**Çevik Metodolojilere Giriş\***

Çevik metodolojiler, geleneksel olmayanların hayal gücünü yakalayan geleneksel olmayan yazılım geliştirme katmanlarından oluşan bir ailedir,

süreç yüklü yaklaşımlar. Çevik metodolojiler eksiklikleri ile karakterize edilir

bu gerçek, çevik metodolojilerin ne zaman olduğu anlamına gelmese de, katı sürecin

doğru şekilde kullanılır, titiz değildir ve endüstriyel uygulamalar için uygun değildir — bunlar

oluyorlar. Bununla birlikte, çevik yaklaşımlarda karakteristik olarak eksik olan, zorunlu toplantılara odaklanan ”yemek kitabı" çözümleri ve geliştirme yaklaşımlarını öngören karmaşık belgelerdir.

Çevik metodolojiler yazılım mühendisliği için geçerlidir. Öğeler varken

sistemlerin mühendisliğine uygulanabilecek çevik metodolojilerden (insan düşüncelerine göre), bu metodolojiler genellikle şu şekilde tanımlanır

yazılım dışı sistemlere uygulandığında hafif veya yalın. Bu çok çevik olduğu için

metodolojiler, bir dizi hızlı, uzaklaşmayan prototipe, bir yaklaşıma bağlıdır

bu donanım tabanlı sistemlerde pratik değildir. Her halükarda, yazılım dışı engi neer hala bu bölümden yararlanabilir çünkü çevik metodolojiler giderek artmaktadır

istihdam edilmek ve çevik yazılım mühendisinin zihniyeti şunları içerdiğinden

bazı sağlıklı bakış açıları.

Yazılım ve Sistemler için Gereksinimler Mühendisliği

Gelişmenin daha iyi yollarını ortaya çıkarıyoruz

bunu yaparak ve başkalarına yardım ederek yazılım.

kaba bu işe değer verdik:

Bireyler ve etkileşimler

süreçler ve araçlar

•

•

Kapsamlı üzerinde çalışma yazılımı

belge

\* Sözleşme üzerinden müşteri işbirliği

müzakere

\* Aşağıdaki değişikliklere yanıt verme

plan

at , öğelerde değer varken

sağda, soldaki öğelere daha çok değer veriyoruz.

Çevik metodolojilerin doğasını tam olarak anlamak için şunları yapmamız gerekir

çevik Manifesto adlı bir belgeyi ve arkasındaki ilkeleri inceleyin.

Çevik Manifesto, çevik'in önde gelen savunucuları tarafından tanıtıldı

felsefelerini açıklamak için metodolojiler (bkz. Şekil 7.1).

Çevik Manifesto'nun imzacıları arasında modern yazılımın birçok armatürü yer alıyor

kent Beck, Mike Beedle, Alistair Cockburn, Ward gibi mühendislik uygulamaları

Cunningham, Martin Fowler, Jim Highsmith, Ron Jeffries, Brian Marick, Robert

Martin, Steve Mellor ve Ken Schwaber. Çevik Manifesto'nun altında yatan bir dizi

prensipler. Aşağıda ilkelerine bak, bu yönleri üzerinde vurgu olduğuna dikkat çeken

italik olarak belirlediğimiz gereksinim mühendisliğine odaklanın.

**Çevik Manifesto'nun Arkasındaki İlkeler**

team Düzenli aralıklarla, ekip nasıl daha etkili olunacağını düşünür, o zaman

davranışını buna göre ayarlar ve ayarlar.

- En yüksek önceliğimiz, müşteriyi erken ve sürekli olarak tatmin etmektir

değerli yazılımların teslimi.

-Gelişimin sonlarında bile değişen gereksinimleri memnuniyetle karşılıyoruz. Çevik süreçler müşterinin rekabet avantajı için değişir.

-Working Çalışma yazılımını birkaç haftadan birkaç haftaya kadar sık sık teslim edin.

aylar, daha kısa zaman çizelgesini tercih ederek.

-Business İş adamları ve geliştiriciler proje boyunca günlük olarak birlikte çalışmalıdır.

-Motivasyonlu bireyler etrafında projeler oluşturun. Onlara çevreyi verin ve

ihtiyaçları olan desteği ve işi yapmaları için onlara güvenin.

-A ve içinde bilgi aktarmanın en etkili ve etkili yöntemi

geliştirme ekibi yüz yüze görüşüyor.

Working Çalışma yazılımı ilerlemenin birincil ölçüsüdür.

Agile Çevik süreçler sürdürülebilir kalkınmayı teşvik eder. Sponsorlar, geliştiriciler,

ve kullanıcılar süresiz olarak sabit bir tempoyu koruyabilmelidir.

Technical Teknik mükemmelliğe ve iyi tasarıma sürekli dikkat, geliştirir

atiklik.

Yapılmaması gereken işin miktarını maksimize edilmesi

basitlik—sanat—tial essen vardır. "İşe yarayabilecek en basit şeyi yapın.”

best En iyi mimariler, gereksinimler ve tasarımlar kendi kendini organize etmekten ortaya çıkar

takımlar (Beck 2000).

İlkelerin süreç boyunca değişmeyi gerektiren kavramı nasıl kabul ettiğine ve benimsediğine dikkat edin. Ayrıca, çevik ilkeler kişisel iletişimi vurgulamaktadır (bu özellik mühendislikte faydalıdır

yazılım dışı sistemler de). Gereksinim mühendisliğinin öne çıkan özellikleri

çevik süreç modellerinde “geleneksel" şelaleden farklı ve daha modern

yinelemeli, evrimsel veya spiral gelişim gibi modeller. Bu diğer modeller

gereksinimler mühendislik süreci üzerinde çok sayıda ön çalışmayı tercih edin ve

üretim, genellikle, hacimli gereksinimleri özellikleri belgeleri.

Çevik Yazılım Geliştirmenin Faydaları

Çevik yazılım geliştirme yöntemleri yinelemeli yöntemlerin bir alt kümesidir\*

bu odaklanmak

değişimi benimsemek ve işbirliğini ve erken ürün sunumunu vurgularken

kaliteyi korumak. Çalışma kodu, geliştirme sürecinin gerçek eseri olarak kabul edilir. Modeller, planlar ve belgeler önemlidir ve

değer, ancak yalnızca çalışma yazılımının geliştirilmesini desteklemek için var, aksine

daha önce tartışılan diğer yaklaşımlarla. Ancak, bu bir ifade etmez

çevik geliştirme yaklaşımı herkes için ücretsizdir. Çevik metodologların benimsemesi gereken çok açık uygulamalar ve öncelikler vardır.

