

# **ÇEVİK (AGILE) YAZILIM GELİŞTİRME**

- Gereksinimler yazılım geliştirildikçe değişebilmektedir. Bu yüzden işin en başında tam ve doğru gereksinim belirlemek neredeyse imkansızdır.
- Bunun doğal bir sonucu olarak Şelale ya da spesifikasyon tabanlı bir süreç uzun bir süreçtir ve ürün teslimi gecikebilir.
- Daha hızlı yazılım geliştirmek için Uç Programlama (Beck 1999), Scrum (Schwaber ve Beedle 2001) ve DSDM (Stapleton 2003) gibi Çevik Yöntemler ortaya çıkmıştır.
- Küçük artırımlarla ilerleyip, her adımda yeni özelliklerin yayınlandığı, iki ya da üç haftalık periyotlarla müşteriye sunulduğu artırımlı geliştirme yöntemleridir.
- Çevik yönteme karar vermeden önce ihtiyacımızın bu yönteme uyup uymadığı belirlenmelidir. Değil ise plan güdümlü yöntem seçilmelidir.

# Çevik Yöntemler

- 1980'lerin ve 1990'ların yazılım tasarım yöntemlerinde yer alan genel ek yüklerden memnuniyetsizlik, çevik yöntemlerin yaratılmasına yol açtı. Bu yöntemler:
  - Tasarım yerine koda odaklanır
  - Yazılım geliştirme yinelemeli bir yaklaşıma dayalıdır
  - Belkide hiç kullanılmayacak olan belgeler ortadan kalkar
  - Çalışan yazılımı hızlı bir şekilde sunması ve bunu değişen gereksinimleri karşılayacak şekilde hızla geliştirmesi amaçlanmaktadır.
- Çevik yöntemlerin amacı, yazılım sürecindeki ek yükleri azaltmak (örneğin, dokümantasyonu sınırlandırarak) ve aşırı yeniden çalışma yapmadan değişen gereksinimlere hızlı bir şekilde yanıt verebilmektir.

# Çevik Yöntemlerin Özellikleri

- Program belirtimi (spesifikasyon), tasarımı ve uygulaması süreçleri iç içe geçmiştir.
- Sistem, sürüm belirtimi ve değerlendirmesine dahil olan paydaşların sürece katılımıyla, bir dizi sürüm veya artış olarak geliştirilir.
- Değerlendirme için yeni sürümler sık sık değerlendirilmelidir.
- Geliştirmeyi desteklemek için kapsamlı araç desteği (örn. Otomatik test araçları) kullanılmalıdır.
- Çalışma koduna odaklanarak minimum dokümantasyon sağlanmalıdır.

# ÇEVİK MANİFESTO

Bizler daha iyi yazılım geliştirme yollarını uygulayarak ve başkalarının da uygulamasına yardım ederek ortaya çıkartıyoruz.

Bu çalışmaların sonucunda:

Süreçler ve araçlardan ziyade bireyler ve etkileşimlere

Kapsamlı dokümantasyondan ziyade çalışan yazılıma

Sözleşme pazarlıklarından ziyade müşteri ile işbirliğine

Bir plana bağlı kalmaktan ziyade değişime karşılık vermeye değer vermeye kanaat getirdik.

Özetle, sol taraftaki maddelerin değerini kabul etmekle birlikte, sağ taraftaki maddeleri daha değerli bulmaktayız.

# Çevik Yazılımın 12 Prensibi

- En önemli önceliğimiz değerli yazılımın erken ve devamlı teslimini sağlayarak müşterileri memnun etmektir.
- Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir. Çevik süreçler değişimi müşterinin rekabet avantajı için kullanır.
- Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır.
- İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdırlar.
- Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır.
- Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüzyüze iletişimidir.
- Çalışan yazılım ilerlemenin birincil ölçüsüdür.
- Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.
- Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır.
- Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca arttırılması sanatı, olmazsa olmazlardandır.
- En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar.
- Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler.

# Çevik Yöntemlerin İlkeleri

Müşterinin Katılımı      Rollerini yeni gereksinimleri sağlamak önceliklendirmek ve sistem artımlarını değerlendirmek

Değişimin  
Benimsenmesi

Sistem değişiklikleri kabul edecek şekilde tasarlanmalıdır.

