YAZILIM MÜHENDİSLİĞİ – SOFTWARE ENGINEERING

YAZ16103
Yazılım Mühendisliğine Giriş

Dr. Öğr. Üyesi Bora ASLAN

İÇİNDEKİLER

3.BÖLÜM

ÇEVİK YAZILIM GELİŞTİRME

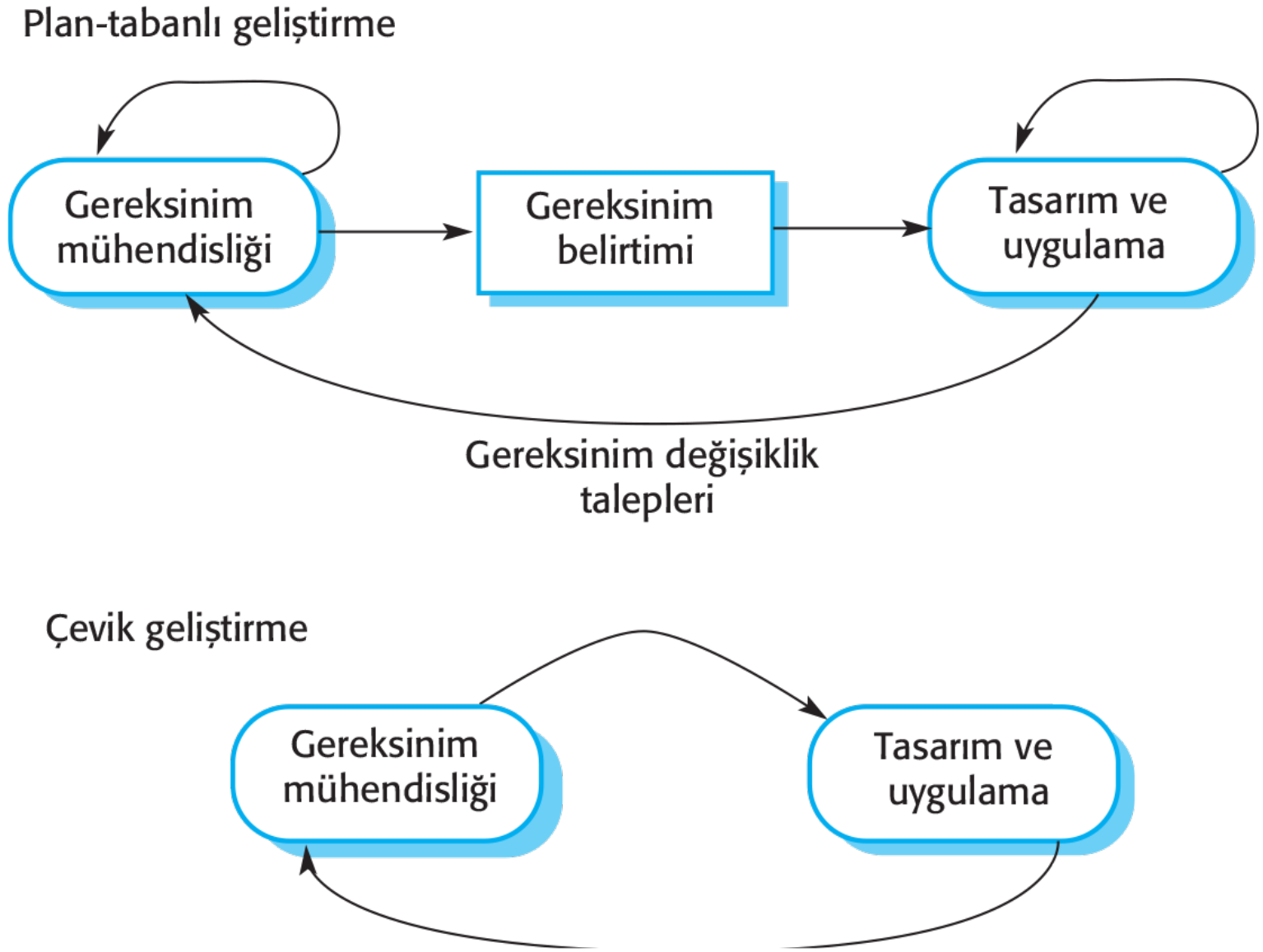
Hızlı yazılım geliştirme (Rapid software development), çevik yazılım geliştirme (agile development) ya da çevik yöntemler (agile methods) olarak bilinir hale gelmiştir. Çevik yöntemler, kullanılabilir yazılımın çabuk bir şekilde üretilebilmesi düşüncesi ile tasarlanmıştır. Önerilen tüm çevik yöntemler bir dizi ortak özelliği paylaşır:

1. Spesifikasyon, tasarım ve uygulama süreçleri iç içe geçmiştir. Detaylı sistem spesifikasyonu ve tasarım dokümantasyonu yoktur ya da sistemi geliştirmek için kullanılan uygulama ortamı tarafından otomatik olarak oluşturulur.

2. Sistem bir dizi artım ile geliştirilir. Son-kullanıcılar ve diğer sistem paydaşları her artımın belirlenmesi ve değerlendirilmesi süreçlerine katılırlar.

3. Geliştirme sürecini desteklemek üzere kapsamlı araç desteği alınır. Bu araçlar otomatik test araçlarını, yapılandırma yönetimini ve sistem bütünleştirmeyi destekleyen ve kullanıcı arayüzü geliştirmeyi otomatikleştiren araçları içerebilir.

- Çevik yöntemler küçük artımlarla ilerleyen, her artımda yeni özelliklerin yayımlandığı, iki ya da üç haftalık aralıklarla sistemin müşteriye sunulduğu artımlı geliştirme yöntemleridir.
- Bu yöntemler değişen gereksinimler hakkında hızlı geri bildirim almak için müşterileri geliştirme sürecine dahil ederler.
- Yazılım geliştirmede çevik yaklaşımlar tasarım ve uygulamanın yazılım sürecindeki temel etkinlikler olduğunu kabul eder.
- Gereksinim çıkarma ve test gibi diğer etkinlikler tasarım ve uygulama etkinlikleri ile birleştirilir.
- Bunun tersi olarak, plan-güdümlü bir yazılım mühendisliği yaklaşımı yazılım sürecindeki ayrı aşamaları her aşama ile bir çıktıyı ilişkilendirerek tanımlar.
- Bir aşama sonunda elde edilen çıktılar takip eden süreç etkinliklerini planlamak için temel olarak kullanılır.