Çevik yöntemler öngörücüden ziyade uyarlayıcıdır. Bu yaklaşım, yazılımı büyük ölçüde planlamayı vurgulayan süreç modellerinden önemli ölçüde farklıdır

uzun bir süre boyunca ve yazılımda hangi önemli değişiklikler için ayrıntı

gereksinim belirtimi sorunlu olabilir. Çevik yöntemler aşağıdakilere bir yanıttır

bataklığa neden olabilecek sürekli değişen gereksinimlerin ortak sorunu

başlangıçta dokümantasyona odaklanan daha “törensel” ön tasarım yaklaşımları.

Çevik yöntemler de süreç odaklı olmaktan çok insan odaklıdır. Bunun anlamı

açıkça geliştirme “benimle eğleniyor vurgu yapmışlar." Muhtemelen, bunun nedeni yazılım gereksinimleri spesifikasyonlarının ve yazılım tasarımının yazılmasıdır

açıklamaları zahmetli ve dolayısıyla, simge durumuna küçültülmüş.

Inayat ve diğ. çevik yazılım projelerinin çalışmalarının sistematik bir incelemesini gerçekleştirdi

2002 ve 2012 yılları arasında yayınlandı. Beş geleneksel gereklilik belirlediler

çevik metodolojilerde bulunan zorunlu mühendislik uygulamaları ile ortadan kaldırılmış veya en aza indirilmiş mühendislik zorlukları. Bu zorluklar şunlardı:

-Tüm paydaşlarla iletişim açığını kapatmak

-Müşteri katılımını artırmak

-Dokümantasyonun boyutunu ve karmaşıklığını azaltma

-Kapsam sürünme azaltma

-Gereksinimlerin doğrulanmasına ulaşmak (Inayat ve ark. 2015)

hey, güçlendirmek için daha fazla araştırmaya ve ampirik kanıtlara ihtiyaç olduğunu belirtti

ve bu sonuçları genelleştirin, ancak çalışmalarının sonuçları hala güçlü bir şekilde işaret ediyor

çevik yöntemlerin gücü.

Çevik metodolojiler arasında Crystal, Extreme Programlama, Scrum, Dinamik

sistem geliştirme yöntemi (DSDM), özellik odaklı geliştirme ve uyarlanabilir

programlama ve başkaları da var. En çok ikisine daha yakından bakacağız

yaygın olarak kullanılan çevik metodolojiler: XP ve Scrum.

**Aşırı Programlama**

**Aşırı Programlama (XP)\***

en yaygın kullanılan çevik metodolojilerden biridir.

XP, geleneksel olarak daha küçük geliştirme ekiplerine yöneliktir ve nispeten az sayıda ayrıntılı yapı gerektirir. XP, gelişimine yinelemeli bir yaklaşım getiriyor

döngüler. Oysa evrimsel veya yinelemeli yöntemin hala farklı gereksinim analizi, tasarımı, uygulaması ve şelale yöntemine benzer test aşamaları, XP bu faaliyetleri birbiriyle ilişkili ve süreklidir.

XP, geliştiricilerin yanıt vermesine yardımcı olan bir dizi 12 temel uygulamayı teşvik eder ve

kaçınılmaz değişimi kucaklayın. Uygulamalar dört uygulamaya göre gruplandırılabilir

alanlar:

◾ Planlama

◾ Kodlama

◾ Tasarım

◾ Test

\* Aşırı programlama bazen "XP"yi vurgulamak için "eXtreme Programming" olarak da yazılır.

**Scrum**

Bir ragbi maçında özellikle çekişmeli bir noktadan sonra adlandırılan Scrum,

genelinde sözlü iletişimi teşvik ederek kendi kendini organize eden ekipleri mümkün kılar.

ekip üyeleri ve tüm paydaşlar arasında. Scrum'ın temel bir ilkesi,

şelale gibi geleneksel, plan odaklı yazılım geliştirme metodolojileri

ve yinelemeli geliştirme, sürece çok fazla odaklanır ve yazılıma yeterince odaklanmaz. Ayrıca, plana dayalı geliştirme, yazılım dışı yapılara odaklanırken

(örn., dokümantasyon) ve süreçler (örn., resmi incelemeler), Scrum aşağıdakileri vurgular:

işleyen yazılımları erken ve sıklıkla üretmenin önemi. Scrum, tüm paydaşlar arasında yüksek kaliteli iletişimi teşvik ederek kendi kendine örgütlenmeyi teşvik eder. Bu durumda, sorunun tam olarak anlaşılamayacağı veya tanımlanamayacağı zımnendir.

(kötü bir sorun olabilir). Ve Scrum'daki odak, ekibin performansını en üst düzeye çıkarmaktır.

ortaya çıkan zorluklara çevik bir şekilde yanıt verme yeteneği.

Scrum, yapılması gereken öncelikli işlerin canlı bir birikimine sahiptir. Bir

büyük ölçüde sabit birikmiş iş listesi kalemleri seti, bir dizi kısa (yaklaşık 30 gün) içinde gerçekleşir.

yinelemeler veya sprintler. Her gün kısa (örneğin 15 dakikalık) bir toplantı veya Scrum yapılır

ilerlemenin açıklandığı, gelecek çalışmaların tanımlandığı ve engellerin

kabarık. Tamamlanacak biriktirme listesi öğelerini tanımlamak için her sprintin başlangıcında kısa bir planlama oturumu gerçekleşir. Kısa bir ölüm sonrası veya kalp atışı retrospektif incelemesi

sprint sonunda gerçekleşir (Şekil 7.3).