Artımlı Teslim

Müşterinin her artım için belirlediği gereksinimlerin dahil edildiği artımlarla geliştirilir.

Sadeliğin  
Sürdürülmesi

Hem geliştirilen yazılımda hem de yazılım geliştirme sürecinde sadeliğe odaklanılıp sistem karmaşıklığı azaltılmalıdır.

Süreçler Değil,  
İnsanlar

Takım üyeleri yeteneklerine göre kendi çalışma yaklaşımlarını geliştirme konusunda özgür olmalıdır.

# Çevik Yöntem Dezavantajları

- Gömülü sistem mühendisliği, büyük ve karmaşık sistemler gibi diğer yazılım türleri için uygun olmayabilir.
- Büyük şirketler için yasal zorunluluk olan kontrat prosedürüne sahip değildir.
- Belge eksikliği çevik yöntemlerle geliştirilmiş projelerin bakımı açısından sorun oluşturur. Bakım yapmak çok güç ve maliyetli olabilir. Bakım zamanı müşteri de tam zamanlı olarak sürece dahil olamayabilir.
- Yazılım bakımından çok yeni yazılım geliştirmekle ilgilenirler.
- Geliştirme takımının sürekliliği sağlanamayabilir.
- Yazılımın dünya geneline dağılmış takımlar tarafından geliştirilmesi gerekebilir.

- Müşteri temsilcileri, denetleyici kurum gibi dış paydaşlar olduğunda, onların görüşlerini çevik takıma tam yansıtamayabilir.
- Çok sayıda paydaşın olduğu sistemde öncelikleri belirlemek zordur.
- Artırımlı teslim olduğu için pazarlamacılar açısından kampanya düzenlemek amaçlı tüm sistem bilinemeyebilir.
- Teslim takvimleri sebebiyle sistem sadeleştirilmesi yapılamayabilir.
- Takım üyelerinin karakteri iletişim kurmaya müsait olmayabilir.

# Çevik Metodolojiler

- Uç Programlama (Extreme Programming)
- SCRUM
- Kanban
- Rational Unified Process
- Feature-Driven Development
- Test-Driven Development
- LEAN (Yalın) Development
- Dynamic System Development Methodology
- Microsoft Solution Framework

# Uç Programlama (UP-Extreme Programming - XP)

- Yazılım geliştirme yöntemini kökten değiştirmiştir.
- Müşteri katılımı esastır ve sistem için kabul testlerinden sorumludurlar.
- UP'de bir sistemin birden fazla sürümü farklı yazılım geliştiriciler tarafından bir gün içerisinde geliştirilip test edilebilir.
- Gereksinimler bir dizi görev olarak geliştirilen senaryolar ile ifade edilir.
- Programcılar ikili gruplar halinde kodlamadan önce kullanıcı öyküsüne ait testi kodlar. Tüm testler her derleme için çalıştırılmalıdır ve yapı yalnızca testler başarıyla çalıştırılırsa kabul edilir.
- Ürün artımları her 2 haftada bir müşterilere teslim edilir.
- Sade tasarım gerçekleştirilmesi esastır.

# Uç Programlamanın Temel Değerleri

- Sağlıklı yüz yüze iletişim
- Basitlik. Karmaşık çözümler yerine günün ihtiyaçlarına basit çözümler getirilmeli.
- Geri bildirim
- Cesaret. Projelerin üzerine yılmadın gidilmesi gerekmektedir. Başarısızlıktan korkman projenin yavaşlamasına neden olur.

# Uç Programlamada 12 Uygulama

- Planlama Oyunu
- Ekipte müşteri
- Müşteri Testi
- Basit Tasarım
- Çiftli Programlama
- Sürekli Tümlleştirme
- Kısa sürümler
- Yeniden yapılandırma
- Ortak sahiplenme
- Metaphor
- Kodlama standardı
- Kırk saat haftada

# Uç Programlama İlkeleri

Ortak Sahiplik

Herkes her şeyi değiştirebilir

Sürekli  
Bütünleştirme

Herhangi bir görev tanımlanır tanımlanmaz sistemle  
bütünleştirilir.

