nin altıncı ve son adımında ise; ürünü geliştirmeye yönelik yapılan çalışmaların raporlanması amaçlanmaktadır. Bu aşamada standart bir raporlama formatı veya tanımlaması bulunmasa da, elde edilen "Kullanıcı Hikayeleri"nin temizlendikten sonra raporlanması ile, geliştirme sürecinde tutulmuş olan notların en az efor ile raporlanması beklenmektedir.

Başlangıç Çemberi dahilinde, ekibin düzensiz ancak başarılı çalışma sistematiğini korurken, küçük eklentilerle, hali hazırda yürütülmekte olan adımların formülize edilmesi amaçlanmaktadır. Bu sayede kişiye bağımlılıklarda azalma ve ekibin çalışma düzenini çok değiştirmeden ön görülebilir ve basit raporlanabilir hale gelmesi beklenir. Bu noktada elde edilecek ilk faydalar; çalışmaların raporlanması ve düzenli yazılım geliştirmeye yumuşak geçişin sağlanmasıdır.

3.2.2. Gelişme Çemberi

Başlangıç Çemberi'nde bir süre çalışan geliştirici grubunun, çalışmalarını daha düzenli bir şekle getirmesi, daha kapsamlı projelerde yer alması veya ekibin büyümesi durumlarında Gelişme Çemberi'ne geçişi tavsiye edilir. Bu noktada objektif bir değerlendirme kriteri olmamakla birlikte, yöneticinin sürece uyumluluk ve ihtiyaçlar konusundaki subjektif değerlendirmesi önemli olacaktır.



Şekil 2. Gelişim Çemberi

Gelişme Çemberi olarak değerlendirebileceğimiz "Orta Çember", "Proje Temelleri"nin değerlendirildiği adım ile başlar. Proje üzerine yapılan ilk incelemeler ve önceki tecrübelerin değerlendirildiği adımda, projenin ana fonksiyonları ile genel tasarım ve veritabanı analizine temel oluşturacak bilgilerin toplanması amaçlanır.

İkinci Adımda, alınan ilk bilgilere ve projelendirme tecrübesine uygun olarak "Sürüm Planlaması" yapılır. Projenin belirlenen fonksiyonları üzerinden kullanıcı ile iletişimi somut örnekler ve geri bildirimlere dayandıracak şekilde bir sürüm planlaması oluşturulur. Bu noktada proje ekibinin zaman planlaması ve kullanıcı ile işbirliği konularında kendini geliştirmesi amaçlanır.

Üçüncü Adımda, "Temel Arayüz – Veritabanı Tasarımı" gerçekleştirilecektir. Belirlenen fonksiyonel ihtiyaçlara göre projenin modüler geliştirmeye uygun hazırlanması amacıyla ortak noktalar olacak Arayüz taslakları (İng. Mockup) ve Veritabanı tasarımları tanımlanır. Burada ana amaçlar, geliştiricilerin proje fonksiyonlarını daha kolay anlamalarını ve modüler geliştirme sırasındaki iletişimi sağlamaktır.

Dördüncü Adımda, "Kullanıcı Hikayeleri" alınması planlanmaktadır. Burada alt sürümlere bölünmüş, modüler tasarlanmış ürünün her modülü için kullanıcı hikayeleri formatında bilgi toplanması ve detaylı analiz yapılması düşünülmektedir. Bu sayede hem modüllerin çalışmasına yönelik detaylı inceleme yapılacak, hem de geliştiricilerin yüksek seviyeli yönetim çalışmalarına girmeleri engellenecektir. Bu sayede geliştiricilerin yeni sisteme adaptasyonunun hızlı gerçekleşmesi beklenmektedir.

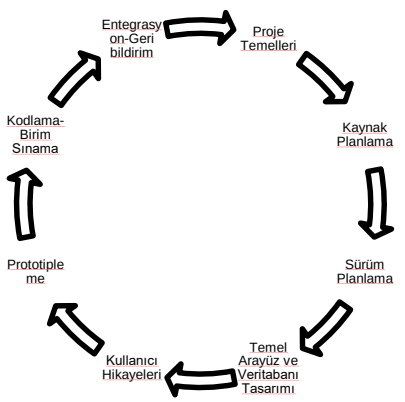
Beşinci ve Altıncı Adımlarda, "Kodlama – Birim Testleri" ve "Entegrasyon – Geri bildirim" çalışmaları yapılacaktır. Burada farklı modüller üzerinde çalışacak ekiplerin dört, beş ve altıncı adımları yinelemeli çalışmaları düşünülmüştür. Her geliştirilen modül, ikinci adımda belirtilen sürüm planına uygun olarak ürüne entegre edilir ve kullanıcıdan geri bildirim alınmasına çalışılır. Elde edilen geri bildirimin değişikliğe neden olduğu durumlarda önceki adımlara da geri dönülebilir. Ancak değişiklik ihtiyacı oluşmazsa tanımlı modüller, daha önceden belirlenmiş sıraya göre geliştirilerek ürünün oluşturulması sağlanır.