Bir Scrum Master, her bir sprint için engelleri veya engelleri kaldırır. Scrum

Usta takımın lideri değildir (kendi kendini organize ettikleri için) ancak takım ile herhangi bir istikrarsızlaştırıcı etki arasında bir üretkenlik tamponu görevi görür. Bazı organizasyonlarda Scrum Master'ın rolü kafa karışıklığına neden olabilir. Örneğin, eğer iki

Bir Scrum takımının üyeleri birlikte iyi çalışmıyorsa, bir

üst düzey yönetici, Scrum Master'ın sorunu çözdüğünü söyler. Takım işlev bozukluğunu düzeltmek

Scrum Master'ın rolü değil. Personel sorunlarının hat tarafından çözülmesi gerekiyor

ilgili tarafların rapor verdiği yöneticiler. Bu senaryo şu ihtiyacı göstermektedir:

bu tür yaklaşımlar olduğunda çevik metodolojiler hakkında kurum çapında eğitim

istihdam edilecek.

**Çevik Metodolojiler için Gereksinim Mühendisliği**

Geleneksel (burada şelale, evrimsel, iteratif, spiral vb.) ve çevik metodolojiler gereksinimlerin toplanmasındadır. Aslında, çevik metodolojilerin bazı savunucuları, sözde avantajları kullanırlar.

çevik yöntemler için bir satış noktası olarak çevik gereksinim mühendisliği uygulamaları.

Çevik metodolojiler için gereksinim mühendisliği yaklaşımları çok fazla olma eğilimindedir.

daha gayri resmi. Diğer bir fark, gereksinim mühendisliği faaliyetlerinin zamanlamasındadır.

Geleneksel sistemlerde ve yazılım mühendisliğinde, gereksinimler sürecin ön ucunda toplanır, analiz edilir, rafine edilir vb. Çevik yöntemlerde, gereksinimler mühendislik devam eden bir faaliyettir; yani, gereksinimler rafine edilir ve keşfedilir her sistem inşası ile. Prototiplemenin kullanıldığı spiral metodolojilerde bile gereksinimlerin iyileştirilmesi için, gereksinim mühendisliği çok daha az geç ortaya çıkar.

geliştirme süreci.

Müşteriler, sürekli olarak gereksinimlerin keşfedilmesi ve iyileştirilmesi ile ilgilenirler.

çevik yöntemler. Tüm sistem geliştiricileri, gereksinim mühendisliği faaliyetinde yer alır ve her biri müşterilerle düzenli etkileşime girebilir ve olmalıdır. İçinde

geleneksel yaklaşımlar, müşterinin gereksinimler bir kez daha az katılımı vardır

şartname yazılmış ve onaylanmıştır ve tipik olarak, katılım genellikle

sistem geliştiricileri ile değil.

Muhtemelen, gereksinim mühendisliğine çevik yaklaşım çok daha fazlasıdır.

p boyunca değişikliklere karşı savunmasız

**Çevik metodolojilerde genel uygulamalar**

Cao ve Ramesh (2008) XP veya Scrum (veya her ikisini) kullanarak 16 yazılım geliştirme organizasyonu üzerinde çalışmışlardır ve doğası gereği çevik olan yedi gereksinim mühendisliği uygulamasını

ortaya çıkarmışlardır.

1-Yüz yüze iletişim (yazılı özellikler üzerinden)

2-Yinelemeli gereksinim mühendisliği

3-Aşırı Önceliklendirme (başlangıçta bir kereden ziyade sürekli önceliklendirme, ve önceliklendirme esas olarak iş değerine dayalıdır)

4-Sürekli planlama

5-Prototipleme

6-Test odaklı geliştirme

7-İncelemeler ve testler

Bu uygulamalardan bazıları çevik olmayan geliştirmede bulunabilir (örneğin, teste dayalı geliştirme ve prototip oluşturma), ancak bu yedi uygulama incelenen tüm kuruluşlarda tutarlıydı.

**Çevik Yazılım Geliştirme Örnek Uygulaması**

Temel yazılım gereksinim özelliklerinin aksine; çevik yöntemlerdeki temel yapı, sürekli gelişen ve iyileştirilen gereksinimler yığınıdır. Bu gereksinimler genellikle kullanıcı hikayeleri şeklindedir.

Her halükarda bu gereksinimler müşteri ve öncelik verilmişler tarafından oluşturulur.

Gereksinimdeki ayrıntı düzeyi ne kadar yüksek olursa, öncelik de o kadar yüksek olur.

Yeni gereksinimler keşfedildikçe yığına eklenir ve önceliklendirmeyi korumak için yığın yeniden karıştırılır (figür7.4)

Listeden herhangi bir zamanda gereksinimlerin eklenmesi, değiştirilmesi veya çıkarılması konusunda herhangi bir kısıtlama yoktur (müşteriye muazzam bir özgürlük veren).

Tabii ki, sistem oluşturulduktan sonra veya muhtemelen inşa edilirken, kullanıcı hikayeleri yığını, sistem bakımı ve diğer geleneksel amaçlar için geleneksel bir yazılım gereksinimleri spesifikasyonuna dönüştürülebilir.

Evcil hayvan mağazası satış noktası sistemini düşünün. Sistemi geliştirmek için bir çevik yazılım geliştirme metodolojisi-özellikle Scrum- kullanmak istediğimizi varsayalım. Gereksinim mühendisliği sürecinin nasıl görünebileceğine bakalım. Scrum Master da dahil olmak üzere beş kişilik bir geliştirme ekibimiz olduğunu varsayalım. Basitlik için, tek paydaşın evcil hayvan mağazası sahibi , mağaza müşterileri, kasiyerler ve muhasebeciler olduğunu varsayalım. Mağaza müşterileri, kasiyerler ve muhasebeciler için temsilciler seçeceğiz.

Toplu olarak, onlara “paydaş paneli” diyelim. Şimdi aşağıdaki faaliyetleri düşünün.

1. Scrum Master, sistem gereksinimleri, kısıtlamalar ve temel kurallar hakkında temel bir anlayış elde etmek için geliştirme ekibi ve müşteri (mağaza sahibi) arasında bir dizi toplantı düzenler.