Artımlı Planlama

Gereksinimler “öykü kartlarına” kaydedilir ve öykülerin zaman ve  
görece önceliğine göre artırımlar belirlenir. Bu görevler zaman ve  
maliyet tahminlerinin temelidir.

Her Zaman Hazır  
Müşteri

Müşteri takıma destek vermek amacıyla tam zamanlı hazır  
olmalıdır.

Eş Programlama

Yazılımcılar geliştirme ve birbirini kontrol amaçlı eşler halinde  
çalışır.

# Uç Programlama İlkeleri

Kodun Yeniden  
Üretilmesi

Kod sürekli yeniden üretilerek iyileştirilmelidir. Böylece kod sadeleşir ve sürdürülebilirliği sağlar.

Sade Tasarım

Sadece mevcut gereksinimleri karşılayacak yeterli düzeyde tasarım yapılır.

Küçük Yayınlar

Sistem sık aralıklarla yayımlanır ve her artımda ilk yayımın üzerine fonksiyonlar eklenir.

Sürdürülebilir Hız

Fazla mesai hataları artıracağı için kaçınılmalıdır.

Test Önce  
Geliştirme

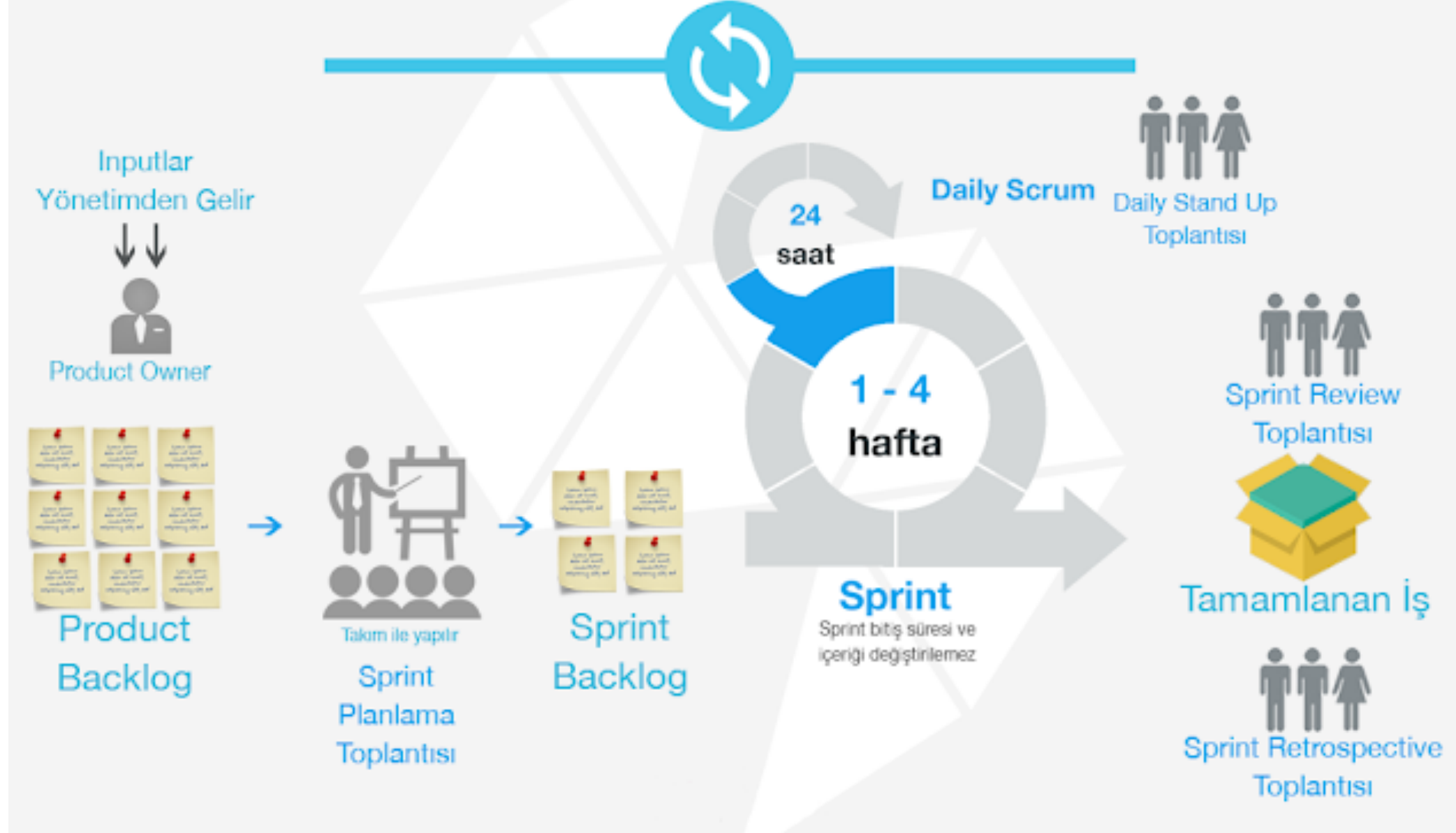
Fonksiyon geliştirilmeden testini geliştirmek için otomatik bir birimin test çerçevesi geliştirilir.