Şekil 3.1 Plan-güdümlü ve çevik geliştirme

ÇEVİK YÖNTEMLER

1980'lerde ve 1990'ların başında daha iyi yazılım geliştirmenin en iyi yolunun dikkatli proje planlama, resmi kalite güvence, yazılım araçları ile desteklenen analiz ve tasarım yöntemlerinin kullanımı, kontrollü ve katı yazılım geliştirme süreçleri ile mümkün olduğuna dair yaygın bir görüş vardı. Bu görüş havacılık ve devlet sistemleri gibi geniş ve uzun-yaşam süreli projeler geliştiren yazılım mühendisliği topluluğu kaynaklıydı.

Bu plan-güdümlü yaklaşım farklı şirketlerde çalışan büyük takımlar tarafından kullanıldı. Takımlar genellikle coğrafi olarak ayrıktı ve yazılım üzerinde uzun süreler çalışmaları gerekiyordu. Bu tür bir yazılıma, örnek olarak, proje başladıktan 10 yıl sonra, teslimi gerçekleştirilebilen modern bir uzay aracının kontrol sistemi verilebilir.

- Fakat bu hantal, plan-güdümlü geliştirme yaklaşımı küçük ve orta ölçekte sistemlere uyguladığında, ortaya çıkan ek yük ve maliyet öyle büyüktür ki, yazılım geliştirme sürecinde baskındır.
- Sistemin nasıl geliştirilmesi gerektiğine program geliştirme ve testten daha çok zaman ayrılır.
- Sistem gereksinimleri değiştikçe, tekrarlar kaçınılmaz hale gelir. en azından prensipte, gereksinim pasifikasyon ve tasarımının programla birlikte değişmesi gerekir.
- Yazılım mühendisliğindeki bu hantal yaklaşımların kullanımıyla ortaya çıkan memnuniyetsizlik nedeniyle 1990'ların sonlarında çevik yöntemler geliştirilmeye başlanmıştır.
- Bu yöntemler geliştirme takımlarının tasarım ve belgelemeden ziyade yazılımın kendisine odaklanmasını mümkün kılmıştır.
- Bu yöntemler, geliştirme süreci boyunca sistem gereksinimlerinin hızla değişmesini öngören uygulamaların geliştirilmesinde en uygun olanıdır.
- Çevik yöntemlerin amacı çalışan yazılımın, sistemin sonraki artımlarında geliştirilmek üzere yeni özellikler ya da değişiklik isteği talebinde bulunabilecek müşteriye hızlı bir şekilde teslim edilmesidir.

İlke/prensip	Tanım
Müşterinin katılımı	Müşteriler geliştirme süreci boyunca geliştirme takımı ile yakın biçimde çalışmalıdır. Rolleri yeni gereksinimleri sağlamak, önceliklendirmek ve sistem artımlarını değerlendirmektir.
Değişimin benimsenmesi	Sistem gereksinimlerinin değişimi kaçınılmazdır, sistem bu değişiklikleri kabul edecek şekilde tasarlanmalıdır.
Artımlı teslim	Yazılım müşterinin her artım için belirlediği gereksinimlerin dahil edildiği artımlarla geliştirilir.
Sadeliğin sürdürülmesi	Hem geliştirilen yazılımda hem de yazılım geliştirme sürecinde sadeliğe odaklanın. Mümkün olduğu sürece sistemin karmaşıklığını azaltmaya çalışın.
Süreçler değil, insanlar	Yazılım geliştirme takımının yetenekleri bilinmeli ve bu yeteneklerden faydalanılmalıdır. Takım üyeleri detaylı tarif veren süreçleri takip etmek yerine kendi çalışma yaklaşımlarını geliştirme konusunda özgür bırakılmalıdır.

Şekil 3.2 Çevik yöntemlerin ilkeleri

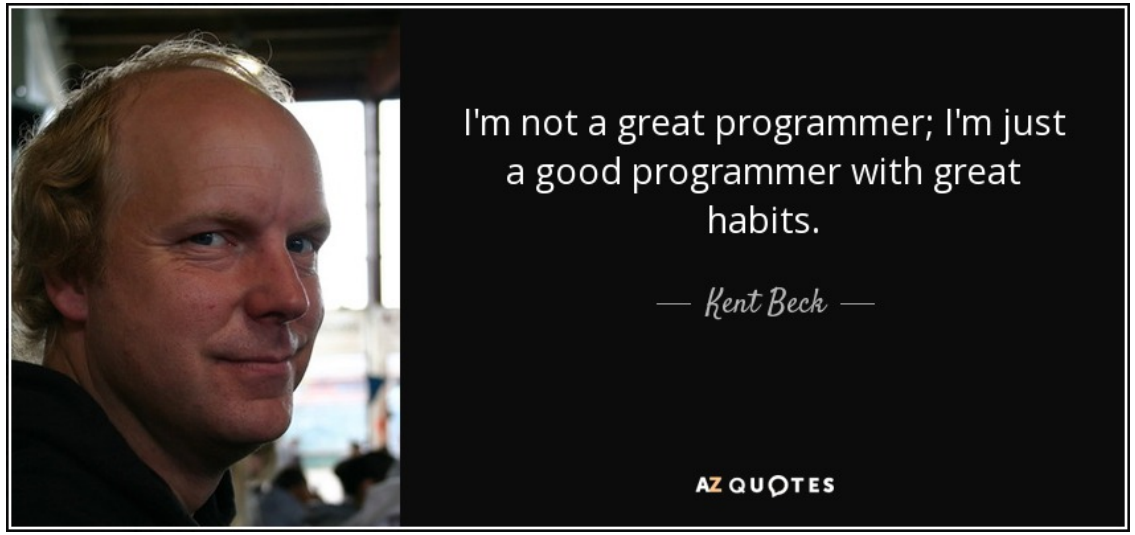
- Çevik yöntemlerin ardında yer alan felsefe, çevik yöntemleri geliştiren kişiler tarafından yayımlanan çevik manifestoda (http://agilemanifesto.org) tanıtılmıştır.

