Yöntem Çevik Olunca

Selma SÜLOĞLU

Datasel Bilgi Sistemleri A.Ş., Ankara, Türkiye

e-posta: ssuloglu@yahoo.com

Özet

Eskiden "hafif siklet" adı verilen çevik yöntemler müşteri odaklı yazılım ihtiyaçlarının belirginleştirilmesi ve küçük adımlarla gerçekleştirilmesi temeline dayanan bir yaklaşım izlemektedir. Türkiye'de çevik yöntemler geleneksel yöntemlere nazaran daha az bilinmekte ve yazılım geliştirimde yeni yeni uygunlanmaya başlamaktadır. Bu makale, çevik yöntemler hakkında bilgi verilmesi, geleneksel ve çevik yöntemlerin birlikte kullanılabilirliğinin ve hangi durumlarda uygulanmalarının uygun olduğunun belirlenmesi amacıyla yapılan literatür araştırmasını içermektedir.

Abstract

Formerly known as light weight, Agile Methodologies follows an approach based on the principal that makes software requirements clear with customer oriented labor and developes software with small iterations and increments. In Turkey, in contrast with the awareness of traditional methodologies, the knowledge of agile methodologies is insufficient and they have been used recently during software development. This paper includes a literature search that aims to give information about agile methodologies, to determine the suitability of the usage of traditional and agile methodologies together and to decide in which condition they can be applied.

Anahtar Kelimeler

Çevik Yöntemler, Geleneksel Yöntemler, Karma Yaklaşım, Ekstrem programlama, Scrum, Kristal, Açık Kaynak Geliştirimi, Dinamik Sistem Geliştirimi, Özellik Güdümlü Geliştirim, Uyarlanabilir Yazılım Geliştirimi

1 . Giriş

En çok bilinenleri çağlayan(waterfall), prototipleme(prototyping), artırımlı(incremental), spiral(spiral) ve hızlı uygulama geliştirimi(Rapid Application Development) olmak üzere geleneksel yöntemler planlama ve dökümantasyona verdiği önemle öne çıkarlar. Her bir yöntemin stratejisi farklı olmakla birlikte yazılım geliştirim süreçleri analiz, tasarım, kodlama, test ve onarım sırasını izler. Yöntemlerin esnek olmayışı, yapılacak her işin kesin tanımlı bir sırasının oluşu, kimlerin hangi işlerden sorumlu olduğunun kesin ve katı çizgilerle belirlenmiş olması ve geliştirim sırasında grup içi ve müşteri ile iletişimin yeterli olmayışı geleneksel yöntemlerin en önemli sorunlarındandır.

Yazılım geliştirim süreçlerine farklı bir yorum getiren çevik yöntemler ise dünyada küçük, orta ve büyük ölçekli bir çok şirkette ve çeşitli endüstri alanlarında sekiz yıldır uygulanmaktadır. Bunlar ağır, dokümantasyona dayalı ve gereksinim değişikliklerine hızlı cevap veremeyen yöntemlere tepki olarak yazılım endüstrisinin içindeki tecrübeli insanlar tarafından geleneksel yöntemlerin yaralarını sarmak için geliştirilmiştir.

Yazılım süreçleri açısından ele alındığında geleneksel yöntemleri savunanlar ile çevik yöntemleri savunanların iki kutba ayrıldığı görülmektedir. Burada önemli olan savaşı kimin kazanacağından çok kullanılan yöntemlerin hangi tür projelerde ve organizasyonlarda başarılı olduklarının belirlenmesi ve buna uygun olarak kullanılmasıdır.

2 . Çevik Yöntemler

Çevik yöntemlerin en önemlileri Arık[1] Geliştirme (AG), Uyarlanabilir Yazılım Geliştirme (UYG), Scrum, Ekstrem Programlama (EP), Kristal Yöntemleri, Özellik Güdümlü Geliştirme (ÖGG), Açık Kaynak Kod Geliştirme (AKKG) ve Dinamik Sistem Geliştirme (DSG) yöntemleridir [1]. Bu yöntemlerden Arık Geliştirme hariç diğerleri

- Süreç ve geliştirme araçlarının yerine *bireyler ve bireyler arası ilişkileri*
- Detaylı dokümantasyon yerine *koşan yazılımı*
- Sözleşme görüşmeleri yerine *müşteri işbirliğini*
- Plan takip etmek yerine *değişikliklere ayak uydurmayı*

tercih etmektedir. Yukarıdaki ortak görüş Ç*evik Yöntem Geliştirme Bildirisi*'nde yer almakta, Agile Alliance (Çevik Birliği) tarafından desteklenmekte ve etkili yazılım geliştirme için anahtar kabul edilmektedir [1, 2].

