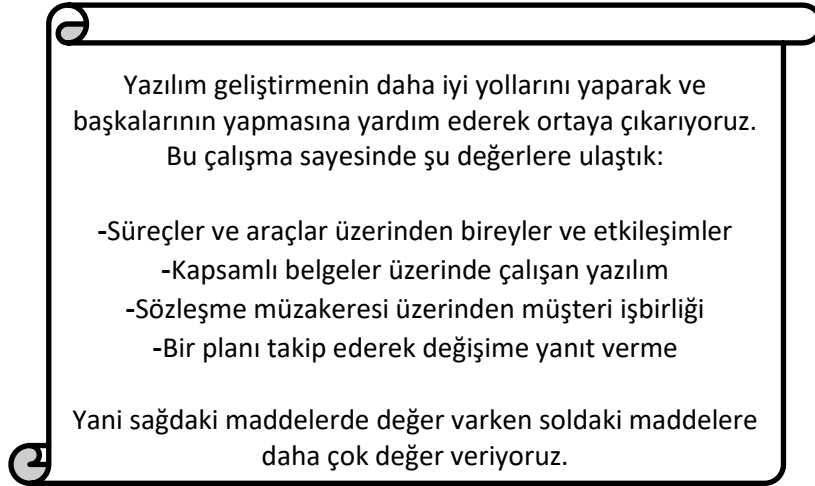


GEREKSİNİM BELİRTİMLERİ VE ÇEVİK METODOLOJİLER

ÇEVİK METODOLOJİLERE GİRİŞ

Çevik metodolojiler, geleneksel ve süreç yüklü yaklaşımlara karşı temkinli olan birçok kişinin hayal gücünü yakalayan, geleneksel olmayan yazılım geliştirme stratejileri ailesidir. Çevik metodolojiler, katı süreç eksikliği ile karakterize edilir, ancak bu gerçek, çevik metodolojilerin doğru bir şekilde kullanıldığında titiz veya endüstriyel uygulamalar için uygun olmadığı anlamına gelmez - öyledir. Bununla birlikte, çevik yaklaşımlarda karakteristik olarak eksik olan şey, zorunlu toplantılara odaklanan “yemek kitabı” çözümleri ve karmaşık dokümantasyon tarafından öngörülen geliştirme yaklaşımlarıdır.

Çevik metodolojiler yazılım mühendisliği için geçerlidir. Sistem mühendisliğine uygulanabilecek çevik metodoloji unsurları olsa da (özellikle insan değerlendirmeleri), bu tür metodolojiler yazılım olmayan sistemlere uygulandığında genellikle hafif veya yalın olarak tanımlanır. Bunun nedeni, çevik metodolojilerin, donanım tabanlı sistemlerde pratik olmayan bir yaklaşım olan bir dizi hızlı, atılmayan prototipe bağlı olmasıdır. Her durumda, çevik metodolojiler giderek daha fazla kullanıldığından ve çevik yazılım mühendisinin zihniyeti bazı sağlıklı bakış açıları içerdiğinden, yazılım dışı mühendisler bu bölümden yine de yararlanabilir.



Şekil 7.1 Çevik yazılım geliştirme için Manifesto. (Beck, K.'den, Extreme Programming Açıklaması: Embrace Change, Longman Higher Education, Londra, 2000.)

Çevik metodolojilerin doğasını tam olarak anlamak için Çevik Manifesto adlı bir belgeyi ve arkasındaki ilkeleri incelememiz gerekiyor. Çevik Manifesto, çevik metodolojilerin önde gelen birkaç savunucusu tarafından felsefelerini açıklamak için tanıtıldı (bkz. Şekil 7.1).

Çevik Manifesto'yu imzalayanlar arasında Kent Beck, Mike Beedle, Alistair Cockburn, Ward Cunningham, Martin Fowler, Jim Highsmith, Ron Jeffries, Brian Marick, Robert Martin, Steve Mellor ve Ken Schwaber gibi modern yazılım mühendisliği uygulamalarının önde gelen isimleri yer alıyor. Çevik Manifesto'nun altında yatan bir dizi ilkedir. İtaliye olarak

belirlediğimiz gereksinim mühendisliğine odaklanan yönlelere vurgu yaparak aşağıdaki ilkelere bakın.

ÇEVİK MANİFESTO'NUN ARKASINDAKİ İLKELER

- Ekip, düzenli aralıklarla nasıl daha etkili olunacağını düşünür, ardından davranışını buna göre ayarlar ve ayarlar.
- En yüksek önceliğimiz, değerli yazılımları erken ve sürekli teslim ederek müşteriye memnun etmektir.
- Geliştirmede geç olsa bile değişen gereksinimleri memnuniyetle karşılayın. Çevik süreçler, müşterinin rekabet avantajı için değişimden yararlanırlar.
- Daha kısa zaman ölçeğini tercih ederek, çalışan yazılımı birkaç haftadan birkaç aya kadar sık sık teslim edin.
- İş adamları ve geliştiriciler, proje boyunca günlük olarak birlikte çalışmalıdır.
- Motive olmuş bireyler etrafında projeler oluşturun. Onlara ihtiyaç duydukları ortamı ve desteği verin ve işi tamamlamaları için onlara güvenin.
- Bir geliştirme ekibine ve içinde bilgi aktarmanın en verimli ve etkili yöntemi yüz yüze görüşmedir.
- Çalışan yazılım, ilerlemenin birincil ölçüsüdür.
- Çevik süreçler sürdürülebilir kalkınmayı teşvik eder. Sponsorlar, geliştiriciler ve kullanıcılar, süresiz olarak sabit bir hızı sürdürebilmelidir.
- Teknik mükemmelliğe ve iyi tasarıma verilen sürekli dikkat, çevikliği artırır.
- Sadelik - yapılmayan iş miktarını en üst düzeye çıkarma sanatı - esastır. "İşe yarayabilecek en basit şeyi yap."«
- En iyi mimariler, gereksinimler ve tasarımlar kendi kendini organize eden ekiplerden ortaya çıkar (Beck 2000).

İlkelerin, gereksinimlerin süreç boyunca değiştiği fikrini nasıl kabul ettiğine ve benimsediğine dikkat edin. Ayrıca, çevik ilkeler sık, kişisel iletişimi vurgular (bu özellik, yazılım dışı sistemlerin mühendisliğinde de faydalıdır). Çevik süreç modellerinde gereksinim mühendisliğinin vurgulanan özellikleri, "geleneksel" şaleden ve yinelemeli, evrimsel veya spiral geliştirme gibi daha modern modellerden farklıdır. Bu diğer modeller, gereksinim mühendisliği süreci ve genellikle hacimli gereksinim belirtim belgelerinin üretimi üzerinde çok sayıda ön çalışmayı destekler.