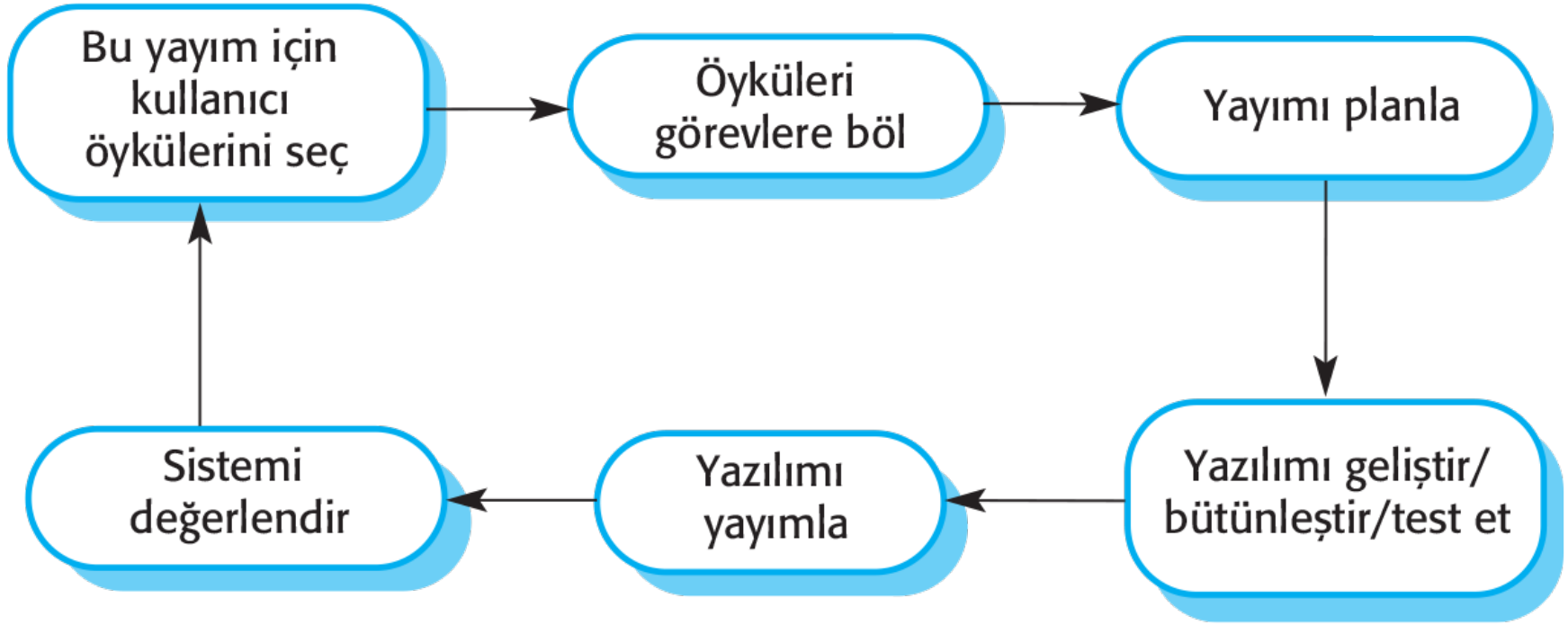
Çevik yöntemler özellikle iki tür sistemin geliştirilmesinde başarılı sonuçlar vermektedir:

1. Bir yazılım şirketinin satış için küçük ya da orta ölçekli ürünler geliştirmesinde Günümüzde neredeyse tüm yazılım ürünleri ve uygulamalar çevik yaklaşımlar kullanılarak geliştirilmektedir.
2. Organizasyon içerisinde geliştirilen, müşterinin kesin bir biçimde süreçlere katılmayı taahhüt ettiği, az sayıda dış paydaşın yer aldığı ve yasa, yönetmelik ve mevzuatlardan çok az etkilenen sistem geliştirilmesinde.

ÇEVİK GELİŞTİRME TEKNİKLERİ

- Çevik yöntemlerin altında yatan fikirler 1990'larda farklı kişiler tarafından aynı zamanlarda geliştirilmeye başlandı.
- Fakat yazılım geliştirme kültürünü kökten sarsan belki de en önemli yaklaşım Uç Programlama'nın geliştirilmesi (Extreme Programming) oldu.
- Yöntemin ismi Kent Beck (Beck 1998) tarafından, UP yöntemi var olan iyi pratikleri ör.; yinelemeyi geliştirme gibi- uç limitlere kadar zorladığı için verilmiştir.
- Örneğin UP'de bir sistemin birden fazla sürümü farklı yazılım geliştiriciler tarafından bir gün içerisinde geliştirilip test edilebilir.





Şekil 3.3 UP yayım döngüsü

Uç programlama ortaya çıktığı yıllarda tanıttığı bir dizi çevik pratikle tartışmalara neden olmuştu. Uç programlamadaki çevik manifestodaki yansımaları aşağıda verilmiştir:

1. Artımlı yazılım geliştirme, küçük ve sık yayımlarla (release) desteklenir. Gereksinimler, bir artımda olması gereken fonksiyonlara karar vermede temel oluşturacak olan ve basitçe yazılmış kullanıcı öyküleri ya da senaryoları şeklinde ifade edilir.
2. Müşterinin katılımı geliştirme takımına sürekli olarak dahil edilmesi ile sağlanır. Müşteriyi temsil eden kişiler geliştirmede yer alır ve sistem için kabul testlerini tanımlamaktan sorumludur.
3. Süreçler değil kişiler, eş programlama, sistem kodunun ortak sahipliği ve fazla mesai içermeyen sürdürülebilir çalışma saatleri ile desteklenir.
4. Değişim, müşteriye sunulan düzenli sistem yayımları, test önce geliştirme, kod bozulmasını engellemek için kodun yeniden düzenlenmesi ve yeni fonksiyonların sürekli bütünleştirilmesi ile benimsenir.
5. Sadeliğin sürdürülebilmesi, kod kalitesini iyileştiren yeniden düzenleme ve zorunlu olmadıkça gelecek değişiklikleri göz önünde bulundurmayan sade tasarım ile desteklenir.