Birbirinden farklı çevik yöntemler birçok özelliği paylaşsa da aralarında önemli farklılıklar vardır. Bazıları proje yönetimi ve işbirliği içinde çalışmaya önem verirken (UYG, Scrum, ve DSG), diğerleri örneğin Ekstrem Programlama yazılım geliştirme çalışmalarına odaklanmıştır [1]. Bu yöntemlerden her biri önemli noktalarına değinilerek aşağıda anlatılmaktadır.

2.1 Ekstrem Programlama (eXtreme Programming)

Ekstrem Programlama;

- Basitlik, haberleşme, geri bildirim ve cesaret temelleri üzerine kurulmuş olan
- Test yapma, kodlama, dinleme ve tasarım yapma aktivitelerinden oluşan
- Hızlı geri bildirim, basitlik, değişimi benimseyen, artımlı (incremental) değişim ve kaliteli iş prensipleri ile çalışan
- Aktiviteleri başarıyla gerçekleştirmek için kullanılan 12 pratik içeren (planlama oyunu, küçük içerikli sürümler, benzetme, basit tasarım, yeniden düzenleme, önce test geliştirme, ikili programlama, kolektif ortaklık, sürekli entegrasyon, haftada 40 saat çalışma, yerinde müşteri ve kodlama standartları)

[1] Arık kelimesinin sözlük karşılığı zayıf, kuru, sıska olup ve burada gereksiz şeylerden arındırılmış anlamıyla kullanılmaktadır.

- Gerçek hayatta yukarıda belirtilen uygulamaları başarıyla gerçekleştirebilmek için tanımlanmış stratejilerden oluşan bir yazılım geliştirme yöntemidir [3, 4, 5].

EP ile yazılım geliştirme sürecine projenin yinelemelere[2] (iteration) bölünmesiyle başlanır. Her bir yineleme içinde müşteri ile birlikte yapılan toplantılarda gereksinimler belirlenir ve planlama yapılır. Geliştirme süresince programcılar müşterileri ve çalışma arkadaşları ile devamlı iletişim halindedir. Kodlama standartlarına uygun olarak yazılan kod programcılar arası iletişimin önemli bir parçasını oluşturur. Kodlama süresince sadece planlanan özelliklerin gerçekleştirilmesine odaklanılır ve gerektiğinde kodu sadeleştirmek için sistem tasarımında değişiklikler yapılabilir.Test sürecine yönelik olarak üretilen kodlar ilk günden başlanarak test edilir. Projenin durumu müşteri değerlendirmeleri, takım üyelerinin görüşleri ve test sonuçları ışığında her gün yapılan değerlendirme toplantıları ile izlenir. Sistem mümkün olan en kısa zamanda çalışır hale getirilip müşteriye teslim edildikten sonra müşteriden gelecek isteklere göre değişiklikler planlanır ve uygulanır [3, 4]. EP yazılım geliştirimindeki sürece önem verir. Bu yöntem gereksinimlerin projenin başında tam ve net olarak tanımlanamadığı, gereksinimlerin dinamik olarak değiştiği ve kısa sürede çalışan bir program ortaya çıkartmayı gerektiren projelerde kullanılabilir.

2.2 Scrum

Scrum herhangi bir ürün geliştirmek ya da bir işi yönetmek için kullanılan artımlı ve yenilemeli bir süreçtir. Projenin başında uygulanabileceği gibi ortasında yada üretimin problemli olduğu zamanlarda da uygulanabilmektedir. Sadece yazılım geliştiriminde değil finansal ve medikal ürünlerin üretilmesinde de kullanılmaktadır. [6] Scrum yönteminde planlama, görev tanımları ve dökümantasyona daha az zaman harcanarak proje takımının değişen koşullara uyumlu, esnek bir sistemi nasıl geliştirecekleri üzerine yoğunlaşılır. Bu yöntem bayrak teslimi(sprint) planlama, bayrak tesliminde yapılacak işler listesini(sprint backlog) belirleme ve scrum toplantılarını içerir. Kodlama safhası için yönteme özgü bir yazılım geliştirim tekniği tanımlanmamıştır. Genel olarak 5-10 kişiden oluşan scrum takımları kendi içinde organize olmuşlardır. 4 haftalık bayrak teslimleri sonucu gözle görülür işlevselliğe sahip bir ürün geliştirilir. Takım projenin başında tüm gereksinimleri içeren *ürün yapılacak işler(product backlog)* listesini çıkarır. Takım *ürün yapılacak işler* listesinden öncelikli olduklarına karar verdikleri işleri seçerek her bayrak tesliminde yapılması gerekenleri belirler ve gerektiğinde listeyi günceller. Buradaki önemli nokta gereksinimlerin bayrak teslimi içinde değişmez olduğudur. Her yinelemede analiz, tasarım, geliştirme ve test süreçleri uygulanır. Her gün geliştirme takımı ile Scrum olarak adlandırılan 15 dakikalık toplantılar yapılarak yapılan işler ve işlerin gidişatı konuşularak yeni eklenen özellikler tartışılır. Bu toplantılar problem çözümünü içermez. Takımlar müşteriyi de içine alacak bir yapıda etkili bir biçimde en az sayıda doküman üretir ve kod teslimine odaklanır. [6, 7, 8, 9]