ÇEVİK YAZILIM GELİŞTİRMENİN FAYDALARI

Çevik yazılım geliştirme yöntemleri, değişimi benimsemeye odaklanan ve kaliteyi korurken işbirliğini ve erken ürün teslimatını vurgulayan yinelemeli yöntemlerin* bir alt kümesidir. Çalışma kodu, geliştirme sürecinin gerçek eseri olarak kabul edilir. Modeller, planlar ve belgeler önemlidir ve değerleri vardır, ancak daha önce tartışılan diğer yaklaşımların aksine, yalnızca çalışan yazılımın geliştirilmesini desteklemek için vardır. Ancak bu, çevik bir geliştirme yaklaşımının herkes için ücretsiz olduğu anlamına gelmez. Çevik metodolojistlerin benimsemesi gereken çok net uygulamalar ve ilkeler vardır.

*Çoğu insan çevik metodolojileri artımlı olarak tanımlar. Ancak artımlı geliştirme, teslim edilen her sürümün özelliklerinin ve zamanlamasının planlandığı anlamına gelir. Bazı durumlarda, çevik metodolojiler, neredeyse her zaman planlandığı gibi olmayan özellik setlerine ve teslim tarihlerine sahip sürümlere yol açma eğilimindedir.

Çevik yöntemler öngörücü olmaktan çok uyarlanabilirdir. Bu yaklaşım, yazılımın uzun bir süre boyunca ayrıntılı olarak planlanmasını vurgulayan ve yazılım gereksinimleri belirtimindeki önemli değişikliklerin sorun yaratabileceği süreç modellerinden önemli ölçüde farklıdır. Çevik yöntemler, başlangıçta ağırlıklı olarak dokümantasyona odaklanan daha “törenselsel” ön tasarım yaklaşımlarını boğabilecek, sürekli değişen gereksinimlere ilişkin yaygın soruna bir yanıttır.

Çevik yöntemler ayrıca süreç odaklı olmaktan çok insan odaklıdır. Bu, geliştirmeyi "eğlenceli" hale getirmeye çalıştıkları anlamına gelir. Muhtemelen bunun nedeni, yazılım gereksinimleri spesifikasyonlarının ve yazılım tasarım açıklamalarının yazılmasının zahmetli olması ve dolayısıyla en aza indirilmesidir.

Inayat et al. 2002 ve 2012 yılları arasında yayınlanan çevik yazılım projelerinin çalışmalarının sistematik bir incelemesini yaptılar. Çevik metodolojilerde bulunan gereksinim mühendisliği uygulamaları tarafından ortadan kaldırılan veya en aza indirilen beş geleneksel gereksinim mühendisliği zorluğu belirlediler. Bu zorluklar şunlardı:

- Tüm paydaşlarla iletişim boşluğunu kapatmak
- Müşteri katılımını artırmak
- Dokümantasyon boyutunun ve karmaşıklığının azaltılması
- Kapsam kaymasını azaltma
- Gereksinimlerin doğrulanmasını sağlamak (Inayat et al. 2015)

Bu sonuçları güçlendirmek ve genelleştirmek için daha fazla araştırma ve ampirik kanıta ihtiyaç olduğunu belirttiler, ancak çalışmalarının sonuçları hala çevik yöntemlerin gücüne güçlü bir şekilde işaret ediyor

Çevik metodolojiler arasında Crystal, Extreme Programming, Scrum, Dinamik sistem geliştirme yöntemi (DSDM), özellik odaklı geliştirme ve uyarlanabilir programlama yer alır ve diğerleri vardır. En yaygın kullanılan çevik metodolojilerden ikisine daha yakından bakacağız: XP ve Scrum.

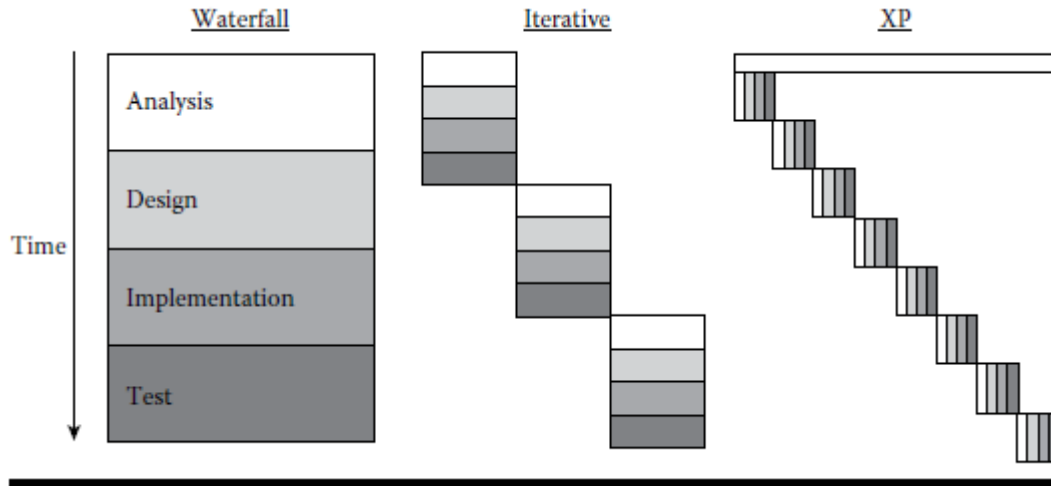
EXTREME PROGRAMMING

Extreme Programming (XP)*, en yaygın kullanılan çevik metodolojilerden biridir. XP geleneksel olarak daha küçük geliştirme ekiplerine yöneliktir ve nispeten az sayıda ayrıntılı eser gerektirir. XP, geliştirme döngülerine yinelemeli bir yaklaşım getiriyor. Geleneksel bir şelale ile Model, yinelemeli modeller ve XP arasındaki süreç farkını görselleştirebiliriz(Şekil 7.2). Oysa evrimsel veya yinelemeli yöntemin hala farklı gereksinim analizi, tasarımı, uygulaması ve şelale yöntemine benzer test aşamaları, XP bu faaliyetleri birbiriyle ilişkili ve sürekli.

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

XP, geliştiricilerin yanıt vermesine yardımcı olan bir dizi 12 temel uygulamayı destekler ve kaçınılmaz değişimi benimseyin. Uygulamalar dört uygulamaya göre gruplandırılabilir alanlar:

- Planlama
- Kodlama
- Tasarım
- Test



Şekil 7.2 Şelale, yinelemeli ve XP geliştirme döngülerinin karşılaştırması. (İtibaren Beck, K., Aşırı Programlama Açıklaması: Değişimi Kucakla, Longman Higher Eğitim, Londra, 2000.)

XP'nin ayırt edici planlama özelliklerinden bazıları, günlük stand-up toplantıları düzenlemeyi,* sık sık küçük yayınlar yapmayı ve insanları hareket ettirmeyi içerir.Önce durumlar ve ikili programlamanın kullanılması (iki geliştiricinin aynı kod üzerinde birlikte çalışır. Herhangi bir kod biriminin bölgesel mülkiyetinin kaldırılması XP'nin bir başka özelliğidir.