Gelişme Çemberi bu şekilde sona ermektedir. Burada raporlama adı altında bir başlık olmamasının ana nedeni, bir önceki çembere göre daha olgun olan ekibin adımlarda tanımlanmış rapor ihtiyaçlarını sağlayabilecek durumda olmasıdır. Ayrıca model çerçevesinde

iş tanımı raporlama olan bir yazılım geliştiricinin de ekibe eklenmesi faydalı olabilecektir. Başlangıç Çemberi'nden geçiş sürecinde ihtiyaç olan ve çemberin altıncı adımında hazırlanmış olan raporlar ilgili adımlarda yerleştirilerek, düzenli raporlamaya yaklaşılması planlanır. Bu nedenle "Gelişme Çemberi" içerisindeki ekibin, zaman planlaması konusunda becerilerini arttırması, daha kapsamlı ürünleri projelendirebilmesi ve daha düzenli raporlama yapabilmesi amaçlanır.

3.2.3. Olgunluk Çemberi

Olgunluk Çemberi olarak adlandırabileceğimiz dış çember, yazılım geliştirme ekibinin en düzenli hale geldiğine işaret eder. Bu aşamaya gelen ekip, başlangıç anında daha kapsamlı projeler üzerinde çalışırken, geniş bir ekiple ve farklı projeleri aynı zaman aralığında geliştirebilir durumda da olabilecektir. Bu nedenle "Olgunluk Çemberi" seviyesinde hem geliştirilen yazılım projesinin çalışmasına yönelik adımlar tanımlanmakta, hem de genel olarak proje ekibinin yönetimine yönelik çalışmalar bulunmaktadır.



Şekil-3. Olgunluk Çemberi

Sürekli gelişme ile olgunlaşmış ekibin en yüksek seviyesi olarak düşündüğümüz dış çember, "Gelişme Çemberi"ne benzer şekilde, "Proje Temelleri"nin tanımlanması ile başlar. Önceki süreçten tecrübeli olan ekip ve yönetim, projenin ilk analizini yaparak ihtiyaç listesine yönelik altyapıyı hazırlar.

İkinci adım olarak, "Kaynak Planlaması" öngörülmektedir. Bu noktada gerek birden çok kaynağa ve ihtiyacı olan projeye sahip yönetimin proje ön analizine dayanarak kaynaklarını planlaması, gerekse proje sürecini planlama yoluyla şeffaflaştırması amaçlanmaktadır. Önceki adımlarda, hem proje yönetimi, hem de geliştirici kaynaklarının değerlendirilmesi konularında tecrübe kazanmış ekibin bu çalışmayı yapabilmesi öngörülmektedir.

Üçüncü, Dördüncü ve Beşinci adımlar, yine orta çember ile aynı amaçla ve sırayla "Sürüm Planlaması", "Temel Arayüz – Veritabanı Tasarımı" ve "Kullanıcı Hikayeleri" olarak tanımlanmıştır.

Altıncı Adım olarak tanımlanmış olan "Prototipleme" kullanıcı ihtiyaçlarının doğrulanmasına yönelik olarak yazılmıştır. Hızlı prototipleme sayesinde, kullanıcı hikayelerinde elde edilen ihtiyaçların incelenmesi ve sınanması sağlanacaktır. Burada ana amaç; analiz sonuçlarının kontrol edilmesi ile entegrasyon sonrası geri bildirim sürecinin ve buna bağlı geri dönüşlerin kısaltılmasıdır. Kontrol ve raporlama konusunda tecrübesi ve alışkanlığı artmış olan, olgun ekibin bu aşamaya sorunsuz adaptasyonu beklenmektedir.

Yedinci ve Sekizinci Adımlar da, gelişme çemberi'nde belirtilen aynı adlı adımlarla aynı amaçla çalışmaktadır.

Olgunluk Süreci olarak adlandırılabilecek dış çembere uyum sağlayan ekibin, olgunluğunu arttırmak suretiyle daha planlı, düzenli ve ekip üyelerindeki değişikliklere daha dirençli olması beklenmektedir.