# Çevik Proje Yönetimi (SCRUM)

- Plan-güdümlü yaklaşımlar yöneticilerin projenin durumu, hedefe ulaşp ulaşmadığını, yeterli bütçe ile ve zamanında teslim edilip edilemeyeceğini bilmelerini sağlar.
- Scrum (Toplantı) çevik yazılım geliştirme yöntemidir. Eş programlama ve test önce geliştirme gibi pratiklerin kullanımı zorunlu değildir. Böylece şirketteki diğer pratiklerle bütünleştirilebilir.
- Scrum Çevik yöntemi, çevik projelerin organize edilebilmesi için bir çerçeve sunmak ve belirli bir düzeye kadar en azından ne olup bittiği ile ilgili görünürlük sağlamak amacıyla geliştirilmiştir.
- Başta aynı lokasyonda çalışan yazılımcılar için düşünülse de dağıtık geliştirme çevreleri ve çoklu takımlar için de uyarlamaları mevcuttur.

- Scrum'da üç aşama vardır.
  - İlk aşama, proje için genel hedefleri belirlediğiniz ve yazılım mimarisini tasarladığınız bir taslak planlama aşamasıdır.
  - Bunu, her döngünün sistemin bir artışını geliştirdiği bir dizi sprint döngüsü izler.
  - Proje kapanış aşaması projeyi tamamlar, sistem yardım çerçeveleri ve kullanıcı kılavuzları gibi gerekli belgeleri tamamlar ve projeden öğrenilen dersleri değerlendirir.

# SCRUM PROJE YÖNETİMİ SÜRECİ



# Scrum Terminolojisi

|                                                 |                                                                                                   |
|-------------------------------------------------|---------------------------------------------------------------------------------------------------|
| Geliştirme Takımı                               | 7 kişiden fazla olmayan, kendi kendine organize olabilen grup                                     |
| Potansiyel Olarak Teslim Edilebilir Ürün Artımı | Bir Sprint sonunda teslim edilen yazılım artımı                                                   |
| Ürün İş Listesi                                 | Scrum takımının uğraşmak zorunda olduğu yapılacak iş listesi                                      |
| Ürün Sahibi                                     | Ürün iş listesini düzenli olarak inceleyen kişi ya da gruplar                                     |
| Scrum                                           | Scrum takımının günlük toplantısı                                                                 |
| Scrum Uzmanı                                    | Scrum sürecinin takip edildiğinden emin olmaktan sorumlu kişi                                     |
| Sprint                                          | 2-4 hafta arasında süren yazılım geliştirme artımı                                                |
| Hız                                             | Bir Sprint içerisinde takımın, ürün iş listesi içerisinden ne kadarlık iş yapabileceğinin tahmini |

# Scrum'un Pozitif Yönleri

- Ürün, paydaşların dahil olduğu anlaşılabilir ve yönetilebilir küçük parçalara bölünebilir.
- Belirsiz gereksinimler süreci tıkamaz.
- Projede olan biten her şey tüm takım tarafından bilinir. Takım iletişimi ve morali iyileşir.
- Müşteriler artımları tam zamanında teslim alabilir ve ürünün nasıl çalıştığına dair geri bildirim alabilir. Son dakika sürprizleriyle karşılaşmazlar.
- Müşteri ve yazılımcılar arasında güven sağlanır ve herkesin projenin başarısını istediği pozitif kültür oluşturulur.