Tasarım uygulamaları, önce en basit çözümleri aramayı, aynı zamanda kaçınmayı içerir. Gelecekteki büyüme için çok fazla planlama (spekülatif genellik) ve kodu sürekli olarak yeniden düzenlemeyi(yapısını iyileştirmeyi) içerir.

Test uygulamaları, bir hata bulunduğunda yeni test durumları oluşturmayı ve muhtemelen XUnit gibi çerçeveleri kullanarak tüm kodlar için birim testleri yapmayı içerir.

SCRUM

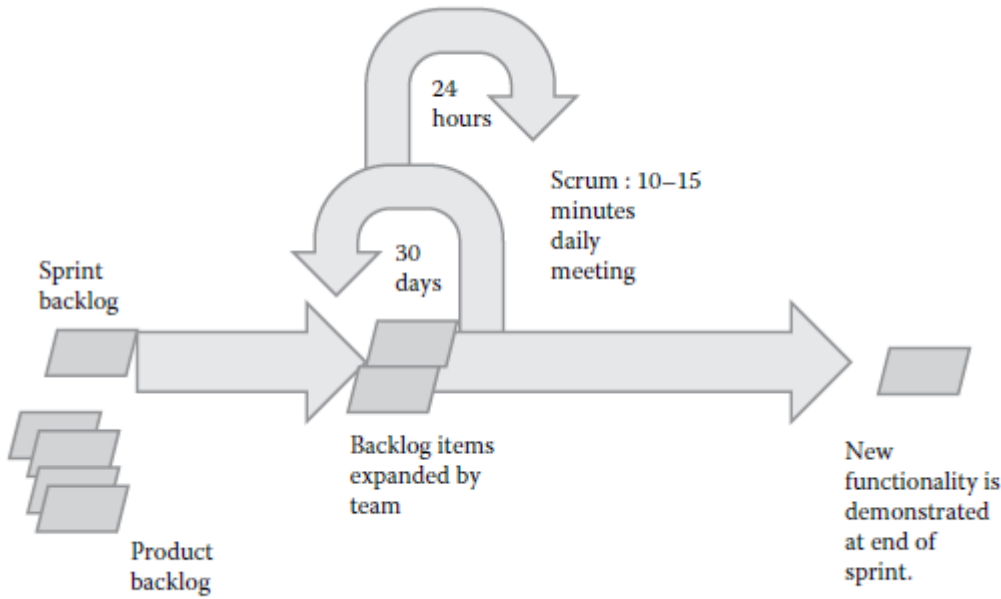
Scrum Bir ragbi maçında özellikle çekişmeli bir noktadan sonra adlandırılan Scrum, tüm ekip üyeleri ve tüm paydaşlar arasında sözlü iletişimi teşvik ederek kendi kendini organize eden takımlara olanak tanır. Scrum'un temel ilkesi, şelale ve yinelemeli geliştirme gibi geleneksel, plan odaklı yazılım geliştirme metodolojilerinin sürece çok fazla odaklanması ve yazılıma yeterince odaklanmamasıdır. Ayrıca, plan odaklı geliştirme yazılım dışı eserler (ör. dokümantasyon) ve süreçlere (ör. resmi incelemeler) odaklanırken Scrum, işleyen yazılımların erken ve sık üretilmesinin önemini vurgulamaktadır. Scrum teşvik eder.

* Stand-up toplantıları 10 dakika veya daha kısa sürelidir; burada katılımcılar tam anlamıyla ayağa kalkar ve bir önceki günün etkinlikleri ve o güne ilişkin planları hakkında sözlü bir rapor verir.

Tüm paydaşlar arasında yüksek kaliteli iletişimi öz-örgütlenme teşvik eder. Bu durumda, sorunun tam olarak anlaşılamayacağı veya tanımlanamayacağı örtük bir durumdur (kötü bir sorun olabilir). Ve Scrum'daki odak, ekibin ortaya çıkan zorluklara çevik bir şekilde cevap verme yeteneğini en üst düzeye çıkarmaktır.

Scrum, yapılması gereken öncelikli çalışmalardan oluşan bir yaşam birikimine sahiptir. Büyük ölçüde sabit bir bekleme listesi ögesi kümesinin tamamlanması, bir dizi kısa (yaklaşık 30 gün) yineleme veya sprintte gerçekleşir. Her gün, ilerlemenin açıklandığı, yaklaşımda olan çalışmaların anlatıldığı ve engellerin yükseltildiği kısa (örneğin, 15 dakikalık) bir toplantı veya Scrum düzenlenir. Tamamlanacak bekleme listesi öğelerini tanımlamak için her sprintin başında kısa bir planlama oturumu oluşur. Sprintin sonunda kısa bir postmortem veya kalp atışı retrospektif incelemesi gerçekleşir (Şekil 7.3).

Bir Scrum Ustası, her sprintin önündeki engeli veya engelleri kaldırır. Scrum Master, ekibin lideri değildir (kendi kendini organize ettikleri için), ancak ekip ile istikrarsızlaştırıcı etkiler arasında bir verimlilik tamponu görevi görür. Bazı organizasyonlarda Scrum Ustasının rolü karışıklığa neden olabilir. Örneğin, bir Scrum ekibinin iki üyesi birlikte iyi çalışmıyorsa, Scrum Yöneticisinin sorunu çözmesi üst düzey bir yönetici tarafından beklenebilir. Takım işlev bozukluğunu düzeltmek Scrum Ustasının rolü değildir. Personel sorunlarının, ilgili tarafların rapor verdiği hat yöneticileri tarafından çözülmesi gerekir. Bu senaryo, bu tür yaklaşımların kullanılacağı durumlarda çevik metodolojiler hakkında kurum çapında eğitime duyulan ihtiyacı göstermektedir.



Şekil 7.3 Boehm ve Turner'dan (2003) Scrum geliştirme süreci. (Schwaber, K. ve Beedle, M.'den uyarlanmıştır, Agile Software Development with SCRUM, Prentice Hall, Upper Saddle River, NJ, 2001.)

ÇEVİK METODOLOJİLER İÇİN GEREKSİNİM MÜHENDİSLİĞİ

Çevik Metodolojiler için Gereksinimler Mühendisliği Geleneksel (burada şelale, evrimsel, yinelemeli, spiral vb.) Arasındaki büyük fark.) ve çevik metodolojiler gereksinimlerin toplanmasıdır. Aslında, çevik metodolojilerin bazı savunucuları, çevik gereksinimler mühendisliği uygulamalarının sözde avantajlarını çevik yöntemler için bir satış noktası olarak kullanmaktadır. Gereksinimler Çevik metodolojiler için mühendislik yaklaşımları çok daha gayri resmi olma eğilimindedir.