4. Tartışma

Metod; yazılım geliştirme süreçlerinin önemli sorunlarından birisi olan adaptasyon sorununu adreslemektedir. Özellikle güncel teknolojiler ile ürün geliştirmek zorunda olan ve hızlı değişen kullanıcı ihtiyaçları ile karşılaşan ekiplerde, ekibin yönetim ve denetim amacıyla kullanılacak zaman ve mali kaynaklarının da düşük olması durumunda önemli bir sorun olarak karşımıza çıkan "ekibin yönetilebilirlik ve ürünün bakım yapılabilirlik özelliklerine" olumlu etkileri bulunan Yazılım Geliştirme Metodolojilerinde adaptasyon önemli bir sorundur. Zira bu noktada "Hız" – "Bütçe" – "Ürün Kalitesi" ve benzeri özellikler ile "Seçili Sürece Adaptasyon" arasında bir seçim yapılması gerekmektedir. Bu nedenle çoğu kuruluşta ekibin büyümesinde önemli sıkıntılar yaşanmakta, başarılı ve küçük bir ekipten başarılı ve tecrübeli bir ekibe geçişte bir çok yazılım firmasının süreci yönetemediği görülmektedir. (Temel çözüm olarak; ekibin

büyütülmemesi veya önemli kaynaklar harcanarak tüm sistemin baştan tasarlanması ve uygulamaya konması denenmektedir ki iki durum da istenen sonuçları üretmekten uzaktır.) Bu nedenle adaptasyon süreci daha az yıkıcı olan süreçler geliştirilmiştir.

5. Sonuç

Yazılım Geliştirme Modelleri incelendiğinde, farklı durum ve ihtiyaçlara uygun olarak modellerin de bir evrim sürecinden geçtiği görülmektedir. Bu da temel yazılım mühendisliği gelişim fikrinde yer alan "Sorunların çözümünde kullanılan yöntemin oluşması için yazılım geliştirmenin doğasının anlaşılması gerekliliği" [12] ile açıklanabilir.

Temel mühendislik fikirleri ile tanımlanmaya çalışılmış olan yazılım geliştirme süreçlerinin, daha faydalı olması ve daha kaliteli sonuçların üretiminde kullanılması için farklı durumlar için kullanılması ve değerlendirmelerinin yapılarak evrimleşmesi gerekmektedir.

Çalışmada tanımlanmış olan metodoloji vasıtasıyla da, gelişim sürecine hızlı yazılım geliştirme ve geliştiricilere bağımlı olarak başlayan yazılım firmalarına bir öneri yapılmaktadır. Tanımlanan sürece uygun olarak gelişimini planlayacak firmaların da, geçiş adımlarındaki yeterlilikleri sağlamaları durumunda firma kırılganlıkların azalması ve olgunluklarının artması beklenmektedir. Bunun sonucunda da daha başarılı (planlı zamana / bütçeye uygun, tanımlanmış ihtiyaç listesinin hatasız olarak yerine getiren bir ürüne yönelik) bir yazılım geliştirme sürecinin planlanması ve uygulanması sağlanabilir.

Kaynakça

[1] Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J., Agile Software Development Methods: Review and Analysis, VTT Publication, 2002

[2] Akbar, M., Statistical Analysis of the Effects of Heavyweight and Lightweight Methodologies on the Six-Pointed Star Model, IEEE Access, 2018

[3] Boehm, B., A Spiral Model of Software Development and Enhancement, ACM SIGSOFT Software Engineering Notes, 11, 4, 14-25, 1988

[4] Davis, A.M., Bersoff, E. H. and Comer, E.R., A Strategy for Comparing Alternative Software Development Life Cycle Models, IEEE Transactions on Software Engineering, 14, 10, 1453-1462, 1988

[5] Fowler, M., 2000, https://www.martinfowler.com/articles/ new MethodologyOriginal.html (21.06.2000)

[6] Papazoglou, M.P. and Van Den Heuvel, W., Life Cycle Methodology, Communications of the ACM, 50, 10, 79-86, 2007

[7] Paulks, M.C., Curtis, B., Chrissis, M. B. and Weber C.V., Capability Maturity Model v1.1, IEEE Software, 0740, 93, 18-28, 1993

[8] Rajlich, V.T., Bennett, K. H., A Staged Model for Software Life Cycle, IEEE Computer, 0018, 9162, 66-72, 2000

[9] Royce, W., Managing the Development of Large Software Systems, Proceedings, IEEE WESCON, 1-9, 1970

[10] Shipra, R., Sharma, R., Gupta, K., Strategies for web application development methodologies, 2016 ICCCA, 2016

[11] Silva, F., Soares, F., Peres, A., Azevedo, I., Vasconcelos, A., Kamei, F., Meira, S., Using CMMI together with agile software development, A Systematic Review, Elsevier Information and Software Technology, 2015

[12] Sommerville, I, Software Engineering, Pearson, 2005

[13] Tripp, J, Armstrong, D., Agile Methodologies: Organizational Adoption Motives, Tailoring and Performance, Journal of Computer Information Systems, 58:2, 2016

[14] Quelal, R., Villavicencio, M., Mendoza, L., A Survey of Agile Software Development Methodologies in Ecuador, 2018CISTI, 2018

[15] Vijayasarathy, L. and Butler, C., Choice of Software Development Methodologies, IEEE Software, 0740-7459, 86-95 2016