Scrum, yönetimi ön plana çıkaran bir yöntem olarak daha çok yinelemeli planlama ve izleme süreçlerine önem verir. Tahmin edilemeyen, gitttikçe karmaşıklaşan iş alanlarında, projenin esnekliğini ve kalitesini arttırmak istediğimiz geliştirme projelerinde kullanılabilir [10]. Bazı açılardan diğer çevik yöntemlere benzer ve özellikle EP 'nin kodlama pratikleriyle iyi uyum sağlar [11]. Örneğin, Primavera şirketi 2002'de yaptığı projesinde Scrum ile EP yöntemlerini bir arada kullanmış, yönetim ve planlama sorunlarını Scrum ile çözerken etkin takım çalışmasını ve daha kaliteli ürün geliştirmeyi EP ile başarmıştır [12].

[2] *Yineleme* tekrarlama, birbiri ardına belirli bir işi yapma anlamında kullanılmaktadır.

2.3 Kristal (Crystal)

Farklı projelerin farklı yöntemlere ihtiyaç duyduğu düşüncesinden yola çıkılarak Kristal ailesi yöntemleri (Crystal Clear,Cryristal Orange ve Crystal Orange/Web) ortaya atılmıştır. Bu yöntemler projelerdeki çeşitliliğe projedeki kişi sayısı ve hata sonuçları olmak üzere iki farklı açıdan yaklaşmakta ve her yöntem Konfor(*Comfort*), İsteğe Bağlı Kapital(*Discretionary Money*), Gerekli Kapital (*Essential Money*) ve Hayat Noktası (*Life*) olmak üzere 4 farklı seviye içermektedir. [7, 11, 13] Crysal Clear 6 kişilik küçük gruplar için, Crystal Orange 10 ila 40 kişilik geliştirme grupları ile 1-2 sene sürecek orta büyüklükteki projeler için kullanılır ve birden fazla grup içerir. Crystal Orange/Web ise web tabanlı projelerin geliştirilmesinde kullanılır. [7,14] Kristal kendi kendine adapte olabilen, iletişim odaklı ve insan gücüne önem veren yöntemler topluluğudur. Bu yöntemler aşağıdaki varsayımları temel alır [15]:

1. Her proje biraz farklı birtakım kural, uyum yada yönteme ihtiyaç duyar.
2. Proje çalışmaları insan ile ilgili konularda hassastır ve bu konuların gelişimine bağlı olarak daha da gelişir; insanlar daha iyi oldukça, grup içi çalışmalar daha da iyi olur.
3. Etkin iletişim ve sık yapılan teslimler orta seviye iş ürünlerine gereksinimi azaltır.

Kristal yöntemlerinin iki kuralı vardır: (1) 1 ila 3 ay arasında değişen en fazla 4 ay süren artımlarla geliştirim yapılır. (2) Her bir artım öncesi ve sonrası başarılar ve başarısızlıkların dile getirildiği fikir seminerleri düzenlenir [14]. Kristal yöntemlerinden her biri yöntemin ağırlığına göre beyaz, sarı, turuncu, kırmızı renklerinden biri ile gösterilir. Renk ne kadar koyu ise yöntem o kadar ağırdır. Projenin büyüklüğü ve kritikliğine bakılarak uygun olan renkteki yöntem seçilir. Takım, artımda kodlanacak gereksinimleri ve iş takvimini belirler. Her artım kodlamayı içeren yapılandırma, gösterim ve yeniden gözden geçirmeleri içerir. Süreç takımın teslim ettiği ürünlere ve zamana bağlı duruma göre izlenir ve takip edilir. [7] Takım kendine ait bir süreç oluşturmak için kendi kendini kontrol eder. Aynı zamanda sürecin kendi kendine gelişimini sağlayacak yineleme sonu incelemelerine ağırlık verilir. [11] Yazılım geliştirme süreci belirtilmediğinden XP ve Scrum ile entegre kullanılabilir. Çok kritik projelerde kullanıma çok uygun değildir, aynı ofiste çalışan geliştiricilerle iletişimin fazla olduğu küçük ve orta ölçekte projelerde kullanılabilir [7].