Diğer bir fark, gereksinim mühendisliği faaliyetlerinin zamanlamasıdır. Geleneksel sistemlerde ve yazılım mühendisliğinde gereksinimler toplanır, analiz edilir, rafine edilir vb. Çevik yöntemlerde gereksinim mühendisliği devam eden bir faaliyettir; yani gereksinimler her sistem yapısı ile rafine edilir ve keşfedilir. Gereksinimlerin iyileştirilmesi için prototiplemenin kullanıldığı spiral metodolojilerde bile, gereksinim mühendisliği geliştirme sürecinde çok daha geç gerçekleşir.

Müşteriler, çevik yöntemlerde gereksinimlerin keşfine ve iyileştirilmesine sürekli olarak katılırlar. Tüm sistem geliştiricileri gereksinim mühendisliği faaliyetinde yer alır ve her biri müşterilerle düzenli etkileşime girebilir ve olmalıdır. Geleneksel yaklaşımlarda, gereksinim belirtimi yazıldıktan ve onaylandıktan sonra müşteri daha az katılım gösterir ve genellikle katılım genellikle sistem geliştiricileri ile değildir.

Muhtemelen, gereksinim mühendisliğine çevik yaklaşım, süreç boyunca değişikliklere karşı geleneksel yazılım mühendisliğinden çok daha yenilmezdir.(unutmayın, “değişimi benimseyin”).

ÇEVİK METODOLOJİLERDE GENEL UYGULAMALAR

Cao ve Ramesh (2008) 16 yazılım geliştirme organizasyonu üzerinde çalışmışlardır. XP veya Scrum (veya her ikisini) kullanarak ve yedi gereksinim mühendisliğini ortaya çıkardı.

1. Yüz yüze iletişim (yazılı özellikler üzerinden)
2. Yinelemeli gereksinim mühendisliği
3. Aşırı önceliklendirme (başlangıçta bir kereden ziyade sürekli önceliklendirme, ve önceliklendirme öncelikle iş değerine dayalıdır)
4. Sürekli planlama
5. Prototipleme
6. Test odaklı geliştirme
7. İncelemeler ve testler

Bu uygulamalardan bazıları çevik olmayan geliştirmede bulunabilir (ör.geliştirme ve prototip oluşturma), ancak bu yedi uygulama hepsinde tutarlıydı.

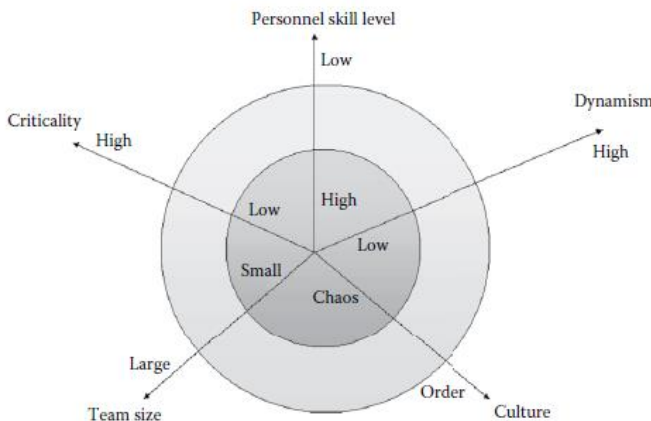


Figure 7.4: Balancing agility and discipline. (Adapted from Boehm, B., and Turner, R., *Balancing Agility and Discipline: A Guide to the Perplexed*, Addison-Wesley, Boston, 2003.)

Temel yazılım gereksinimleri spesifikasyonunun aksine, çevik yöntemlerdeki temel yapı, sürekli gelişen ve rafine edilen bir yığındır. Bu gereksinimler genellikle kullanıcı hikayeleri şeklindedir. herhangi birinde durumda, bu gereksinimler müşteri tarafından oluşturulur ve önceliklendirilir - daha yüksek gereksinimdeki ayrıntı düzeyi ne kadar yüksek olursa, öncelik o kadar yüksek olur. Yeni gereksinimler olarak keşfedilir, yığına eklenirler ve yığın yeniden karıştırılır (Şekil 7.4).

Gereksinimlerin eklenmesi, değiştirilmesi veya kaldırılması konusunda herhangi bir yasak yoktur. Herhangi bir zamanda liste (müşteriye muazzam bir özgürlük verir). Elbette, sistem oluşturulduktan sonra veya muhtemelen inşa edilirken, kullanıcı hikayeleri yığını sistem için geleneksel bir yazılım gereksinimleri spesifikasyonuna dönüştürülebilir bakım ve diğer geleneksel amaçlar.

ÇEVİK YAZILIM GELİŞTİRME ÖRNEK UYGULAMASI

Evcil hayvan mağazası satış noktası sistemini düşünün. Bir çevik kullanmak istediğimizi varsayalım.

Bu sistemi geliştirmek için yazılım geliştirme metodolojisi, özellikle Scrum.

Gereksinim mühendisliği sürecinin nasıl görünebileceğine bakalım.

Scrum da dahil olmak üzere beş kişilik bir geliştirme ekibimiz olduğunu varsayalım.

Usta. Basit olması için, tek paydaşın evcil hayvan mağazası olduğunu varsayalım.

sahibi (sistem için ödeme yaptığı için gerçekten müşteridir), mağaza müşterileri, kasiyerler ve muhasebeciler. Mağaza için sınıf temsilcilerini seçeceğiz

müşteriler, kasiyerler ve muhasebeciler. Toplu olarak, onlara "paydaş" diyelim.

panel." Şimdi aşağıdaki faaliyetleri düşünün.

1. Scrum Master, geliştirme arasında bir dizi toplantı düzenler.

hakkında temel bir anlayış elde etmek için ekip ve müşteri (mağaza sahibi)

sistem gereksinimleri, kısıtlamalar ve temel kurallar. Bu tartışmalara dayanarak, Scrum Master bir dizi ek bilgi toplama etkinliği düzenler,

muhtemelen yapılandırılmış görüşmeler, kart sıralama ve odak grupları dahil

diğer paydaşlarla. Doğrulama faaliyeti olarak bir QFD çalışması kullanılır

(muhtemelen karşılaştırılabilecek birçok başka POS sistemi olduğu için). bu

Bu faaliyetlerin amacı, bir dizi öncelikli kullanıcı hikayesi üretmektir (diyelim ki

yaklaşık 100), her biri için efor tahminleri dahil. Ama müşteri söyledi

bize "son teknoloji" bir sistem istediğini söyledi, bu yüzden sistem ilerledikçe biliyoruz

geliştirme yoluyla, müşteri ve diğer paydaşlar

yeni veya farklı işlevsellik.

2. 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.

3. Geliştirme ekibi, daha fazla ilerlemek için paydaşlara tekrar danışır

Geliştirmeye başlamadan önce bu 10 kullanıcı hikayesi için netlik.

4. Her gün bir saldırı ile başlar. Geliştiriciler, sistem tasarımını ve birim test durumlarını

organik olarak türetmek için test odaklı tasarımı kullanır. Scrum Ustası

geliştirme ekibi ve çeşitli ekipler arasındaki iletişimi kolaylaştırır
Paydaşların yüz yüze olduğu soruların hızlı bir şekilde yanıtlanabilmesi için paydaşlar.
Yaklaşık 30 gün sonra ilk sprint tamamlanır.