İlke ya da pratik	Tanım
Ortak sahiplik	Kod geliştirme sırasında herhangi bir alanda özel bir uzmanlık gelişmemesi ve tüm kodun sorumluluğunun herkes tarafından alınabilmesi amacıyla yazılım geliştiriciler sistemin tüm alanları üzerinde çalışır. Herkes her şeyi değiştirebilir.
Sürekli bütünleştirme	Herhangi bir görev tamamlanır tamamlanmaz sistemle bütünleştirilir. Böyle bir bütünleştirmeden sonra sistemdeki tüm birim sınamalar başarılı olmalıdır.
Artımlı planlama	Gereksinimler "öykü kartlarına" kaydedilir, ve yayıma dahil edilecek öyküler, eldeki zamana ve öykülerin görece önceliğine göre belirlenir. Geliştiriciler bu hikayeleri "görevlere" bölerler. bk. Şekil 3.5 ve Şekil 3.6.
Her zaman hazır müşteri	Sistemin son kullanıcı temsilcisi (müşteri),UP takımına destek vermek amacıyla tam zamanlı olarak hazır bulunmalıdır. Bir U programlama sürecinde müşteri geliştirme takımının bir parçasıdır ve takıma geliştirme amacıyla sistem gereksinimleri iletmekten sorumludur.
Eş programlama	Yazılım geliştirenler birbirinin işini kontrol eden ve işi daha iyi yapabilmek için birbirine destek olan eşler halinde çalışır.
Kodun yeniden üretilmesi	Tüm yazılım geliştiriciler kodu sürekli olarak yeniden üreterek iyileştirmelidir. Bu, kodu sade tutar ve sürdürülebilirliği sağlar.
Sade tasarım	Mevcut gereksinimleri karşılayacak yeterli düzeyde tasarım geliştirilir daha fazlası değil.
Küçük yayımlar	Bir iş değeri sağlayan minimum fonksiyonellik öncelikli olarak geliştirilir. Sistem sık aralıklarla yayımlanır ve her artımda ilk yayımın üzerine yeni fonksiyonlar eklenir.
Sürdürülebilir hız	Fazla miktarda fazla mesai, kod kalitesinde düşüşlere neden olacağı ve orta vadede üretkenliği azaltacağı için kabul edilebilir değildir.
Test önce geliştirme	Fonksiyonun kendisi geliştirilmeden testini geliştirmek için otomatik bir birim test çerçevesi geliştirilir.

Şekil 3.4 Uç programlama pratikleri

Kullanıcı Öyküleri - User Stories

- Yazılım gereksinimleri daima değişir. Bu değişikliklerle baş edebilmek için çevik yöntemler farklı gereksinim mühendisliği etkinliklerine sahip değildir.
- Daha ziyade, çevik yöntemler gereksinimlerin ortaya çıkarılma sürecini geliştirme süreci ile bütünleştirirler.
- Bunu daha kolay yapabilmek için bir sistem kullanıcısı tarafından deneyimlenebilecek senaryoları ifade eden “kullanıcı öyküleri” fikri geliştirilmiştir.
- Mümkün olduğu kadar, sistem müşterisi geliştirme takımı ile birlikte çalışır ve senaryoları takım üyeleri ile birlikte tartışır.
- Birlikte, müşteri ihtiyaçlarını içeren bir öyküyü kısaca tarif eden bir "öykü kartı" geliştirilir. Geliştirme takımı daha sonra bu senaryoyu gelecek yayımlarda geliştirmeyi hedefler.

Reçete yazma

Kate kliniğe gelen bir hastasına reçete yazmak isteyen bir doktordur. Hasta kaydı ekranda önceden görüntülenmiştir. Bu nedenle sadece ilaç alanına tıklar ve 'mevcut ilaç', 'yeni ilaç' ya da 'formül' seçeneklerinden birini seçer.

Eğer 'mevcut ilaç' seçeneğini seçerse, sistem dozu kontrol etmesini ister. Eğer Kate dozu değiştirmek isterse yeni ilaç dozunu girer ve reçeteyi onaylar.

Eğer 'yeni ilaç' seçeneğini seçerse sistem Kate'in hangi ilacı yazacağını bildiğini varsayar. İlaç adının ilk birkaç harfini yazar. Sistem bu harflerle başlayan olası tüm ilaçları listeler. Kate gerekli ilacı seçer ve sistem seçilen ilacın doğru olup olmadığını sorar. Kate son olarak ilaç dozunu girer ve reçeteyi onaylar.

Eğer 'formül' seçeneğini seçerse, sistem onaylanan formülleri görüntülemek için bir arama kutusu gösterir. Kate buradan gerekli ilacı arar, seçer. Sistem seçilen ilacın doğru ilaç olup olmadığını sorar. İlaç dozunu girer ve reçeteyi onaylar.

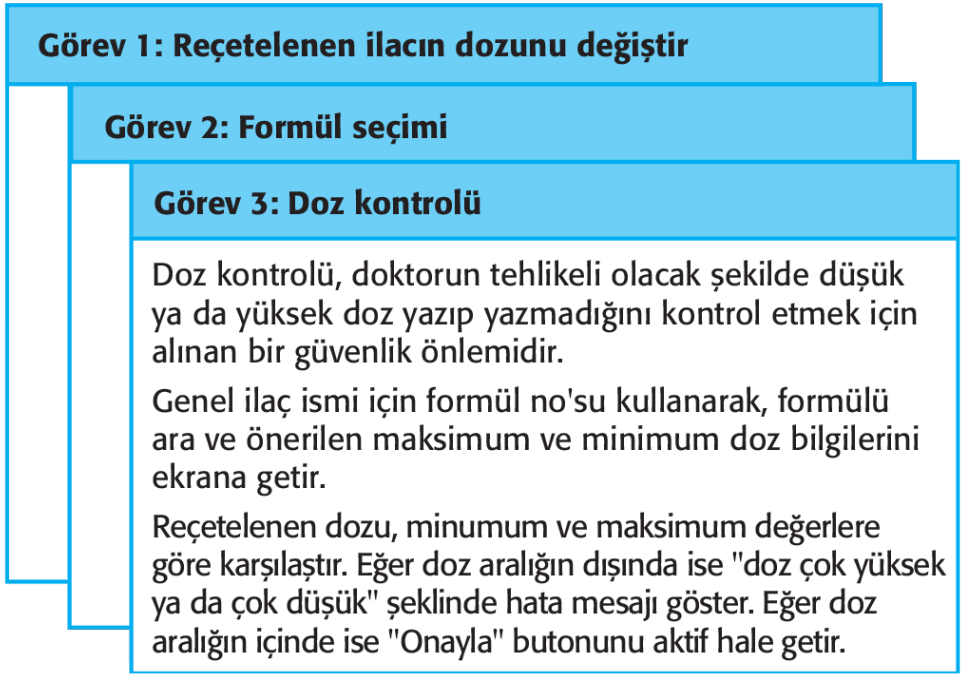
Sistem her zaman, girilen dozun onaylanan değerler arasında olup olmadığını kontrol eder. Eğer değilse Kate'den değiştirmesini ister.