Bu tartışmalara dayanarak, Scrum Master, muhtemelen yapılandırılmış görüşmeler, kart sıralama ve diğer paydaşlarla odak grupları dahil olmak üzere bir dizi ek bilgi toplama etkinliği düzenler. Doğrulama faaliyeti olarak bir QFD çalışması kullanılır (çünkü muhtemelen karşılaştırılabilecek birçok başka POS sistemi vardır). Bu faaliyetlerin amacı, her biri için efor tahminleri de dahil olmak üzere bir dizi öncelikli kullanıcı hikayesi (diyelim ki yaklaşık 100) üretmektir. Ancak müşteri bize "son teknoloji" bir sistem istediğini söyledi, bu nedenle sistem geliştirme sürecinden geçerken müşterinin ve diğer paydaşların yeni veya farklı işlevsellik isteyeceklerini biliyoruz.

1. Geliştirme ekibi, kullanıcı hikayelerini analiz eder ve bunların bir alt kümesini seçer; ilk sprint için 10 diyelim. Bu 10 hikaye, en "kullanıcıya yönelik" hikayeler olacak ve mümkün olduğunca daha sonra için çok fazla arka uç işleme bırakacaktır.
2. Geliştirme ekibi, geliştirmeye başlamadan önce bu 10 kullanıcı hikayesine daha fazla açıklık getirmek için paydaşlara tekrar danışır.
3. Her gün bir scrumla başlar. Geliştiriciler, sistem tasarımını ve birim test durumlarını organik olarak türetmek için test odaklı tasarımı kullanır. Scrum Master, paydaşların karşılaştığı soruların hızlı bir şekilde yanıtlanabilmesi için geliştirme ekibi ve çeşitli paydaşlar arasındaki iletişimi kolaylaştırır. Yaklaşık 30 gün sonra ilk sprint tamamlanır.
4. Paydaş paneli, geri bildirim için sprint sırasında oluşturulan sistemle birlikte sunulur. Bu geri bildirim, mevcut sistem artışındaki değişiklikleri, eklenecek yeni özellikleri ve muhtemelen atlanacak özellikleri içerecektir.
5. Bu geri bildirime dayanarak, geliştirme ekibi yeni kullanıcı hikayeleri oluşturur ve bunları biriktirme listesine ekler, diğer kullanıcı hikayelerini gerektiği gibi değiştirir ve biriktirme listesini yeniden düzenler.
6. Geliştirme ekibi, sonraki 30 günlük sprint için biriktirme listesinden yeni bir kullanıcı hikayesi seti seçer (örneğin 10) ve süreç adımları (C'den F'ye) biriktirme listesi tükenene kadar tekrarlanır.

Müşteri ile yapılan sözleşmenin şartlarına bağlı olarak ek kabul testleri yapılabilir. Bu test, sözleşme görüşmesi sırasında geliştirilen bir dizi kritere dayalı olacaktır. Bir gereksinim belirtimine dayanan bu tür kabul testleri, geleneksel geliştirme için olağandır, ancak çevik geliştirme için tipik değildir.

Başlangıçta 100 kullanıcı hikayesi vardı, ancak sprint (ay) başına 10 kullanıcı hikayesi planlasak bile, toplam geliştirme süresi muhtemelen 10 aydan uzun olacaktır. Bunun nedeni, yeni kullanıcı hikayelerinin eklenmesi ve eskilerinin değiştirilmesidir. Elbette Scrum Master, proje tamamlama tahminlerini hassaslaştırmak için her sprint sırasında proje hızını takip eder.

**Çevik Ne Zaman Önerilir?**

Çevik yazılım metodolojilerinin kullanımı, en azından anket verilerinin gösterdiği gibi, endüstride çok popüler hale geliyor. Örneğin, bir Forrester/Dr. Dobbs geliştirici anketi, yanıt verenlerin %35'i bir tür çevik metodoloji kullandığını bildirdi (Batı ve diğerleri 2010). Daha yakın zamanlarda, Kassab ve ark. (2014), ankete katılan yazılım mühendislerinin %46'sının çevik bir metodoloji kullandığını tespit etti.Çevik yöntemlerin Amerika Birleşik Devletleri'nde şelale geliştirmenin iki katından daha popüler olduğu ve Amerika Birleşik Devletleri dışında yedi kat daha popüler olduğu bulundu. Ayrıca, çevik yazılım geliştirmenin, yaklaşık olarak eşit olarak kullanıldığı kuzeydoğu hariç, Amerika Birleşik Devletleri'ndeki her bölgede şelaleden daha sık kullanıldığını buldular. Son olarak araştırma, finans, bankacılık ve sigorta dışındaki her sektörde çevik yöntemlerin şelale tarzı geliştirmeden daha sık kullanıldığını buldu. En önemlisi, çalışma, katılımcıların şelale stili geliştirmeye kıyasla çevik geliştirmeyi kullanırken gereksinim mühendisliği uygulamalarından daha fazla memnuniyet duyduklarını bulmuştur (Kassab ve diğerleri, 2014).

Ancak şu soru ortaya çıkıyor: Çevik geliştirme metodolojileri ne zaman kullanılmalıdır? Boehm ve Turner, soruyu yanıtlamanın yolunun projeye beş boyutlu bir süreklilik içinde bakmak olduğunu öne sürüyorlar; boyut (katılan personel sayısı açısından), sistem kritikliği, personel beceri düzeyi, dinamizm (belirli bir zaman aralığında beklenen sistem değişikliği sayısı) ve organizasyon kültürü (kuruluşun kaos veya düzen içinde gelişip gelişmediği) (Boehm ve Turner 2003) . Şekil 7.4'te, proje özellikleri diyagramın merkezinden uzaklaştıkça, çevik metodolojileri kullanarak başarılı olma olasılığı düşmektedir.

Bu nedenle, en iç çemberde değerlendirilen projeler çevik yaklaşımlar için muhtemel adaylardır, ikinci çemberdekiler (ancak iç çemberde olmayanlar) marjinaldir ve ikinci çemberin dışındakiler çevik yaklaşımlar için iyi adaylar değildir.