2.4 Açık Kaynak Geliştirimi (Open Source Development)

İnternet'le birlikte çok sayıda katılımcı ile açık kaynak kodlu yazılım geliştirme yaygınlaşmıştır. Yaygınlaşma süreci içinde InterNet News Server, Mozilla Web Server, ApacheWebServer, Linux işletim sistemi, Perl programlama dili gibi başarılı projeler ortaya çıkmış ve açık kaynak geliştirimine ilgi artmıştır. AKG ile şirketler projelerinin yada geliştirilen ürün sürümlerinin dünya çapında kullanılmasını sağlamaktadırlar. Hatta rafa kaldırılmış bazı projeler AKG ile geliştirilmeye devam edilebilir ve bu sayede ürünün fonksiyonalitesi, kalitesi ve kullanımı arttırılabilmektedir. Aynı zamanda birbirinden farklı donanımlarda ve durumlarda kullanıcı testinin yapılmasına olanak verdiğinden sorunların belirlenmesinde ve değişik çözüm önerilerinin geliştirilmesinde yarar sağlamaktadır. Kullanıcılar, dünyanın her yanından açık kaynak kodları görebilmekte, değiştirebilmekte ve lisans kapsamında projelerinde kullanıp geliştirdikleri kodu kapatabilmektedirler. Kullanıcı direk geliştiricilerle iletişim kurup kısa sürede teknik destek alabilmektedir. Bu hem maliyet hem iş gücü tasarrufunu sağlamakta, yazılım geliştirme sürecini hızlandırmakta ve yazılım geliştiriminde tekrar kullanımını arttırarak yazılım dünyasında genel gelişimi ve yetkinleşmeyi sağlamaktadır. Bu yöntem problem tanımı, değişikliği kodlayacak gönüllülerin bulunması, çözümün tanımı, kod geliştirme ve test, değişikliğin gözden geçirilmesi, kod teslimi ve dokümantasyon ile sürüm yönetimi safhalarını içerir. Açık kaynak kodlu geliştirim

toplulukları proje lideri, gönüllü-kıdemli geliştiriciler, kod değişikliğine onay veren ikinci derece geliştiriciler, bakımdan sorumlu kişiler ve testleri yapıp, hataları bildiren veya kodlama yapmayıp sadece haber grubunu takip eden kullanıcıları içerir. [7, 11]

2.5 Uyarlanabilir Yazılım Geliştirimi (Adaptive Software Development)

Hızla değişen ve karmaşıklaşan yazılım piyasasında yarışta başarı sağlamak için UYG'nin felsefesi değişimle savaşmak yerine, değişimle çalışmaktır. Bunu gerçekleştirebilmek için değişimleri kapsayan ve değişimlere cevap veren, uyarlanabilen uygulamalar ile çevik ve kolay adapte olan insanlara, takımlara ve organizasyonlara ihtiyaç duyar. Sürece odaklanmaktan çok işbirliğiyle ortaya çıkarılan ürüne odaklanmak temel strartejisidir. UYG'ye göre, tahmin edilemeyen bir ortamda belirsizliklerle başa çıkabilmek için insanların güçlü bir işbirliğine ihtiyaçları vardır. Bu süreç boyunca yönetim insanlara neyi yapacağını söylemek yerine iletişimi güçlendirerek insanların yaratıcı çözümler bulmasını sağlamaya dikkatini vermelidir. Bu yöntemde (1) Projenin vizyonunu, verilerini ve tanımlarını içeren ortak dokümantasyon tutulur. (2) Oturumlar, müşteri odaklı gruplar ve süreç iyileştirme ve geliştirme toplantıları gibi işbirliği teknikleri uygulanır. (3) İşbirliği tekniklerinin planlanmasından sorumlu bir organizatör tanımlanır. UYG özellikle sürekli öğrenme ve işbirliği çalışmaları üzerinde durur, yenilemeli ve artımlı bir süreci destekler. [14] Yenilemeli geliştirim, özellik-bileşen tabanlı planlama ve müşteri odaklı grup toplantıları önemli uygulamalarıdır. UYG'nin merkezinde 3 safha vardır: spekülasyon, işbirliği ve öğrenme. Spekülasyon safhasında proje amacı ve geliştirimde gerekecek bilgiler belirlenir ve uyarlanabilir döngü planlaması yapılır. İşbirliği safhasında eş zamanlı bileşen geliştirimi yapılır. Öğrenme safhasında kalite toplantıları ve özellikle proje başında müşteri ve geliştiricilerin istenen sistem özelliklerinin tartışarak iletişimi arttırdıkları ortak uygulama toplantıları yapılır. Sürecin nasıl geçtiğinden çok ne öğrenildiği önemlidir. UYG takım yapısını ayrıntılı olarak belirtmemiştir ve yöntem hakkında yol gosterici yönerge çok fazla bulunmamaktadır. [1, 5, 7].