5. Paydaş paneli, sprint sırasında oluşturulan sistemle birlikte sunulur
geri bildirim için. Bu geri bildirim, mevcut sistem artışıdaki değişiklikleri, eklenecek yeni
özellikleri ve muhtemelen atlanacak özellikleri içerecektir.

6. Bu geri bildirimle dayanarak geliştirme ekibi yeni kullanıcı hikayeleri oluşturur ve
bunları biriktirme listesine ekler, diğer kullanıcı hikayelerini gerektiği gibi değiştirir ve
biriktirme listesini yeniden düzenler.

7. Geliştirme ekibi, sonraki 30 günlük sprint için backlog'dan yeni bir kullanıcı hikayesi seti
(10 diyalim) seçer ve süreç adımları (C'den F'ye) tekrarlanır birikim bitene kadar.

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

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

ÇEVİK NE ZAMAN ÖNERİLİR?

Çevik yazılım metodolojilerinin kullanımı endüstride çok popüler hale geliyor.
en azından anket verileriyle kanıtlandığı gibi. Örneğin, bir Forrester/Dr. Dobbs geliştiricisi
Ankete göre, 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 buldu. Ayrıca çevik
yöntemlerin Amerika Birleşik Devletleri'nde şelale gelişimi kadar iki kat daha popülerdi
ve Amerika Birleşik Devletleri dışında yedi kat daha popüler. Ayrıca çevik buldular
yazılım geliştirme her bölgede şelaleden daha sık kullanılmıştır.
Amerika Birleşik Devletleri, kuzeydoğu hariç, yaklaşık olarak eşit olarak kullanılırlar. En
sonunda, araştırma, çevik yöntemlerin şelaleden daha sık kullanıldığını buldu
finans, bankacılık ve sigorta hariç her sektörde stil geliştirme. Daha da önemlisi, çalışma,
katılımcıların daha fazla memnuniyet ifade ettiğini buldu.çevik geliştirmeyi kullanırken
gereksinim mühendisliği uygulamaları ile karşılaştırıldığında şelale stili gelişimi (Kassab ve
diğerleri 2014).

Ancak şu soru ortaya çıkıyor: Çevik geliştirme metodolojileri ne zaman kullanılmalıdır?
Boehm ve Turner, soruyu cevaplamanın yolunun şuna bakmak olduğunu öne sürüyorlar.
beş boyutlu bir süreklilik boyunca projelendirme; büyüklük (personel sayısı açısından)
dahil), sistem kritikliği, personel beceri düzeyi, dinamizm (beklenen sistem belirli bir zaman
aralığında değişir) ve organizasyonel kültür (ister organizasyon kaos veya düzen üzerinde
gelişir) (Boehm ve Turner 2003).

proje özellikleri diyagramın merkezinden uzaklaştıkça, çevik metodolojileri kullanarak başarılı olma olasılığı azalır.

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

ÇEVİK GEREKSİNİMLER EN İYİ UYGULAMALARI

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

- Aktif paydaş katılımına sahip olun
- 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 CRC'ler, kabul testleri, iş kural tanımları, değişiklik durumları, veri akış şemaları, kullanıcı arayüzleri, kullanım senaryoları, proto türleri, özellikler ve senaryolar, kullanım senaryoları diyagramları ve gereksinimleri modellemek için kullanıcı hikayeleri (Ambler 2007). Bu unsurlar yazılım gereksinimlerine eklenebilir kullanıcı hikayeleri ile birlikte şartname belgesi. Gereksinimlerin ortaya çıkarılması için görüşmeleri (hem yüz yüze hem de elektronik), odak grupları, JAD, eski kod analizi, etnografik gözlem, etki alanı analizi ve müşterinin her zaman yerinde olması (Ambler 2007). İçin Bu bölümün geri kalanında, tartışmamız kullanıcı hikayelerinin kullanımına odaklanmaktadır.

XP'DE GEREKSİNİM MÜHENDİSLİĞİ

Xp'deki gereksinim mühendisliği, Şekil 7.5'te gösterilen Ambler'ın modelindeki gereksinim yığınının kullanıcı hikayelerini ifade ettiği modeli izler. Ve XP'de, kullanıcı hikayeleri "planlama oyunu" aracılığıyla kod olarak yönetilir ve uygulanır."

Xp'deki planlama oyunu iki biçim alır: sürüm ve yineleme planlaması.

Sürüm planlaması, ilk kullanıcı hikayeleri yazıldıktan sonra gerçekleşir.

Bu hikayeler kümesi, genel proje planını geliştirmek ve yinelemeleri planlamak için kullanılır.

Küme, her kullanıcı hikayesi ve genel proje için yaklaşık zamanlamaya karar vermek için de kullanılır. Yineleme planlaması, bir dizi kullanıcı hikayesinin ve önceki yinelemelerdeki başarısız testlere yönelik düzeltmelerin uygulandığı bir zaman dilimidir.

Her yineleme 1-3 haftadır süresi içinde. Önceki yinelemelerden (proje hızı olarak adlandırılır) kullanıcı hikayelerinin uygulama hızını izlemek, geliştirme programını iyileştirmeye yardımcı olur.

Bu süreçler sırasında gereksinimler sürekli geliştiği için XP yaratıcısı Kent Beck, “XP’de gereksinimlerin bir belge değil, diyalogtur” diyor (Beck ve ark. 2001) kullanıcı hikayeleri yığınına bir yazılım gereksinimleri belirtimine dönüştürmek tipik olsa da.

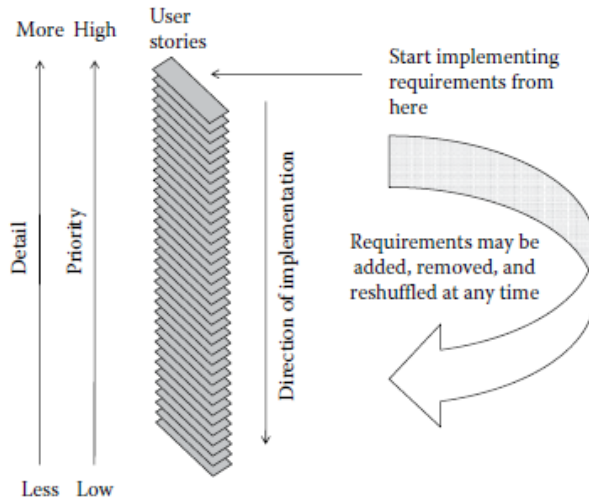


Figure 7.5 Agile requirements change management process. (Adapted from Ambler, S., Agile requirements change management, 2007, <http://www.agilemodeling.com/essays/changeManagement.htm>, last (accessed January 2017).)