Kate reçeteyi onayladıktan sonra, tekrar kontrol için gösterilir. Gösterim sonrasında 'Onay' ya da 'Değiştir' butonlarından birini tıklar. Eğer 'Onay'a tıklarsa, reçete denetleme veri tabanına kaydedilir. Eğer 'Değiştir'i tıklarsa Reçete Yazma süreci tekrar eder.

Şekil 3.5 Bir "reçete yazma" öyküsü

Kullanıcı Öyküleri - User Stories

- Kullanıcı öyküleri sistem artırımlarını planlamada kullanılabilir. Öykü kartları geliştirildikten sonra, geliştirme takımı bunları görevlere ayırır. ve her bir görevi geliştirmek için gerekli iş gücünü ve kaynakları kestirir.
- Bu genellikle gereksinimlerin iyileştirilmesi için müşteri ile görüşmeleri kapsar. Daha sonra müşteri, geliştirme için hızlı bir şekilde iş desteği alınabilecek öykülerden seçerek, kullanıcı öyküleri önceliklendirir.



Şekil 3.6 Reçete yazma için örnek görev kartları

Yeniden Üretim - Refactoring

- Geleneksel yazılım mühendisliğinin en temel kurallarından biri, değişiklik için tasarım yapma gerekliliğidir. Yani, sistemi tasarlarken değişikliklerin kolaylıkla uygulanabilmesi için, ileride oluşabilecek değişikliler göz önünde bulundurulmalıdır.
- Fakat Uç Programlama, değişim göz önünde bulundurularak yapılan tasarıma dair çabaların çoğunlukla boşa harcandığını ileri sürerek bu prensibi göz ardı etmiştir. Değişiklikle baş edebilmek amacıyla, programın daha genel olması için harcanan zamana değmemektedir. Sıklıkla, öngörülen değişiklikler gerçekleşmemekte ya da öngörülenden tamamen farklı değişiklik talepleri oluşmaktadır.
- Yeniden üretim programlama takımının yazılımda olası iyileşme noktalarını sürekli olarak araştırması ve fark ettiğinde de öncelikli olarak gerçekleştirmesidir.
- Takım üyeleri, iyileştirilebilecek bir kod bulduklarında, bu değişiklikleri, iyileştirmeye acil bir ihtiyaç olmasa bile yaparlar.

Yeniden Üretim - Refactoring

- Artımlı geliştirmenin temel bir problemi, yerel değişiklilerin yazılım yapısını bozma eğiliminde olmasıdır.
- Böyle durumlarda, yazılımda sonraki aşamalarda gerekecek değişikleri uygulamak her geçen gün daha da zor olacaktır.
- Aslında geliştirme süreci, yazılım geliştirme sırasında karşılaşılan problemlere hızlı çözümler bulunarak ilerler. Ancak bu çözümler geride tekrarlayan (duplicate) ve uygunsuz şekilde tekrar kullanılmış (reuse) kod parçaları bırakır.
- Değişen kodun bu şekilde sisteme eklenmesi ile de, sistemin bütüncül yapısı bozulur.

- Yeniden üretim, yazılım yapısını ve okunabilirliği iyileştirir ve böylelikle yazılım değişirken doğal olarak ortaya çıkan yapısal bozulmanın da önüne geçer.

Test-Önce Geliştirme - Test-First Development

- Artımlı geliştirme ve plan-güdümlü geliştirme arasındaki en önemli farklardan biri sistemin test edilmesindedir. Artımlı geliştirmede, sistem testlerini geliştirmek amacıyla ayrı bir test takımı tarafından kullanılabilecek sistem spesifikasyonu yoktur.
- Bunun sonucu olarak, plan-güdümlü geliştirmeyle karşılaştırıldığında artımlı geliştirmenin bazı yaklaşımlarının resmi olmayan test süreçleri vardır.
- Uç Programlama spesifikasyon olmadan test yapabilmenin zorluklarını işaret edebilmek için program testine yeni bir yaklaşım getirmiştir. Test otomatik hale getiiilir ve geliştirme sürecinin merkezinde yer alır.
- Tüm testler başarılı bir şekilde çalıştırılmadan geliştirme süreci devam etmez. UP testinin önemli noktaları şunlardır:
    1. Test Önce Geliştirme
    2. Senaryolardan Yola Çıkarak Artımlı Test Geliştirme
    3. Test Geliştirme Ve Geçerleme Aşamalarına Kullanıcıların Dahil Edilmesi
    4. Otomatik Test Çerçevelerinin Kullanılması

- UP'nin test öncelikli felsefesi, daha genel test önce yazılım geliştirme tekniklerine doğru evrimleşmiştir.
- Önce kodu ve daha sonra o kodun testini yazmak yerine; kodu yazmadan testleri yazarsınız.
- Bu, testleri kod geliştirilirken koşturabileceğiniz ve problemleri geliştirme sırasında
- tespit edebileceğiniz anlamına gelir.
- Test-önce geliştirmede, test oluşturma görevlerini yerine getiren kişiler sistem için gerekli testleri yazabilmek için spesifikasyonları tam olarak anlamalıdır. Bu da uygulama başlamadan önce gereksinimlerdeki belirsizliklerin, unutulan noktaların çözüme kavuşturulması anlamına gelir.
- Ayrıca, bu durum "test gecikmesi" probleminin de önüne geçer. Test gecikmesi, yazılım geliştiren kişiler test yapan kişilerden daha hız çalıştığı durumda meydana gelir.