# Kanban

- İş süreçlerini görselleştirmek ve verimliliği artırmak amacıyla kullanılan bir iş yönetim metodolojisidir. Japonca “görsel kart” anlamına gelir ve Toyota’nın üretim süreçlerinde verimliliği artırmak için geliştirilmiştir.

- Kanban Panosu

- Kanban panosu, iş süreçlerinin görselleştirildiği temel araçtır. Pano genellikle sütunlar halinde bölünmüştür ve her sütun işin belirli bir aşamasını temsil eder.
- En yaygın kullanılan sütunlar “Yapılacak (To Do)”, “Yapılıyor (In Progress)” ve “Tamamlandı (Done)” şeklindedir. Ancak süreçlere göre daha fazla sütun eklenebilir (örneğin, “Beklemede”, “Test Ediliyor” gibi).

- Kartlar

- Her iş birimi veya görev, bir kart ile temsil edilir. Kartlar, yapılacak işin detaylarını içerir ve sürecin hangi aşamasında olduğunu göstermek için farklı sütunlara taşınır.
- Kartlar genellikle işin adı, yapılacak görevler, öncelik düzeyi, sorumlu kişi ve işin durumu gibi bilgileri içerir.

- İş Akışının Görselleştirilmesi

- Kanban’ın temel amacı, iş akışını görselleştirerek tüm sürecin şeffaf olmasını sağlamaktır. Böylece ekip üyeleri hangi işlerin yapıldığını, hangi işlerin beklediğini ve sürecin neresinde darboğazlar olduğunu görebilir.

- Çekme Sistemi (Pull System)
  - Kanban’da iş akışı, bir çekme sistemi ile düzenlenir. Bu, bir işin bir sonraki aşamaya geçmeden önce tamamlanması gerektiği anlamına gelir. Bu sayede aynı anda çok fazla iş yapılmaya çalışılmasının önüne geçilir ve iş yükü dengelenir.
- İş Hacmi Sınırlandırma (Work in Progress Limits)
  - Kanban, bir ekibin aynı anda üzerinde çalışabileceği iş miktarını sınırlandırarak iş yükünü kontrol altında tutmayı hedefler. Her sütun için bir “İş Hacmi Sınırı” (WIP limit) belirlenir. Bu sınırlar, çok fazla işin aynı anda yapılmasını önleyerek, ekibin verimliliğini artırır ve odaklanmayı sağlar.
- Sürekli İyileştirme (Kaizen)
  - Süreçlerin düzenli olarak gözden geçirilmesi, darboğazların ve verimsizliklerin tespit edilmesi ve gerekli değişikliklerin yapılması, Kanban’ın başarısında kritik öneme sahiptir. Ekipler, süreçleri analiz ederek ve WIP limitlerini ayarlayarak iş akışını optimize edebilir.

# Kanban ve Scrum Arasındaki Farklar

- **Zaman Çerçevesi:** Scrum, belirli zaman dilimlerinde (sprintler) çalışırken, Kanban sürekli akış üzerine odaklanır ve belirli bir zaman çerçevesi yoktur.
- **Roller ve Sorumluluklar:** Scrum, ürün sahibi, scrum master ve geliştirme ekibi gibi belirli rolleri içerir. Kanban'da ise bu roller zorunlu değildir.
- **Değişiklik Yönetimi:** Scrum'da sprint sırasında değişiklikler yapılmazken, Kanban değişiklik konusunda daha esnektir.

# Rational Unified Process (RUP)

- Büyük ve karmaşık projelerde yazılım geliştirme sürecini yapılandırmak için yaygın olarak tercih edilir.
- RUP, IBM tarafından geliştirilen ve birçok yazılım mühendisliği ilkesini içeren bir metodolojidir.
- RUP'un amacı, yazılım geliştirme sürecini daha verimli ve kaliteli hale getirmektir.
- RUP, yazılım geliştirme sürecini dört ana faza böler. Her faz, farklı hedeflere ve çıktılara sahiptir.
- RUP'ta her faz, bir veya birden fazla iterasyon içerir. Her iterasyon, belirli bir süre boyunca (genellikle 2-6 hafta) sürer ve sonunda belirli bir çıktının elde edilmesi hedeflenir.