SCRUM'DA GEREKSİNİM MÜHENDİSLİĞİ

Scrum'da Şekil 7.5 modelinde gösterilen gereksinimler yığını, XP’de olduğu gibi, kullanıcı hikayelerinin gelişen birikimidir. 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 bekleme (backlog) listesi üç türe ayrılır: ürün, sürüm ve sprint. Ürün bekleme listesi sürüm geri kayıtlarını içerir ve her sürüm sprint bekleme listesini içerir. Şekil 7.6, bekleme listesi öğelerinin muhafaza ilişkisini gösteren bir Venn diyagramıdır.

Ürün bekleme listesi, bir noktada serbest bırakılması hedeflenen gereksinimler için bir depo görevi görür. Ürün bekleme listesindeki gereksinimler düşük, orta ve üst düzey gereksinimleri içerir.

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

Son olarak, sprint bekleme listesi, ekibin sprint'in sonunda tamamlayacağı (tamamen kodlanmış, test edilmiş ve belgelenmiş) bir dizi sürüm gereksinimidir.

Bu gereksinimler çok yüksek bir ayrıntı düzeyine evrilmiştir ve bu nedenle öncelikleri yüksektir.

Scrum, kayda değer bir başarı ile birçok büyük şirkette benimsenmiştir. Yazarın öğrencilerinden bazıları Scrum'u derslerde de kullanmaktadır. Bu durumlarda, uzun gereksinim bulma süreçleri için çok az zaman olduğunda oldukça etkili olduğunu kanıtlar.

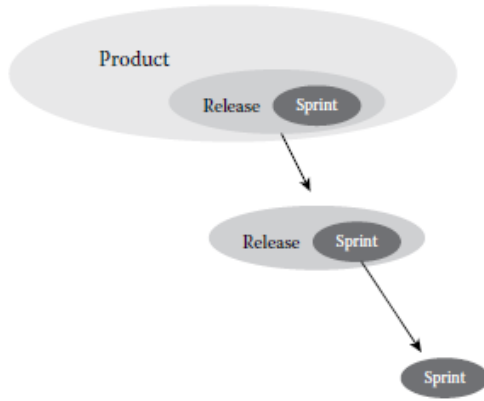


Figure 7.6 Backlog relationship between product, releases, and sprints.

KULLANICI HİKAYELERİ YAZMA

Kullanıcı hikayeleri, çoğu çevik metodolojideki en temel gereksinim birimidir. Her kullanıcı hikayesi, müşteri tarafından istenen bir özelliği temsil eder. Kullanıcı hikayeleri (Kent Beck tarafından yazılan bir terim) müşteri tarafından dizin kartlarına yazılır, ancak süreç vikiler veya diğer araçlar aracılığıyla otomatikleştirilebilir. Resmi gereksinimler, kullanım durumları ve diğer eserler, gerektiğinde yazılım mühendisliği ekibi tarafından kullanıcı hikayelerinden türetilir.

Bir kullanıcı hikayesi aşağıdaki bileşenlerden oluşur:

Başlık-bu hikaye için kısa bir saptır. Başlıkta aktif seste şimdiki zaman fiili arzu edilir.

Kabul testi - bu, hikayeyi test etmek için bir yöntemin adı olacak benzersiz bir tanımlayıcıdır.

Öncelik-bu, kabul edilen önceliklendirme şemasına dayanmaktadır. Öncelik, önemin “geleneksel” önceliklendirilmesine veya ayrıntı düzeyine göre atanabilir (daha yüksek ayrıntıya daha yüksek öncelik verilir).

Hikaye noktaları -bu, kullanıcı hikayesini uygulamak için tahmini süredir. Bu yön, kullanıcı hikayelerini çaba ve maliyet tahmini için yararlı kılar.

Açıklama-bu hikayeyi açıklayan bir ila üç cümledir.

Dizin kartındaki bu öğeler için örnek bir düzen Tablo 7.1'de gösterilmiştir. İlk kullanıcı hikayeleri genellikle küçük site dışı toplantılarda toplanır. Hikayeler, hedefe yönelik olarak oluşturulabilir (örneğin, “bir müşterinin nasıl bir şey yaptığını tartışalım (“satın alma”) yaklaşımlarıyla veya etkileşimli (bilinç akışı) yaklaşımlarıyla. Kullanıcı hikayeleri geliştirmek “yinelemeli ve etkileşimli” bir süreçtir. Geliştirme ekibi aynı zamanda tekdüzelik için hikayelerin boyutunu da yönetir (örneğin, çok büyük—bölünmüş, çok küçük—birleştirme).

Evcil hayvan mağazası POS sisteminde ürün iade eden bir müşteri için örnek bir kullanıcı hikayesi Tablo 7.2'de gösterilmiştir.

Kullanıcı hikayeleri müşteriler için anlaşılabilir olmalı ve her hikaye değer katmalıdır.

Geliştiriciler kullanıcı hikayeleri yazmaz, kullanıcılar yapar. Ancak hikayelerin, yineleme başına birkaçının tamamlanabileceği kadar küçük olması gerekir.

Hikayeler bağımsız olmalıdır (mümkün olduğunca); yani bir hikaye başka hikayelere ileri geri gönderme yapmamalıdır.

Son olarak, hikayeler test edilebilir olmalıdır - herhangi bir gereksinim gibi, test edilemezse, bu bir gereklilik değildir.

Her hikayenin test edilebilirliği geliştirme ekibi tarafından değerlendirilir.

Tablo 7.3, havaalanı bagaj taşıma sistemindeki bir güvenlik tehdidi tespitini açıklayan başka bir örnek kullanıcı hikayesini göstermektedir.

Son olarak, kullanım durumları ve kullanıcı hikayeleri arasında önemli bir fark olduğunu belirtmek gerekir. Kullanıcı hikayeleri müşteri perspektifinden gelir ve basittir ve uygulama ayrıntılarından kaçınır.

Kullanım durumları daha karmaşıktır ve uygulama ayrıntılarını (ör. fabrikasyon nesneler) içerebilir.

Müşteriler genellikle kullanım durumları yazmazlar (ve eğer yaparlarsa, dikkatli olun, çünkü artık müşteri “yazılım mühendisliği” ile uğraşmaktadır).

Son olarak, kullanıcı hikayesi başına kullanım durumlarının sayısı için eşdeğerliğin ne olduğunu söylemek zor. Bir kullanıcı hikayesi bir veya 20'den fazla kullanım durumuna eşit olabilir.

Table 7.1 User Story Layout

Title		
Acceptance Test	Priority	Story Points
Description		

Table 7.2 User Story: Pet Store POS System

Title: Customer returns items		
Acceptance Test: custRetItem	Priority: 1	Story Points: 2
When a customer returns an item, its purchase should be authenticated. If the purchase was authentic then the customer's account should be credited or the purchase amount returned. The inventory should be updated accordingly.		

Table 7.3 User Story: Baggage Handling

Title: Detect Security Threat		
Acceptance Test: detSecThrt	Priority: 1	Story Points: 3
When a scanned bag has been determined to contain an instance of a banned item, the bag shall be diverted to the security checkpoint conveyor. The security manager shall be sent an email stating that a potential threat has been detected.		