**Çevik Gereksinimlerin En İyi Uygulamaları**

Scott Ambler (2007), çevik yöntemler kullanan gereksinim mühendisliği için aşağıdaki en iyi uygulamaları önerir. Uygulamaların çoğu, doğrudan Çevik Manifesto'nun arkasındaki ilkelerden kaynaklanmaktadır:

◾Aktif paydaş katılımına sahip olmak

◾ Kapsayıcı (paydaş) modeller kullanın

◾ Genişliğe öncelik veren bir yaklaşım benimseyin

◾ "Fırtına" ayrıntılarını (son derece değişken gereksinimler) tam zamanında modelleyin

◾ Gereksinimleri uygulayın, belgelemeyin

◾ Bir noktaya kadar platformdan bağımsız gereksinimler oluşturun

◾ Küçüğün daha iyi olduğunu unutmayın

◾ Soru izlenebilirliği

◾ Teknikleri açıklayın

◾ Paydaş terminolojisini benimseyin

◾ Eğlenceye devam edin

◾ Yönetim desteği alın

◾ Paydaşları geliştiricilere dönüştürün

◾ Gereksinimlere öncelik verilmiş bir yığın gibi davranın

◾ Sık, kişisel etkileşimde bulunun

◾ Yazılımları sık sık teslim edin

◾ Gereksinimleri özellik olarak ifade edin

Ambler ayrıca gereksinimleri modellemek için CRC'ler, kabul testleri, iş kuralı tanımları, değişiklik durumları, veri akış şemaları, kullanıcı arayüzleri, kullanım senaryoları, prototipler, özellikler ve senaryolar, kullanım senaryoları ve kullanıcı hikayeleri gibi yapay nesnelerin kullanılmasını önerir (Ambler 2007). Bu unsurlar, kullanıcı hikayeleri ile birlikte yazılım gereksinimleri spesifikasyon belgesine eklenebilir.

Gereksinimlerin ortaya çıkarılması için görüşmeleri (hem yüz yüze hem de elektronik), odak gruplarını, JAD'yi, eski kod analizini, etnografik gözlemi, alan analizini ve müşterinin her zaman yerinde olmasını önerir (Ambler 2007). Bu bölümün geri kalanında, tartışmamız kullanıcı hikayelerinin kullanımına odaklanmaktadır. modeli gereksinimleri.

**XP’de Gereksinim Mühendisliği**

XP'deki gereksinim mühendisliği, Ambler'in modelindeki gereksinim yığınının kullanıcı hikayelerine atıfta bulunduğu Şekil 7.5'te gösterilen modeli takip eder. Ve XP'de kullanıcı hikayeleri yönetilir ve “planlama oyunu” aracılığıyla kod olarak uygulandı.

XP'deki planlama oyunu iki şekilde gerçekleşir: yayın ve yineleme planlaması. Yayın planlaması, bir dizi kullanıcı hikayesi yazıldıktan sonra gerçekleşir. Bu hikaye seti, genel proje planını geliştirmek ve yinelemeler için plan yapmak için kullanılır. Set ayrıca her bir kullanıcı hikayesi ve genel proje için yaklaşık programa karar vermek için kullanılır.

Yineleme planlaması, önceki yinelemelerdeki başarısız testlere yönelik bir dizi kullanıcı hikayesi ve düzeltmenin uygulandığı bir zaman dilimidir. Her yineleme 1-3 hafta sürer. Önceki yinelemelerden (proje hızı olarak adlandırılır) gelen kullanıcı hikayelerinin uygulanma oranını izlemek, geliştirme çizelgesini iyileştirmeye yardımcı olur.

Bu süreçler sırasında gereksinimler sürekli olarak geliştiğinden, XP'nin yaratıcısı Kent Beck, "XP'de gereksinimler bir belge değil bir diyalogdur" diyor (Beck ve diğerleri 2001), kullanıcı öyküleri yığınını bir yazılım gereksinimleri belirtimine dönüştürmek tipik olsa da.

***Yazılım ve Sistemler için Gereksinim Mühendisliği***

***Şekil 7.5 Çevik gereksinimler değişiklik yönetimi süreci. (Ambler, S., Agile gereksinimleri değişiklik yönetimi, 2007, http://www.agilemodeling.com/essays/changeManagement.htm'den uyarlanmıştır, son (Ocak 2017'de erişilmiştir).)***

**Scrum’da Gereksinim Mühendisliği**

Scrum'da, Şekil 7.5'teki modelde gösterilen gereksinim yığını, XP'de olduğu gibi, gelişen kullanıcı hikayeleri biriktirme listesidir. Ve XP'de olduğu gibi, bu gereksinimler geliştirme kararlılığı için her yinelemede dondurulur. Scrum'da her yineleme yaklaşık bir ay sürer. Yığındaki değişiklikleri yönetmek için, bir kişiye gereksinim önceliklendirmesi için son yetki verilir (genellikle ürün sponsoru).

Scrum'da gereksinim biriktirme listesi üç tipte düzenlenir: ürün, sürüm ve sprint. Ürün biriktirme listesi, sürüm biriktirme listelerini içerir ve her sürüm, sprint biriktirme listesini içerir. Şekil 7.6, biriktirme listesi kalemlerinin çevreleme ilişkisini gösteren bir Venn şemasıdır.

Ürün biriktirme listesi, bir noktada serbest bırakılması hedeflenen gereksinimler için bir havuz görevi görür. Ürün biriktirme listesindeki gereksinimler, düşük, orta ve yüksek düzey gereksinimleri içerir.

Sürüm biriktirme listesi, ürün biriktirme listesinden alınan öncelikli öğeler kümesidir. Yayın biriktirme listesindeki gereksinimler, daha fazla ayrıntı ve düşük seviyeli tahminler içerecek şekilde gelişebilir.

Son olarak, sprint iş listesi listesi, ekibin sprint sonunda tamamlayacağı (tamamen kodlanmış, test edilmiş ve belgelenmiş) bir dizi yayın gereksinimleridir. Bu gereksinimler çok yüksek bir ayrıntı düzeyine evrilmiştir ve bu nedenle öncelikleri yüksektir. Scrum, birçok büyük şirkette dikkate değer bir başarı ile benimsenmiştir.

Yazarın öğrencilerinden bazıları da derslerde Scrum kullanıyor. Bu durumlarda, uzun gereksinim keşif süreçleri için çok az zaman olduğunda oldukça etkili olduğunu kanıtlar.