2.6 Özellik Güdümlü Geliştirme (Feature Driven Development)

ÖGG model tabanlı, küçük yenilemelerden oluşan bir süreçtir. Planlama, öncelik belirleme, tasarım ve geliştirme süreçlerinde özellik(feature) adı verilen müşteri için anlamlı, küçük ve işlevsel gereksinim parçalarını kullanılarak geliştirme yapılır. Artımlı olmayıp dinamik olan süreç, özelliklerin gerçekleştiriminin müşteri tarafından değerlendirilmesi aracılığıyla, artıma gerek kalmadan, küçük yinelemelerle geri bildirim almayı amaçlamaktadır. ÖGG sekiz temel uygulama içermektedir: (1) Problem alanının incelenmesi ve tanımlanmasını ve özelliklerin belirlenmesini içeren *Alan Obje Modellemesi,* (2) Tanımlanmış bir dizi özellik üzerinden geliştirme ve ilerlemeyi izlemeyi içeren *Özelliklere Göre Geliştirme*, (3) Bakımı ve devamlılığı kolaylaştırmak için her sınıfın bir geliştiriciye aitliğini belirten *Bireysel Kod Sahipliği,* (4) Küçük ve dinamik olarak oluşturulan *Özellik Takımları*, (5) Kontrol listesi tabanlı toplantıları ve kod standartlarını içeren *Denetleme,* (6) Yeni bir özelliğin eklenebileceği, her an için çalışan bir sistemi garanti eden *Düzenli Yapılar*, (7) Versiyonlama takibini sağlayan *Konfigürasyon Yönetimi,* (8) Tüm önemli organizasyonel yapıya işin tamamı hakkındaki bilgileri iletmeyi içeren *İlerlemeleri Rapor Etme.* ÖÖG 5 süreç içerir ki ilk 3'ü proje başında yapılır: (1) Çekirdek metotların ve projenin şeklinin belirlendiği *Baştan Başa Bir Model Geliştirme,* (2) Özellikler listesi çıkarma, (3) Özelliklere göre planlama ve son ikisi de her bir yineleme içinde yapılır: (4) Özelliklere göre tasarlama, (5) Özelliklere göre gerçekleştirim. Bu süreçler içinde küçük takımlar modellemeye önem vererek

özellik setleri üzerinde çalışırlar. Diğer adapte edilen yöntemler gibi gözle görülür işlevselliğe sahip teslimleri içeren iki haftalık küçük yinelemelerden oluşur [9, 16, 17].

2.7 Dinamik Sistem Geliştirimi (Dynamic System Development)

DSG, hızlı uygulama geliştiriminden (Rapid Application Development) gelişerek büyüyen bir uygulamadır. Prototiplemeye önem veren bu yöntem fizibilite ve iş alanı çalışmalarıyla başlar. Fizibilite çalışmasıyla DSG metodunun uygunluğuna karar verilir. İş alanı çalışmasıyla ise geliştirimin yapılacağı iş alanını analiz etmek amacıyla bilgi toplanır. Sonuçta ana hatlarıyla sistem mimarisi ve proje planı ortaya çıkarılır. DSG sürecinin kalan kısmı her biri yinelemeli ve artımlı olmak üzere 3 alt süreçten oluşur: *fonksiyonel model çevrimi*; analiz dokümantasyonu ve fonksiyonel prototipleri içerir, *tasarlama ve geliştirim çevrimi*; tasarım prototiplerini içerir ve *kodlama çevrimi*; kodlamayı, son kullanıcı eğitimini ve kabul sürecini içerir. DSG aktif kullanıcı etkileşimini, sık teslimleri, güçlü ve karar vermede yetkin bir takımı, iş amaçlarına tam uyumu, gereksinim bazlı çalışmayı, sıkı işbirliğine dayalı takım çalışmasını, tüm artımlar boyunca test yapmayı içeren prensipleri temel alır. Diğer çevik yöntemler gibi kısa zamanla sınırlı 2 ila 6 hafta süren yinelemeleri kullanır. DSG çevik yöntem yaklaşımını izlerken fazlaca geleneksel yöntemlerin yapısını içerdiğinden dikkate değerdir. Müşteri ile etkileşimin yüksek olduğu, kullanıcı grubunun net bir şekilde tanımlanabildiği, kesin zaman kısıtlarının olduğu, gereksinimlerin öncelik sırasına göre dizilebildiği, gereksinimlerin durağan ve tam tanımlı olmadığı durumlarda kullanılabilir. Küçük projelere uygulanabildiği gibi birden fazla küçük grubu içeren büyük projelere de uygulanabilir. [1, 7, 9, 11]