- **İteratif ve Artımlı Geliştirme:** RUP, projenin tek bir büyük döngü yerine, daha küçük, yönetilebilir döngülerde (iterasyonlar) geliştirilmesini öngörür. Her iterasyon sonunda belirli bir yazılım parçası üretilir ve geliştirme süreci adım adım ilerler.
- **Risk Yönetimi:** RUP, projeye başlarken ve geliştirme süreci boyunca riskleri belirleyip yönetmeyi önemser. Riskler, erken aşamada ele alınarak proje başarısızlığı olasılığı azaltılır.
- **Modeller Üzerinden Tasarım:** Yazılımın gereksinimlerinin, yapısının ve davranışının modellerle ifade edilmesi ve belgelenmesi teşvik edilir. Bu modeller, yazılımın anlaşılabilirliğini artırır ve geliştirme sürecini kolaylaştırır.

# RUP Fazları

- Başlatma (Inception) Fazı:
  - Bu fazın amacı, projenin fizibilitesini değerlendirmek, temel gereksinimleri belirlemek ve projeyi başlatmak için gerekli olan ilk analizleri yapmaktır. Başlangıçtaki riskler belirlenir ve projenin devam edip etmeyeceğine karar verilir.
- Geliştirme (Elaboration) Fazı:
  - Sistemin mimari modeli oluşturulur ve temel işlevler üzerinde çalışılır. Teknik riskler bu aşamada giderilir ve proje planı güncellenir.
- Yapım (Construction) Fazı:
  - Bu aşama, yazılımın geliştirilmesi ve sistemin işlevsel bir sürümünün oluşturulmasına odaklanır. Gereksinimler doğrultusunda kod yazılır, birimler entegre edilir ve sistemin birinci versiyonu ortaya çıkar.
  - Yapım fazında sürekli testler ve doğrulamalar yapılarak, hataların erken aşamada tespit edilmesi sağlanır.
  - Çalışan bir yazılım ürünü ortaya çıkar ve kullanıcı dokümantasyonu oluşturulur.
- Geçiş (Transition) Fazı:
  - Bu faz, sistemin müşteriye teslim edilmesi ve son kullanıcıların kullanımına sunulması ile ilgilidir. Yazılım, müşteri ortamına kurulur ve son kullanıcı eğitimi verilir. Kullanıcı geri bildirimleri toplanır ve gerekirse iyileştirmeler yapılır.

# RUP'un Avantajları

- **Esneklik:** İteratif yapısı sayesinde değişen gereksinimlere hızlı uyum sağlar.
- **Kalite Yönetimi:** Süreç boyunca yapılan testler ve doğrulamalar, yazılımın kalitesini artırır.
- **Risk Azaltma:** Riskler erken aşamalarda ele alınarak projedeki belirsizlikler minimize edilir.
- **Dokümantasyon:** RUP, projelerde kapsamlı bir dokümantasyon sağlar, bu da projeyi daha izlenebilir hale getirir.