***Gereksinimler Spesifikasyonu ve Çevik Metodolojiler***

***Şekil 7.6 Ürün, sürümler ve sprintler arasındaki iş listesi ilişkisi***

**Kullanıcı Hikayelerin Yazılması**

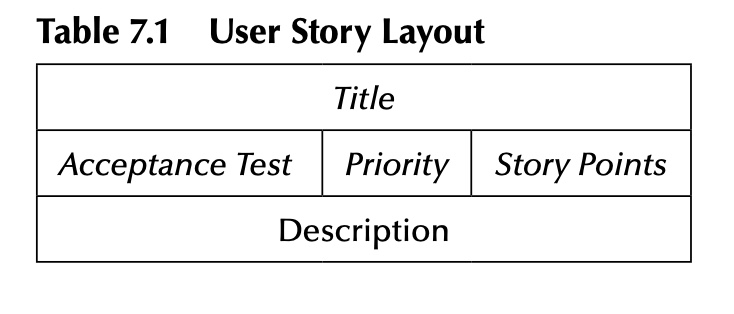
Kullanıcı hikayeleri, çoğu çevik metodolojide en temel gereksinim birimidir. Her kullanıcı hikayesi, müşteri tarafından istenen bir özelliği temsil eder. Kullanıcı hikayeleri (Kent Beck tarafından ortaya atılan bir terim) müşteri tarafından indeks kartlarına yazılır, ancak süreç wiki'ler veya diğer araçlar aracılığıyla otomatikleştirilebilir. Resmi gereksinimler, kullanım senaryoları ve diğer eserler, gerektiğinde yazılım mühendisliği ekibi tarafından kullanıcı hikayelerinden türetilir.

**Kullanıcı hikayesini oluşturan bileşenler:**

* + Başlık - bu hikaye için kısa bir tutamaçtır. Başlıkta aktif sesli bir şimdiki zaman fiili arzu edilir.
  + Kabul testi - bu, hikayeyi test etmek için bir yöntemin adı olacak benzersiz bir tanımlayıcıdır.
  + Öncelik - bu, benimsenen önceliklendirme şemasına dayanmaktadır. Öncelik, "geleneksel" önem önceliklendirmesine veya ayrıntı düzeyine göre atanabilir (daha yüksek öncelik, daha yüksek ayrıntıya atanır)
  + Hikaye puanı - bu, kullanıcı hikayesini uygulamak için tahmini süredir. Bu özellik, kullanıcı hikayelerini çaba ve maliyet tahmini için yararlı hale getirir.
  + Açıklama - bu, hikayeyi anlatan bir ila üç cümledir.

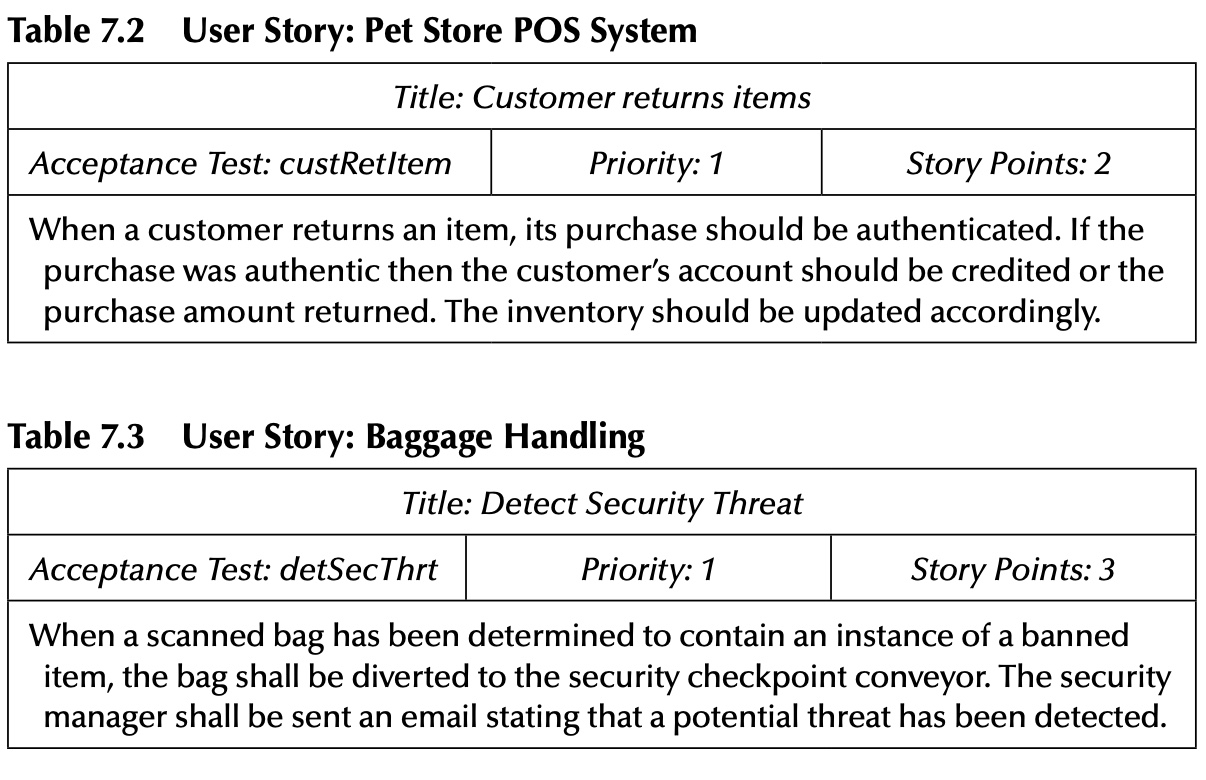
Bu öğeler için bir dizin kartındaki örnek yerleşim Tablo 7.1'de gösterilmektedir.

İlk kullanıcı hikayeleri genellikle küçük tesis dışı toplantılarda toplanır. Hikayeler, hedef odaklı (örneğin, “bir müşterinin nasıl satın aldığını tartışalım”) yaklaşımlarla veya etkileşimli (bilinç akışı) yaklaşımlarla oluşturulabilir. Kullanıcı hikayeleri geliştirmek “yinelemeli ve etkileşimli” bir süreçtir. Geliştirme ekibi aynı zamanda tek biçimlilik için öykülerin boyutunu da yönetir (örneğin, çok büyük—bölünmüş, çok küçük—birleştir).