Örneğin, Tablo 7.2'deki müşteri iadesi kullanıcı hikayesi için, müşteri iadesinde ortaya çıkabilecek çeşitli durumlarla başa çıkmanın çok daha fazla kullanım durumu gerektireceğini düşünebilirsiniz.

Çevik metodolojilerde, kullanıcı hikayeleri durumları kullanmak için çok tercih edilir.

Ek D, hem çevik hem de çevik olmayan ortamlarda kullanıcı hikayelerinin daha fazla tartışılmasını içerir.

ÇEVİK GEREKSİNİMLER MÜHENDİSLİĞİ

Çevik metodolojiler için gereksinim mühendisliği ile çevik gereksinim mühendisliği arasında bir ayrım yapmamız gerekiyor. Çevik gereksinimler mühendisliği, genel olarak, geleneksel gereksinimler mühendisliğinden daha esnek olduğu iddia edilen herhangi bir özel gereksinim mühendisliği yaklaşımı anlamına gelir. Bu tanım, az önce tartıştığımız gibi çevik

metodolojilerde gereksinim mühendisliği için özel uygulamalarla karıştırılmamalıdır (Sillitti and Succi 2006).

Son birkaç yılda, birçoğu özensiz gereksinim mühendisliğinden çok daha fazla olmayan bir dizi çevik gereksinim mühendisliği yaklaşımı getirilmiştir. Ancak bu alandaki son çalışmalardan bazıları da iyiydi. Bununla birlikte, karşılaşılabileceğiniz "meşru" çevik gereksinimler mühendisliği metodolojileri için, uygulamaların çoğu Çevik Manifesto'ya kadar izlenebilir. Örneğin, Vlaanderen et al. (2011), Scrum'un belirli gereksinim mühendisliği uygulamalarının yazılım ürün yönetimi için de (teslimattan sonra) nasıl olabileceğini göstermiştir. Özellikle sprint döngülerini, backlog yönetimini, günlük toplantıları, erken ve sık işbirliğini gerekli uygulamalar olarak belirlediler.

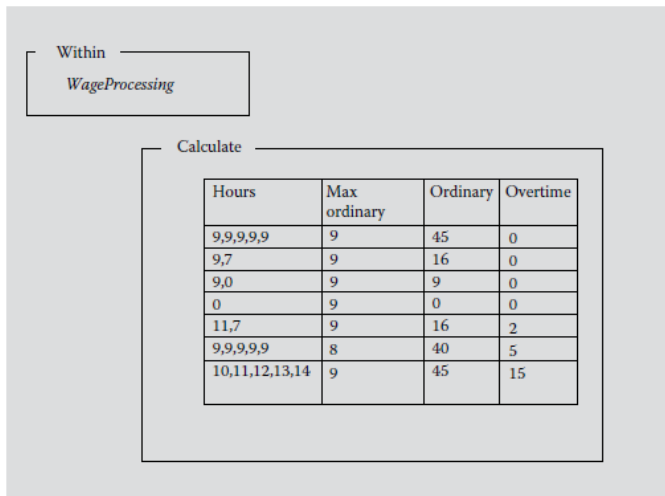
HİKAYE TESTİ ODAKLI GELİŞTİRME

Çevik bir metodolojiyi göstermek için dikkate değer bir örneği açıklıyoruz. Hikaye testine dayalı geliştirme (SDD) adı verilen bu metodolojide çevik metodolojilerin alışılmış müşteri ile (bire bir) iletişim özelliklerinin birçoğu, XP ve Scrum'da kısa süreli yinelemeli geliştirme patlamaları ile birlikte dahil edilmiştir. Bununla birlikte, SDD' deki bir fark, geleneksel kullanıcı hikayeleri yerine müşterilerin "hikaye testleri" yazması veya gözden geçirmesidir. Hikaye testleri, davranışın doğru olduğunu doğrulayan testlerle birlikte davranışın teknik olmayan açıklamalarıdır. Hikaye testleri, "bir hikayede birçok eksik parçayı veya tutarsızlığı keşfetmeye" yardımcı olur (Mugridge 2008).

Hikaye testinin güzel bir özelliği, kullanıcıların hikayelere tablo biçiminde test durumları oluşturmalarına izin vermek için Fit framework kullanmasıdır (bkz. Bölüm 8). Bu nedenle, kullanıcılar sezgisel olarak davranışı ve bu davranışa ilişkin sınamaları gereksinimlerde belirtir.

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

Şekil 7.7' deki tablo, örneğin, bir çalışanın 5 ardışık 9 saatlik iş günü çalışması durumunda, normal olarak 45 saat ve fazla mesai ücretinde sıfır saat olarak kabul edildiğini göstermektedir.



Hours	Max ordinary	Ordinary	Overtime
9,9,9,9,9	9	45	0
9,7	9	16	0
9,0	9	9	0
0	9	0	0
11,7	9	16	2
9,9,9,9,9	8	40	5
10,11,12,13,14	9	45	15

Figure 7.7 Using a story test to show how a company calculates pay for ordinary and overtime hours. (From Mugridge, R., *Software*, 25: 68–75, 2008.)

Bir işçi haftada 10, 11, 12, 13 ve 14 saatlik günleri izlerse, 45 saatlik düzenli ve 15 saatlik fazla mesai ücreti alma hakkına sahiptir. Ancak Fit framework etkileşimli bir belirtimdir - yanlış olan tabloya farklı değerler eklemek geçersiz olarak gösterilir (bkz. Bölüm 8).

SDD geliştirmenin, birincisinin sistemin bütününe geliştirmeye uygulanması bakımından test odaklı geliştirmeyi (TDD) tamamlayıcı olduğu düşünülmektedir (Mugridge 2008).

ÇEVİK METODOLOJİLERDE GEREKSİNİM MÜHENDİSLİĞİ İÇİN ZORLUKLAR

Tabii ki, herhangi yeni bir teknolojiye zorluklar vardır ve özellikle gereksinim mühendisliği ile ilgili olarak çevik metodolojilerin kullanılmasında bazı zorluklar vardır. Williams (2004) bu eksikliklerin bazılarını tartışmaktadır.

Örneğin, çevik metodolojiler işlevsel olmayan gereksinimlerle her zaman iyi baş edemez. Bu neden böyle olmalı? Bunun bir nedeni, yalnızca gereksinim işlevselliği ile ilgilenirken (kullanıcı hikayeleri aracılığıyla) her zaman belirgin olmamalarıdır. Williams, kullanıcı hikayelerini rekabetçi analiz gibi uygun işlevsel olmayan gereksinim bulma teknikleriyle genişleterek bu zorluğun üstesinden gelmeyi önerir.

Diğer bir eksiklik, çevik metodolojilerle müşteri etkileşiminin güçlü olmasıdır - ancak çoğunlukla prototipleme yoluyla. Gördüğümüz gibi, nüanslı gereklilikleri ortaya çıkarmanın ve paydaş ihtiyaçlarını anlamının başka yolları da vardır, örneğin, çeşitli görüşme tekniklerini kullanmak arzu edilir.