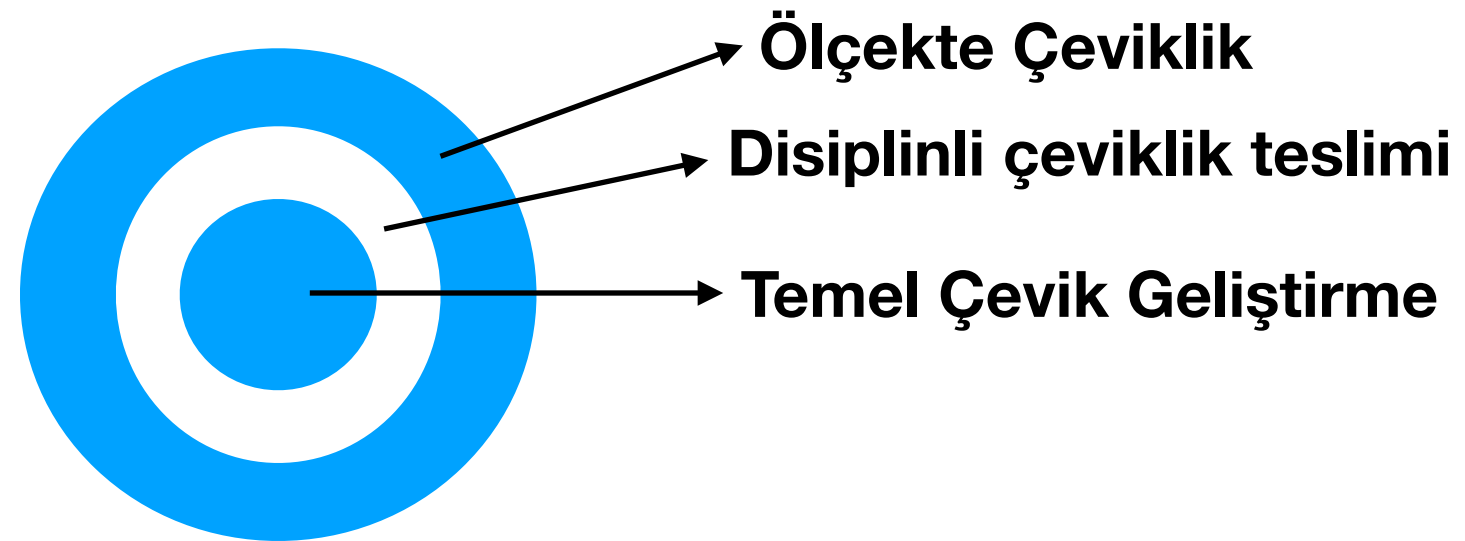
# RUP'un Dezavantajları

- **Karmaşıklık:** RUP, özellikle küçük projelerde karmaşık ve maliyetli olabilir.
- **Zaman Alıcı:** Detaylı dokümantasyon ve süreçlerin yönetimi, daha uzun süre alabilir.
- **Dışarıdan Eğitim Gereksinimi:** RUP'u uygulamak için ekiplerin RUP metodolojisi konusunda eğitilmesi gerekebilir.

# Çevik Ölçekleme Modeli (Agile Scaling Model)

- "Ölçek büyütme", küçük bir ekip tarafından geliştirilemeyen büyük yazılım sistemleri geliştirmek için çevik yöntemlerin kullanılmasıyla ilgilidir.
- "Ölçeklendirme", çevik yöntemlerin uzun yıllara dayanan yazılım geliştirme deneyimine sahip büyük bir kuruluştaki nasıl uygulanabileceğiyle ilgilidir.
- Çevik yöntemleri ölçeklendirirken, çevik temelleri korumak önemlidir:
  - Esnek planlama, sık sistem sürümleri, sürekli entegrasyon, test odaklı geliştirme ve iyi ekip iletişimi.

- IBM’de, çevik yöntemlerin büyük ölçekte kullanımına ilişkin Çevik Ölçekleme Modeli’ni geliştirmiştir.



- ÇÖM’de ölçeklemenin basamaklı bir süreç olduğu ifade edilir, geliştirme takımları, temel çeviklik pratiklerinden Disiplinli Çeviklik Teslimi olarak isimlendirilen basamağa hareket eder.
- Bu basamak, pratiklerin disiplinli bir organizasyon düzenine uyarlanmasını içerir. Takımların sadece geliştirmeye odaklanamayacağını aynı zamanda gereksinimler ve mimari tasarım gibi yazılım mühendisliği sürecinin diğer aşamalarını da ele alması gerektiğini ifade eder.
- Son basamak büyük projelerdeki doğal karmaşıklığın ele alındığı Ölçekte Çevikliktir. Dağıtık geliştirme, karmaşık eski kod ortamları gibi faktörler ele alınır.

## RUP

## Çevik Ölçekleme Modeli

Çevik (Agile) prensiplerine dayanır ve çevik yaklaşımları büyük ölçekli projelere veya organizasyonlara uygulamayı amaçlar.