3 . Planlı Mı Çevik Mi ?

Geleneksel ve çevik yöntemlerin birbirlerine karşı üstünlüklerini savunan pek çok görüş olmasına karşın her bir yaklaşımın kendine has uygulama avantajları vardır. Yazılım sektöründe fark yaratmak, rekabet ortamında başarılı olmak ancak hızlı ve yeni fikirlerle ortaya çıkarak başarılabilir. Geleneksel ve çevik yöntemleri proje için alınan riskler açısından değerlendirirsek çevik yöntemler, geliştirim hızını risk almaya tercih eder. Projede ne kadar planlama ve çeviklik kullanılacağı ve bu kullanımın getireceği riskler göz önüne alınmalıdır. [18] Çevik ve geleneksel yöntemler; geliştiriciler, müşteriler, gereksinimler, mimari, varolan kodun iyileştirilmesi, büyüklük ve temel amaç konularına göre aşağıda özetlenmiş ve değerlendirilmiştir [19, 20]:

(1) Geliştiriciler açısından bakıldığında: Çevik yöntemler özellikle kişiler arası iletişim ve bilgi paylaşımına önem verdiği için bireysel faktörlerin -arkadaşlık, beceri ve iletişimin- yeri önemlidir. İletişime verilen önem, yazılı olmayan bilgilerin yanlış yapılandırılması sonucunda oluşabilecek kötü bir tablo olasılığını da beraberinde getirir. Buna karşın geleneksel yöntemler -her ne kadar değişikliklere karşı bilgileri güncellemek zor ve zaman alıcı olsa da- bu riski planlama ile aşarlar.

(2) Müşteri katılımı açısından bakıldığında: Proje geliştiriminde; gereksinimlerin iyi belirlenmesi, değişikliklerin ve geri beslemenin daha çabuk yapılması açısından müşteri katılımı önemli yer tutar. Çevik yöntemler müşteriyi de katılımcı olarak gören, müşteri odaklı bir yaklaşım izlerken iletişim problemlerinden, yanlış anlaşılmalar ve yanlış tanımlamalardan kaynaklanan riskleri de taşımaktadır. Buna karşın geleneksel yöntemler dökümantasyon tutarak ve mimariyi gözden geçirme toplantıları yaparak, bu problemlerin üstesinden gelmektedir. Bu yöntemde müşteri temsilci rolünde dışardan yetkili kişi konumundadır.

(3) Gereksinimlerin değişebilirliği açısından bakıldığında: Çevik yöntemler gereksinimlerin çabuk değişmesine kolayca adapte olabilen bir yaklaşımdır. Geleneksel yöntemler; gereksinimlerin oldukça sabit olduğu, ayda %1 oranda değiştiği projelerde kullanılabilecek en iyi yaklaşımdır.

(4) Projenin mimarisinin değişebilirliği açısından bakıldığında: Projede uygulama ayrımı, çalışan iş bölümü, konfigürasyon yönetimi ve kaynak seçimi mimariye göre yapılandırıldığından mimarinin değişmesi zor ve zahmetli bir süreçtir. Çevik yöntemler, geçerli versiyonu geliştirirken o an için gerekli olmayan gereksinimleri göz önüne almayarak ekstra iş yapmamayı öngörür. Fakat; tahmin edilebilir gereksinimlerin varlığında mimarinin buna uygun olarak yapılandırılmaması ve her yinelemede değiştirilmesi zaman ve ekstra çalışma gerektirir. Geleneksel yöntemler ise gelecekteki ve şu anki gereksinimleri göz önüne alarak geliştirim yapar. Tahmin edilebilir gereksinimlerin varlığında ileriye yönelik olarak mimarinin geliştirilmesi ile hem zamandan kazanılır hem de ekstra çalışma önlenmiş olur. Elbette mimari seviyesinde yapılacak değişikliklerden her iki yöntem de etkilenecektir.