Test 4: Doz kontrolü

Girdi:
1. İlacın tek dozuna ait mg ile ifade edilen bir sayı
2. İlacın tek dozlar şeklinde günlük alınma sayısı

Testler:
1. İlacın tek doz miktarının doğru fakat günlük alım sıklığının fazla olduğu durumu test et.
2. İlacın tek dozunun yüksek ya da düşük olduğu durumları test et.
3. İlaç alım sıklığının yüksek ya da düşük olduğu durumları test et.
4. İlaç alım sıklığının izin verilen değerler içinde olduğu durumu test et.

Çıktı:
Onay mesajı ya da ilaç dozunun belirtilen değerler dışında olduğunu gösteren hata mesajı.

Şekil 3.7 Doz kontrolü için test durumu tanımı

ÇEVİK PROJE YÖNETİMİ - Agile Project Management

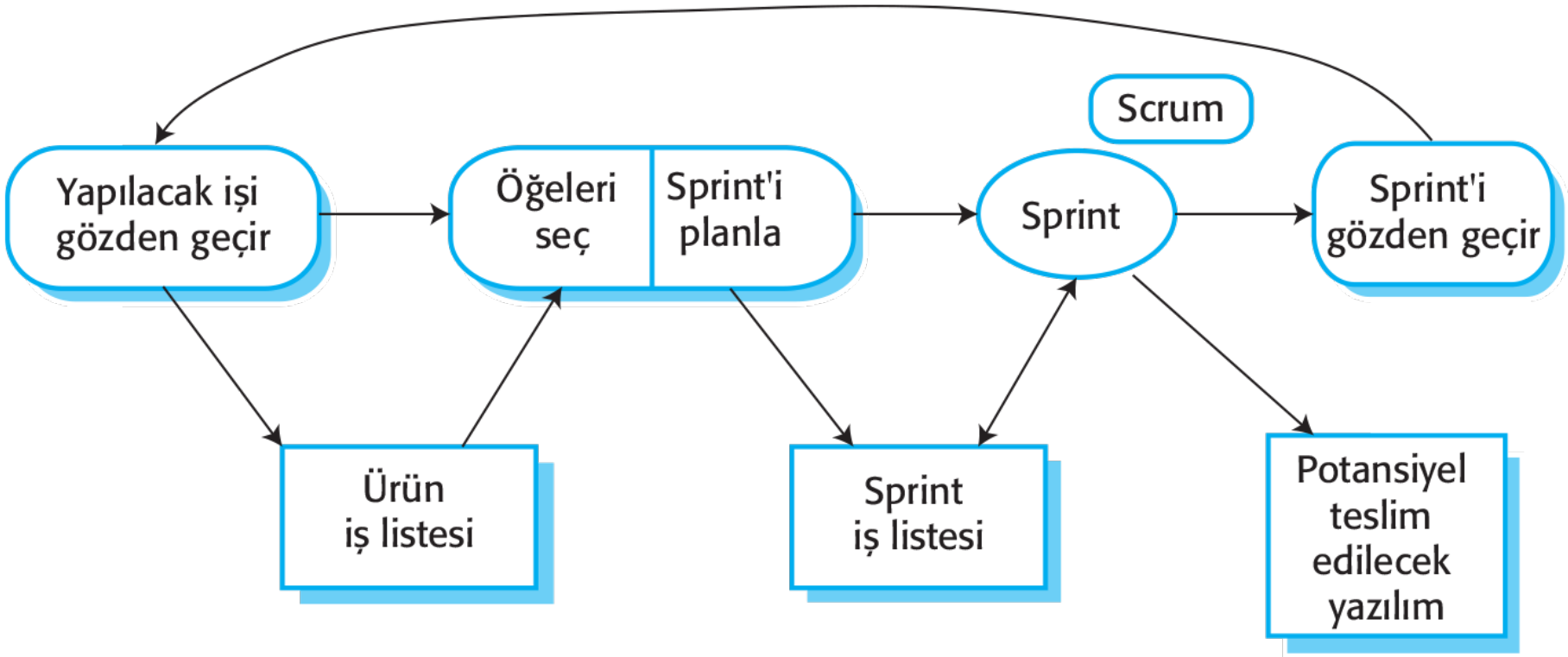
Her yazılım işinde, yöneticiler neler olup bittiğini, projenin hedeflerini karşılayıp karşılayamayacağını ve yazılımın tahmin edilen bütçe ile zamanında teslim edilip edilmeyeceğini bilmek ister.

Yazılım geliştirmede plan-güdümlü yaklaşımlar bu ihtiyacı gidermek için geliştirilmiştir. Plan-güdümlü bir yaklaşım, yöneticinin geliştirilecek her şeyi ve geliştirme süreçlerini devamlı olarak görebilmesini gerektirir.