Pet shop POS sisteminde ürün iade eden bir müşteri için örnek bir kullanıcı hikayesi Tablo 7.2'de gösterilmektedir. Kullanıcı hikayeleri müşteriler için anlaşılır olmalı ve her hikaye değer katmalıdır. Geliştiriciler kullanıcı hikayeleri yazmaz, kullanıcılar yazar. Ancak hikayelerin, yineleme başına birkaç tanesinin tamamlanabilmesi için yeterince küçük olması gerekir. Hikayeler bağımsız olmalıdır (mümkün olduğunca); yani, bir hikaye ileri geri başka hikayelere atıfta bulunmamalıdır. Son olarak, hikayeler test edilebilir olmalıdır - herhangi bir gereksinim gibi, test edilemiyorsa bu bir gereklilik değildir. Her hikayenin test edilebilirliği geliştirme ekibi tarafından değerlendirilir.

Tablo 7.3, havalimanı bagaj taşıma sisteminde bir güvenlik tehdidi algılamasını açıklayan başka bir örnek kullanıcı hikayesini göstermektedir. Son olarak, kullanım senaryoları ile kullanıcı hikayeleri arasında önemli bir fark olduğunu belirtmekte fayda var. Kullanıcı hikayeleri müşteri perspektifinden gelir ve basittir ve uygulama ayrıntılarından kaçınır. Kullanım senaryoları daha karmaşıktır ve uygulama ayrıntılarını içerebilir (örneğin, fabrikasyon nesneler). Müşteriler genellikle kullanım senaryoları yazmazlar (ve yazarlarsa dikkatli olun çünkü müşteri artık “yazılım mühendisliği” ile uğraşmaktadır). Son olarak, kullanıcı hikayesi başına kullanım vakası sayısı için eşdeğerliğin ne olduğunu söylemek zor. Bir kullanıcı hikayesi, bir veya 20'den fazla kullanım örneğine eşit olabilir.



Örneğin, Tablo 7.2'deki müşteri iade kullanıcı hikayesi için, bir müşteri iadesinde ortaya çıkabilecek çeşitli durumlarla başa çıkmanın çok daha fazla kullanım senaryosu gerektireceğini hayal edebilirsiniz. Çevik metodolojilerde, kullanıcı hikayeleri vakaları kullanmaktan daha çok tercih edilir.

Ek D, hem çevik hem de çevik olmayan ayarlarda kullanıcı hikayelerinin daha ayrıntılı bir tartışmasını içerir.

**Çevik Gereksinim Mühendisliği**

Çevik metodolojiler için gereksinim mühendisliği ile çevik gereksinim mühendisliği arasında bir ayrım yapmamız gerekiyor. Çevik gereksinim mühendisliği, genel olarak, geleneksel gereksinim mühendisliğinden daha esnek olduğu iddia edilen herhangi bir geçici gereksinim mühendisliği yaklaşımı anlamına gelir. Bu tanım, az önce tartıştığımız gibi, çevik metodolojilerdeki gereksinim mühendisliği için özel uygulamalarla karıştırılmamalıdır (Sillitti ve Succi 2006). Son birkaç yılda, çoğu özensiz gereksinim mühendisliğinden daha fazla olmayan bir dizi çevik gereksinim mühendisliği yaklaşımı tanıtıldı. Ancak bu alandaki son çalışmaların bazıları da iyi oldu. Bununla birlikte, karşılaşabileceğiniz herhangi bir "meşru" çevik gereksinim mühendisliği metodolojisi için uygulamaların çoğu, Çevik Manifesto'ya kadar izlenebilir. Örneğin, Vlaanderen ve ark. (2011), Scrum'daki belirli gereksinim mühendisliği uygulamalarının nasıl yazılım ürünü yönetimi için de (teslimattan sonra) olabileceğini gösterdi. Özellikle sprint döngülerini, biriktirme listesi yönetimini, günlük toplantıları ve erken ve sık işbirliğini gerekli uygulamalar olarak belirlediler.

**HİKAYE-TEST DAYALI GELİŞTİRME**

**Story Test-Driven Development**

Çevik bir metodolojiyi göstermek için dikkate değer bir örneği açıklıyoruz. Hikaye testi güdümlü geliştirme (SDD) olarak adlandırılan bu metodolojide, çevik metodolojilerin olağan müşteriye yönelik özelliklerinin çoğu, kısa patlamalarla birlikte dahil edilmiştir. XP veya Scrum için yinelemeli geliştirme. Bununla birlikte, SDD'deki bir fark,geleneksel kullanıcı hikayeleri yerine müşteriler “hikaye testleri” yazar veya gözden geçirir. Öykü testleri, davranışın teknik olmayan tanımlarıyla birlikte doğrulayan testlerdir.Davranışın doğru olmasıdır. Öykü testleri “birçok eksik parçayı keşfetmeye veya bir hikayedeki tutarsızlıklar” (Mugridge 2008).

Hikaye testinin güzel bir özelliği, kullanıcıların aşağıdakileri yapmasına izin vermek için Fit çerçevesini kullanmasıdır. Öykülere tablo şeklinde test senaryoları oluşturun (bkz. Bölüm 8). Bu nedenle, kullanıcılar gereksinimlerde davranışı ve bu davranış için testleri sezgisel olarak belirtin.

Örneğin, bir işletme için tipik bir bordro sistemi için (evcil hayvan mağazası gibi), bir çalışan için normal ve fazla mesai ücretinin nasıl hesaplanacağını açıklayan bir hikaye testi çeşitli saatlere dayalı olarak Şekil 7.7'de verilmiştir.

Şekil 7.7'deki tablo, örneğin, bir çalışan arka arkaya 5 iş günü 9 saat çalışıyorsa, normal çalışma saatinde 45 saat ve fazla mesai ücretinin sıfır saat olarak kabul edildiğini göstermektedir. Bir işçi 10, 11, 12, 13 ve 14 saatlik günleri izliyorsa bir hafta içinde 45 saat normal ve 15 saat fazla mesai hakları vardır.Ödemek ancak Fit çerçevesi etkileşimli bir özelliktir; farklı tabloya yanlış olan değerler geçersiz olarak görünecektir (bkz. Bölüm 8).SDD geliştirme, test odaklı geliştirmenin (TDD) tamamlayıcısı olarak kabul edilir, çünkü birincisi genel sistem geliştirmeye uygulanır (Mugridge 2008).