(5) Varolan kodun iyileştirilmesi açısından bakıldığında: Çevik yöntemler kullanılarak geliştirilen küçük sistemlerde kodu gözden geçirmek bir risk oluşturmamaktadır. Diğer taraftan gereksinimlerin ve kullanıcı hikayelerinin (user stories) artmasına veya mimarinin değişmesine bağlı olarak da gözden geçirme sürecinin artacağı göz önüne alınmalıdır. Geleneksel yöntemlerde ise kodun iyileştirilmesi tüm sistem kodunu kapsadığından gözden geçirme görevleri hem zaman alacak hem de ekstra iş yükü getirecektir.

(6) Proje büyüklüğü açısından bakıldığında: Proje ekibinin büyük olması çevik yöntemlerin kullanım oranını azaltır. Buna karşın 250 ve üzeri çalışanla geliştirilmiş başarılı projeler olduğunu da göz önüne almak gerekir [21]. Geleneksel yöntemlerin çok sayıda geliştirici içeren geniş takımlar için uygun olduğu, küçük takımlar bu yaklaşımı kullandığında çok verimli olmadıkları görülmektedir.

(7) Geliştirimdeki temel amaç göz önüne alındığında: Çevik yöntemlerde ana amaç, geliştirilmiş yazılımın sürekli ve erken teslim edilmesiyle müşteriyi memnun etmektir. Geleneksel yöntemler ise sistem gereksinimlerinin başta tam ve doğru olarak belirlenebileceği düşüncesinden yola çıkarak yazılımı eksiksiz geliştirme amacı gütmektedir.

4 . Karma Yaklaşım

Geleneksel ve çevik yöntemlerin iki farklı kutupta olduğu düşünülse de ikisinin sentezi geliştiricilere geniş çaplı uygulama seçenekleri sağlamaktadır [18]. Sadece hızlı üretim ya da sadece yüksek başarı günümüzde geniş kitlelere hizmet veren sektörler için yeterli olmamaktadır, her ikisi birden tercih edilmektedir. Bunu başarabilmek için geleneksel ve çevik yöntemlerin bir arada kullanılması önerilmektedir. [19] Her iki yaklaşımın da yetersiz ve üstün olduğu yerler olduğundan, projenin yapısı incelendikten sonra risk analizi yapılarak iki yaklaşım arasında denge kurulmaya çalışılmalıdır. Bu dengenin sağlanabilmesi için organizasyonlar yetkin değerlendirmeler yaparak etkili süreçler geliştirmelidir. Proje geliştiriminde çevik ve geleneksel yöntemlerin bir arada kullanılarak dengelenmesi konusunda aşağıda ortak görüşler belirtilmektedir [22]:

- Ne çevik ne de geleneksel yöntemler tek başına tüm projelerin sorunlarını tamamen çözemeyebilir.
- Çevik ve geleneksel yöntemlerin birbirlerine üstün oldukları noktalar vardır.

- Gelecekteki eğilim, çeviklik ve disiplin gerektiren bir uygulama geliştirimi olacaktır.
- Her iki yaklaşımı dengeleyen yaklaşımlar ortaya çıkmaktadır.
- Organizasyonunuza uygun bir yöntem geliştirip bunu geliştirim sürecinde kullanmak daha iyi olacaktır.
- Yöntemler önemlidir; fakat insanlar, insani değerler, iletişim ve beklenti yönetimine önem vermek potansiyel güç noktasıdır.

Çevik ve geleneksel yöntemlerin birbirinden ayrıldığı noktalar, önem verdikleri geliştirim parçaları ve genel özellikleri göz önünde bulundurularak projenin hangi yöntem ile geliştirilmeye daha yatkın olduğu belirlenmelidir. Projenin özellikleri dikkate alınarak hangi yönteme daha eğilimli olduğunun belirlenmesine etki eden riskler 3 kategoride toplanmıştır [23]: (1) Çevik yöntemlerden kaynaklanan riskler: ölçeklenirlik, kritiklik, tasarımda sadelik, personel yetenekleri. (2) Geleneksel yöntemlerden kaynaklanan riskler: gereksinimleri ortaya çıkarma, sürekli değişiklik, hızlı çözümlere gereksinim, personel yetenekleri. (3) Genel çevreden kaynaklanan riskler: teknoloji belirsizlikleri, çeşitli ortaklar, kompleks sistemler. Bu riskler proje için tek tek incelenmeli ve olasılıkları belirlenmelidir. Dikkatli yapılan analiz çalışması sonucunda her iki yöntemin bir arada nasıl kullanılacağına karar verilmelidir. Böylece her iki yöntemin güçlü noktaları entegre edilerek daha güçlü bir yaklaşım izlenmiş olur.