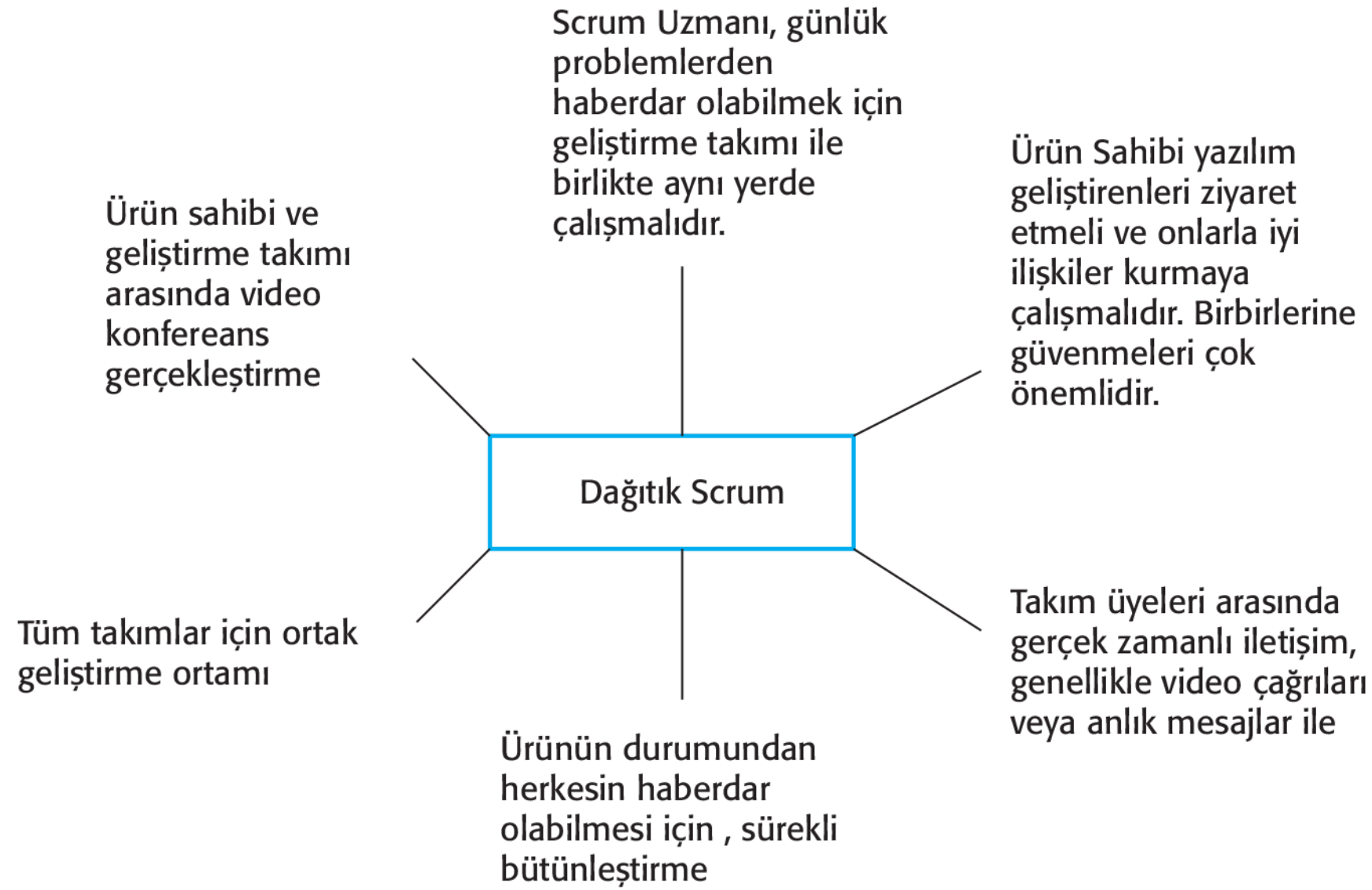
Çevik yöntemleri önce benimseyen kişilerin önerdiği resmi olmayan planlama ve kontrol yöntemleri, iş gereksinimlerinden biri olan görünürlük ile çelişmiştir. Takımlar kendi kendilerine organize olabilir, proje belgesi üretmez ve geliştirmeyi kısa döngüler şeklinde planlar.

Scrum terimi	Tanım	Scrum term
Geliştirme takımı	Yedi kişiden fazla olamaması gereken, yazılım geliştiricilerden oluşan ve kendi kendine organize olan bir gruptur. Yazılım geliştirmekten ve diğer gerekli proje belgelerinden sorumludurlar.	Development team
Potansiyel olarak teslim edilebilir ürün artımı	Bir sprint sonunda teslim edilen yazılım artımıdır. Bir artımın "potansiyel olarak teslime edilebilir" olarak nitelendirilmesinin nedeni artımın, son ürünle bütünleştirmek için test yapılması gibi ek işlere neden olmayacak şekilde son aşamasına gelmiş olmasının beklenmesidir. Uygulamada bu daima başarılabilir değildir.	Potentially shippable product increment
Ürün iş listesi	Scrum takımının uğraşmak zorumda olduğu "yapılacak" iş kalemleri listesidir. Bu kalemler geliştirilecek özellik tanımlarını, yazılım gereksinimlerini, kullanıcı öykülerini ya da mimari tanım ya da kullanıcı dokümantasyonu gibi ihtiyaç duyulan destekleyici görevlerin tanımlarını içerebilir.	Product backlog
Ürün sahibi	İşleri, kullanıcı özelliklerini ya da gereksinimleri belirlemek, geliştirme için bunları önceliklendirmek ve projenin kritik iş ihtiyaçlarını karşılayıp karşılamadığından emin olmak üzere ürün iş listesini düzenli olarak incelemek olan kişiler ya da küçük gruplardır.	Product owner
Scrum	Scrum takımının günlük toplantısı, o gün içerisinde yapılacak işi önceliklendirmek ve işin gidişatını değerlendirmek amacıyla gerçekleştirilir. İdeal olarak bu toplantılara tüm takım katılmalı ve toplantılar yüz yüze yapılmalıdır.	Scrum
Scrum Uzmanı	Scrum Uzmanı, Scrum sürecinin takip edildiğinden emin olmaktan sorumludur ve takımı Scrum'ın etkin kullanımı için yönlendirir. Şirketin kalanı ile iletişim halindedir ve Scrum takımının dış engellemeler dolayısıyla odağından ayılmadığına emin olur. Scrum geliştiriciler, Scrum Uzmanının bir proje yöneticisi gibi düşünülmemesi konusunda kararlıdırlar. Fakat diğerleri farkı daima kolaylıkla göremeyebilir.	ScrumMaster
Sprint	Bir yazılım geliştirme artımı. Sprintler genellikle 2 ila 4 hafta arasındadır.	Sprint
Hız	Bir sprint içerisinde takımın, ürün iş listesi içerisinden ne kadarlık iş yapabileceğinin tahmini. Takımın hızını anlamak bir sprint içerisinde nelerin ele alınacağının kestirilmesine yardımcı olur ve sürekli gelişen performansın ölçülmesi için bir temel oluşturur.	Velocity

Şekil 3.8 Scrum terminolojisi



Şekil 3.9
Scrum Sprint döngüsü



Şekil 3.10 Dağıtık Scrum

ÇEVİK YÖNTEMLERİ ÖLÇEKLEME

Çevik yöntemler aynı odada birlikte çalışabilecek ve resmi olmayan şekillerde iletişim kurabilecek küçük programlama takımları için geliştirilmiştir. Aslında küçük ve orta ölçekli sistemlerin ve yazılım ürünlerinin geliştirilmesinde kullanılmaktaydılar. Küçük şirketler, resmi süreçler ve bürokrasi olmadan, çevik yöntemleri hevesle erken benimseyenlerden oldular.