Ayrıca, çevik metodolojilerle geçerleme, test ve prototipleme yoluyla güçlüdür, ancak doğrulama o kadar güçlü değildir. Williams (2004), biçimsel yöntemlerin kullanılmasının çevik gereksinim mühendisliğini güçlendirebileceğini öne sürmektedir (2004).

Rubin ve Rubin (2011) çevik yazılım geliştirmede sıklıkla gözden kaçan dokümantasyon yapılarını belirlediler ve bu sorunu azaltmanın yollarını önerdiler. Özellikle, etki alanı bilgisinin ve ortaya çıkan sistem mimarisi ve tasarımının geleneksel çevik gereksinim uygulamaları tarafından uygun şekilde yakalanmadığını belirtmişlerdir. Bulgular, çevik gereksinimler belgelerinin uygun alan bilgisi ile açıklanması gerektiğini ve karşılık gelen çevik mimari ve tasarım belgelerinin sırasıyla son sistem mimarisi ve tasarımı ile açıklanması gerektiğini ima eder.

Gereksinim yönetimi sürece dahil edilmiştir (örneğin, XP ve Scrum), ancak çoğunlukla kod düzeyine odaklanmıştır. William, daha standart yapılandırma yönetimi uygulamaları eklenerek gereksinim yönetiminin güçlendirilebileceğini öne sürüyor.

Son olarak, Inayet et al. (2015) bir dizi çevik gereksinim mühendisliği zorluğunu özetlemiştir. Inayet et al. tarafından belirlenen zorluklar:

- İşlevsel olmayan gereksinimleri ihmal etme
- Gereksinimlerin izlenebilirliğinin eksikliği
- Yanlış gereksinimlerin önceliklendirilmesi
- Minimum gereksinimler dokümantasyonu
- Sözleşme sorunları
- Müşteri uygunluğu
- Müşteri sözleşmesi

Bu zorluklar başkaları tarafından tanınlananlardan bazılarını tekrarladı.

Yazılım mühendisliği için çevik metodolojilerin kullanılmasında hala önemli zorluklar olsa da, önemli avantajlar da vardır. Bu nedenle, çevik yazılım metodolojileri en azından çoğu proje için dikkate alınmalıdır. Çevik metodolojiler belirli yazılım projeleri için uygun

görülmendiğinde veya proje yazılım olmadığında bile çevik metodolojiler için tartışılan uygulamaların çoğu kullanılabilir. Örneğin, müşterinin her zaman yerinde olması, dokümantasyon üzerinden ürüne odaklanma ve test durumlarının erken geliştirilmesi gibi uygulamalar hemen hemen her projeye dahil edilmelidir.

BÖLÜM 7.1 EKONOMİK BAKIM YASASI WEB SİTESİ

2010 yılında Amerika Birleşik Devletleri, tüm vatandaşların bir tür sağlık sigortası almasını gerektiren Ekonomik Bakım Yasasını kabul etti. Yasa tartışmalı bir yasaydı ve tartışmaya eklenmesi federal hükümetin ilgili web sitesinin feci bir şekilde ortaya çıkmasıydı. Proje karmaşıktı ve 55 müteahhit içeriyordu. Sağlam bir geliştirme sürecine odaklanmak yerine, web sitesini dağıtmak için acele edildi ve şaşırtıcı olmayan bir şekilde birçok sorundan muzdaripti. Birçok yapısal ve güvenlik sorunu vardı, ancak sık sık çökme, kötü yanıt süreleri, eksik veya yanıltıcı içerik ve gezinme zorlukları gibi kullanılabilirlik sorunları en çok kamuoyunun dikkatini çekti (Venkatesh et al. 2014). Dağıtımın ilk haftasında yaklaşık 9.47 milyon kullanıcı kaydolmaya çalıştı ancak yalnızca 271.000 kişi başarılı oldu (Cleland-Huang 2014). Web sitesinin başarısız lansmanına ilişkin çalışmalar, proje takviminin küçümsenmesi, kötü tanımlanmış kapsam, kötü gereksinim belirtimi, verimsiz risk analizi ve yönetimi gibi çeşitli temel nedenleri göstermiştir. (Anthopoulos et al. 2016).

Birçok uzman, daha iyi bir geleneksel gereksinim sürecinin bu sorunları azaltacağını doğru bir şekilde öne sürse de, siteyi hızlı bir şekilde dağıtma baskısı muazzamdı. Bununla birlikte, çevik, dinamik bir gereksinim keşfi ve geliştirme sürecinin daha iyi proje sonuçlarına ve kamuoyunun algılanmasına yol açması ve daha erken bir dağıtım izin vermesi muhtemeldir. Böyle bir sürecin önerildiği veya değerlendirildiği bildirilmemiştir.

ALIŞTIRMALAR

- 7.1 SRS belgelerini çevik bir framework' a nasıl sığdırabiliriz?
- 7.2 Müşteri yerinde değilken çevik metodolojileri kullanmak mümkün müdür? Eğer öyleyse, nasıl?
- 7.3 Çevik metodolojiler neden yazılım projelerinin aksine donanım tabanlı projeler için genellikle uygun değildir?
- 7.4 Çevik metodolojilerin işlevsel olmayan gereksinimleri karşılaması neden zor olabilir?
- 7.5 Gereksinimleri kullanıcı hikayelerine kapsüllemeye herhangi bir sorun var mı?
- 7.6 Evcil hayvan mağazası POS sistemi için, müşteri satın alımları için bir kullanıcı hikayesi oluşturun.
- 7.7 Evcil hayvan mağazası için POS sistemi için, çeşitli müşteri alımları için kullanım durumları oluşturun.
- 7.8 Havaalanı bagaj taşıma sistemi için, başka bir uçuşa yönlendirilecek bagajla başa çıkmak için bir kullanıcı hikayesi oluşturun.
- 7.9 Havaalanı bagaj taşıma sistemi için, kayıp bir bagaj parçasıyla başa çıkmak için kullanım senaryoları oluşturun.
- 7.10 Havaalanı bagaj taşıma sistemi için, başka bir uçuşa yönlendirilecek bagajla başa çıkmak için kullanım durumları oluşturun.
- 7.11 Aşağıdaki sistemler için çevik bir yaklaşım kullanmanın avantaj ve dezavantajlarını tartışın (yazılım bileşenleri için):
 - 7.11.1 Evcil hayvan mağazası POS sistemi
 - 7.11.2 Havayolu bagaj taşıma sistemi
 - 7.11.3 Akıllı ev sistemi (Ek A)
- 7.12 Çevik bir projenin gereksinim faaliyetlerini yönetmek için kullanılabilecek hem ticari hem de açık kaynaklı bir dizi yazılım aracı vardır. Bu araçları araştırın ve bir ürün karşılaştırma matrisi hazırlayın.