5 . Sonuç

Günümüzde yazılım, gereksinimler yeteri kadar iyi anlaşılmadan tasarlanmakta, kodlanmakta ve müşteriye gösterilmektedir. Hata ve eksiklikler belirlenip düzeltildikten sonra yazılım müşterinin ihtiyaçlarını karşılar hale getirilmektedir. Bu duruma tepki olarak, çevik yöntemler ortaya atılmıştır. Yazılım yöntemleri kullanılarak müşteri odaklı, yazılım ihtiyaçlarının belirginleştirilmesi ve küçük adımlarla gerçekleştirilmesi temeline dayanan bir yaklaşım izlenmektedir. Çevik yöntemlerden en önemlileri Ekstrem Programlama, Scrum, Kristal, Açık Kaynak Geliştirimi, Uyarlanabilir Yazılım Geliştirimi, Özellik Güdümlü Geliştirim ve Dinamik Sistem Geliştirimidir. Geleneksel ve çevik yöntemlerin geliştiriciler, müşteriler, gereksinimler, mimari, varolan kodun iyileştirilmesi, büyüklük ve temel amaç konularına bakış açıları birbirinden farklıdır. Bu farklılıklar, her iki yöntemin kendi alanlarında yetersiz ve üstün oldukları noktaları göz önüne serer. Bu noktalara dikkat ederek projenin özelliklerine göre en uygun yöntem seçilebilir ya da projenin taşıdığı riskler göz önüne alınıp bu iki yöntem dengelendikten sonra daha etkili bir süreç yapısı izlenebilir.

6 . Kaynakça

[1]. Jim Highsmith, "What is agile Software Development?", CrossTalk, The journal of Defense Software Engineering, Ekim 2002

[2]. Agile Manifesto, 2001, ulaşabilecek adres http://www.agilemanifesto.org

[3]. Don Wells, "Extreme Programming", 2004, ulaşılabilecek adres http://www.extremeprogramming.org/rules.html

[4]. Ronald E Jeffries, "XProgramming.com an agile development resource", 1999-2004

[5]. Dirk Riehle Skyva, "A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming: How Methodologies May Learn from Each Other", 2000

[6]. "Scrum It's about common sense", Control Chaos, 2005, ulaşılabilecek adres http://www.controlchaos.com/about/how.php

[7]. Pekka Abrahamsson, Outi Salo, Jussi Ronkainen ve Juhani Warsta, "Agile Software Development Methods, Review and Analysis", VTT Publications 478, EPSOO 2002

[8]. Colin Bird ve Haward van Rooijen, "The value of agile methods", 12 Nisan 2005

[9]. Mike Cohn, "Selecting an agile process, comparing the leading alternatives", Mountain Goat Software, October 2002

[10]. Jim Highsmith, "Agile Methodologies: Problems, Principles and Practices", 2001

[11]. Martin Fowler, The new methodology, Nisan 2003, ulaşılabilecek adres http://www.martinfowler.com/articles/newMethodology.html#N101F8

[12]. "Primavera Success Story", Object Mentor, Inc. and Advanced Development Methods, July 2004

[13]. Everette R. Keith, "A software development process, A different approach to software design", Dec 2002

[14]. Alistair Cockburn, "Agile Software Development", Agile Software Development Series, 2000-2001

[15]. Alistar Cocburn, Jim Highsmith, Kay Johansen ve Mike Jones, "Crystal Methodologies", 2004

[16]. Steve Palmer ve Peter Coad, "Feature Driven Development(FDD)"

[17]. Neubulon Pty. Ltd., "Feature Driven Development(FDD)", 2002-2004, ulaşılabilecek adres http://www.featuredrivendevelopment.com

[18]. Tom De Marco ve Barry Boehm, "The Agile Methods Fray", Haziran 2002

[19]. Barry Boehm, "Get ready for agile methods,with care", IEEE Şubat 2002

[20]. Alistair Cockburn ve Jim Highsmith, "Agile Software Development: The business of innovation", IEEE Journal, Eylül 2001

[21]. Cockburn and J. Highsmith, "Agile Software Development: The People Factor," Computer, Nov. 2001

[22]. Barry Boehm ve Richard Turner, "Balancing Agility and Discipline: A Guide for the Perplexed", 2003

[23]. Laurie Willams ve Alistair Cockburn, "Agile software development:It's about feedback and change", Haziran 2003