Çevik yöntemleri ölçekleme ile ilgili konular:

1. Yöntemlerin tek bir takım tarafından geliştirilmeyecek kadar büyük olan sistemler için büyük ölçeğe göre uyarlanması.

2. Yöntemlerin büyük şirketlerde çok uzun yıllar yazılım geliştirme deneyimi olan özelleşmiş geliştirme takımları için küçük ölçeğe göre uyarlanması.

Prensip	Pratik
Müşterinin katılımı	Bu geliştirme takımı ile zaman geçirme isteğine ve zamanına sahip ve tüm sistem paydaşlarını temsil edebilecek bir müşteriye sahip olmaya bağlıdır. Çoğunlukla müşteri temsilcileri zamanlarını başka işlere de ayırdıkları için geliştirme takımında tam zamanlı olarak yer alamazlar. Örneğin geliştirilecek sistemde denetleyici kurum gibi dış paydaşlar olduğu zaman, onların görüşlerini çevik takımlara yansıtmak zordur.
Değişimin benimsenmesi	Özellikle çok sayıda paydaşın olduğu sistemlerde değişiklikleri önceliklendirmek çok zor olabilir. Tipik olarak, her paydaş farklı değişikliklere farklı öncelikler atayacaktır.
Artımlı teslim	Hızlı artımlar ve kısa-vadeli planlama, uzun vadeli planlama döngüleri olan iş planları ve pazarlama ile daima örtüşmeyebilir. Pazarlama yöneticileri etkin pazarlama kampanyaları düzenleyebilmek için ürün özelliklerini aylar öncesinden bilmeye ihtiyaç duyabilir.
Sadeliğin sürdürülmesi	Teslim takvimlerinin baskısı nedeniyle, takım üyeleri arzu edilen sistem sadeleştirmelerini gerçekleştirmeye zaman bulamayabilir.
Süreçler değil, insanlar	Takım üyelerinin her biri, çevik geliştirme için tipik olan yoğun katılım sağlamaya uygun karakterde olmayabilir ve bu nedenle diğer takım üyeleri ile iyi bir şekilde iletişim kuramayabilir.

Şekil 3.11 Çevik prensipler ve organizasyon pratikleri

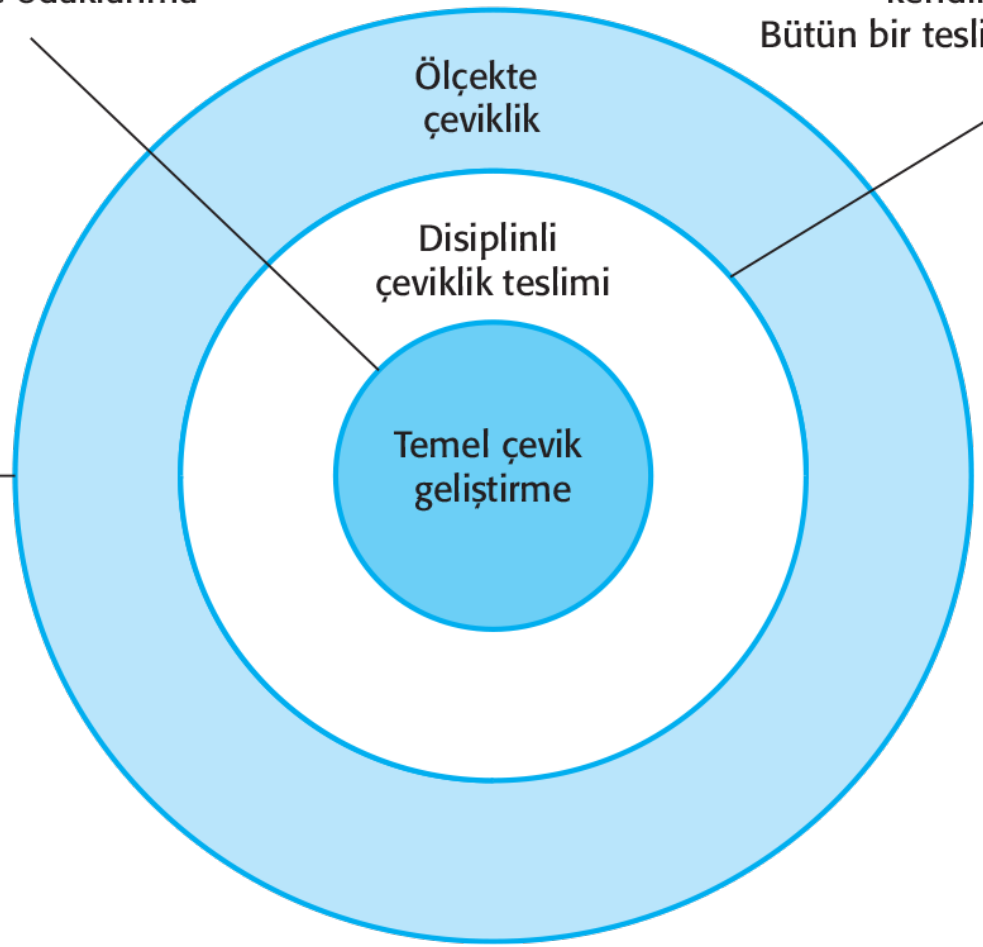
Şekil 3.12 Plan-güdümlü ya da çevik geliştirme seçimini etkileyen faktörler

Şekil 3.13 Büyük proje özellikleri

Temel çevik geliştirme
Değer-güdümlü yaşam döngüsü
Kendi kendine organize olan takımlar
Geliştirmeye odaklanma

Disiplinli çevik teslimi
Risk + değer-güdümlü yaşam döngüsü
Uygun çerçeve denetiminde kendi kendine organize olma
Bütün bir teslimat için yaşam döngüsü

Ölçekte çeviklik
Aşağıdaki ölçek faktörleri geçerli olduğunda disiplinli çevik teslimi:
Büyük takımlar
Coğrafi dağılım
Denetleyici kurumlara uyum
Alan karmaşıklığı
Organizasyon dağılımı
Teknik karmaşıklık
Organizasyonel karmaşıklık
Kurum disiplini



Şekil 3.14 IBM'in Çeviklik Ölçekleme modeli (IBM 2010)