**GEREKSİNİM MÜHENDSİLİĞİ İÇİN ÇEİK METEDOLOJİLERİN ZORLUKLARI**

Elbette, herhangi bir yeni teknolojinin zorlukları vardır ve bazı zorluklar da vardır.Özellikle gereksinim mühendisliği ile ilgili olarak çevik metodolojiler kullanmak. Williams (2004) bu eksikliklerden bazılarını tartışıyor. Örneğin, çevik metodolojiler her zaman işlevsel olmayan durumlarla iyi başa çıkmaz. Gereksinimler bu neden böyle olmalı? Bunun bir nedeni, yalnızca gereksinim işlevselliği ile uğraşırken (kullanıcı hikayeleri aracılığıyla) bunların her zaman belirgin olmamasıdır.

Williams, kullanıcı hikayelerini aşağıdakilerle artırarak bu zorlukla başa çıkmayı önerir:

Rekabetçi gibi uygun işlevsel olmayan gereksinimleri keşfetme teknikleri analiz. Diğer bir eksiklik ise, çevik metodolojilerle müşteri etkileşiminin güçlüdür - ancak çoğunlukla prototipleme yoluyla. Gördüğümüz gibi, başka yollar da var. Nüanslı gereksinimleri ortaya çıkarmak ve paydaş ihtiyaçlarını anlamak

Örneğin, çeşitli görüşme tekniklerinin kullanılması arzu edilir. Ayrıca, çevik metodolojilerle doğrulama, test yoluyla güçlüdür ve prototipleme, ancak doğrulama o kadar güçlü değil. Williams (2004) şunu önermektedir:

Biçimsel yöntemler kullanmak, çevik gereksinim mühendisliğini güçlendirebilir (2004). Rubin ve Rubin (2011), sıklıkla kullanılan dokümantasyon yapılarını tanımladı. çevik yazılım geliştirmede kaçırıldı ve bunu hafifletmenin yollarını önerdiler sorun.

Özellikle, alan bilgisinin ve ortaya çıkan sistem mimarisi ve tasarımının geleneksel çevik gereksinim uygulamaları tarafından uygun şekilde yakalanmadığını belirttiler. Bulguları, çevik gereksinim belgelerinin uygun alan bilgisi ve ilgili çevik ile açıklamalı olmalıdır mimari ve tasarım belgelerine nihai sistem mimarisi ile açıklama eklenebilir ve tasarım sırasıyla : Gereksinim yönetimi sürece dahil edilmiştir (örneğin, XP ve Scrum), ancak çoğunlukla kod düzeyine odaklanır. Williams, daha fazla standart konfigürasyon yönetimi ekleyerek gereksinim yönetiminin güçlendirilebileceğini öne sürüyor uygulamalar. Son olarak, Inayat ve ark. (2015) bir dizi çevik gereksinim mühendisliğini özetledi zorluklar. Inayat ve diğerleri tarafından belirlenen zorluklar şunlardı:

◾ İşlevsel olmayan gereksinimlerin ihmal edilmesi

◾ Gereksinim izlenebilirliğinin olmaması

◾ Yanlış gereksinim önceliklendirmesi

◾ Minimum gereksinim belgeleri

◾ Sözleşme sorunları

◾ Müşteri mevcudiyeti

◾ Müşteri sözleşmesi

Bu zorluklar, başkaları tarafından tanımlananlardan bazılarını tekrarladı. Çevik metodolojileri kullanmada hala önemli zorluklar olsa da yazılım mühendisliği için de önemli avantajlar var. Bu nedenle, çevik yazılım metodolojileri en azından çoğu proje için düşünülmelidir. Ve ne zaman çevik metodolojiler belirli yazılım projeleri için uygun görülmemektedir veya proje yazılım olmadığında, çevik metodolojiler için tartışılan uygulamaların çoğu kullanılabilir. Örneğin müşteri her zaman sahada, odak noktası gibi uygulamalar ürün üzerinde dokümantasyon ve test senaryolarının erken gelişimi, yine de hemen hemen her projeye dahil edilmiştir.

**UYGUN BAKIM YASASI WEB SİTESİ**

2010 yılında Amerika Birleşik Devletleri Uygun Bakım Yasasını kabul etti. Tüm vatandaşların bir tür sağlık sigortası almalarını şart koşuyor. Yasa tartışmalıydı ve tartışmalara ek olarak federal hükümetin feci bir şekilde piyasaya sürülmesiydi. İlgili web sites proje karmaşık ve ilgiliydi 55 müteahhit. Sağlam bir gelişmeye odaklanmak yerine sürecinde, web sitesini dağıtmak için bir acele oldu ve şaşırtıcı olmayan bir şekilde birçok sorunla karşılaştı. Çok fazla vardı yapısal ve güvenlik sorunları ancak kullanılabilirlik sorunları gibi sık kilitlenme, yetersiz yanıt süreleri, eksik veya yanıltıcı içerik ve gezinme zorlukları en çok dikkat (Venkatesh ve ark. 2014). Yaklaşık 9,47 milyon kullanıcı konuşlandırmanın ilk haftasında kaydolmaya çalıştı ancak sadece 271.000 başarılı oldu (Cleland-Huang 2014). Çalışmaları web sitesinin başarısız lansmanı, çeşitli temel nedenlere işaret etti proje takviminin eksik tahmin edilmesi, kötü tanımlanmış kapsam, zayıf gereksinimler spesifikasyonu ve verimsiz risk analizi ve yönetim (Anthopoulos ve ark. 2016). Birçok uzman daha iyi bir geleneksel gereksinim sürecinin bu sorunları azaltacağını doğru bir şekilde öne sürse de sorunlar, siteyi hızlı bir şekilde dağıtma baskısı muazzamdı. Bununla birlikte, çevik, dinamik bir gereksinim olması muhtemeldir. Keşif ve geliştirme süreci daha iyi sonuçlara yol açabilirdi izin verirken proje sonuçları ve kamuoyu algısın önceki dağıtım. Böyle bir sürecin yaşandığı bildirilmemiştir. Önerilmiş veya düşünülmüştür.