Planlı ve yapılandırılmış bir süreç modelidir. Yazılım geliştirme sürecini dört ana faza böler.

Kısa iterasyonlarla (örneğin, 2-4 haftalık sprintler) sürekli çalışan yazılım sunmayı hedefler

İteratif ve artımlı bir yaklaşım benimser ancak fazlara ayrılmış ve daha uzun süreli iterasyonlar içerebilir

Yeterli dokümantasyon

Kapsamlı dokümantasyon

Ekipler esnek ve kendini organize eden bir yapıya sahiptir

Belirli roller ve disiplinler vardır

Değişen gereksinimlere hızlı ve esnek bir şekilde uyum sağlar

Değişikliklerin entegrasyonu daha fazla değerlendirme ve planlama gerektirebilir

# Test-Driven Development

- Testlerin koddan önce yazıldığı bir yaklaşımdır.
- Bu metodoloji, yazılımın kalitesini artırmak, hataları erken tespit etmek ve daha güvenilir bir kod tabanı oluşturmak için kullanılan bir tekniktir.
- Üç adımlı bir döngüyle uygulanır: **Red (Kırmızı)**, **Green (Yeşil)** ve **Refactor (Yeniden Düzenleme)**.

- İlk olarak, henüz var olmayan bir işlevsellik için başarısız olacak (kırmızı durumda olacak) bir test yazılır. Bu test, yazılacak olan kodun nasıl çalışması gerektiğini belirler ve geliştiricinin amacını ortaya koyar.
- İkinci adımda, testin geçmesini sağlayacak kadar kod yazılır. Bu aşamada kodun basitliği ve işlevselliği ön plandadır; amaç sadece testi başarılı bir şekilde geçirmektir.
- Son adımda, yazılan kodun kalitesi artırılır ve gerektiğinde kodun yapısı düzenlenir. Bu, kodun temizlenmesi, tekrar eden bölümlerin kaldırılması veya daha iyi bir tasarıma dönüştürülmesi anlamına gelir.

# Feature-Driven Development

- Müşteri tarafından tanımlanan özellikler etrafında yazılım geliştirmeyi odak noktası haline getiren bir yaklaşımdır.
- Tasarım odaklı işleyen bir çevik süreçtir.
- 1997 yılında Jeff De Luca ve Peter Coad tarafından büyük ölçekli ve karmaşık yazılım projelerini daha verimli yönetmek için geliştirilmiştir.
- FDD, yazılım geliştirme sürecini küçük, yönetilebilir parçalara ayırarak yazılımın kademeli olarak oluşturulmasını sağlar.
- FDD'nin temel amacı, iş hedeflerine uygun olarak çalışan yazılım özelliklerini düzenli ve sık aralıklarla teslim etmektir.

- FDD'nin 5 Süreci:
  - Genel Model Geliştirme
  - Özellik Listesinin Oluşturulması
  - Özelliklere Göre Plan Yapma
  - Özellik Tasarımı
  - Özellik Geliştirme

# Lean (Yalın) Development

- Lean Development, yedi temel ilke üzerine kuruludur. Bu ilkeler, yazılım geliştirme sürecini daha verimli ve etkili hale getirmek için rehberlik eder:
  - İsrafı Azalt
  - Öğrenmeyi Geliştir
  - Kararları Geç Al
  - Hızlı Teslim Et
  - Ekibi Güçlendir
  - Bütün Sistemi Optimize Et
  - Bütünlüğü Koruyarak İnşa Et

- Lean Development ve Scrum, müşteri odaklılık ve esneklik açısından benzerlik gösterir. Ancak Lean Development, daha çok israfın azaltılması ve süreç optimizasyonu üzerinde dururken, Scrum daha çok sprint döngüleri ve iteratif gelişim üzerine odaklanır.
- Lean Development, Kanban ile birlikte sıkça kullanılır çünkü ikisi de israfın azaltılması ve sürekli iyileştirme prensiplerini paylaşır. Kanban, işleri görselleştirerek Lean Development'ın temel ilkelerini destekler.

# Dynamic System Development Methodology

- Yazılım projelerinde hızlı, esnek ve etkili bir geliştirme süreci sunmayı amaçlar.
- Projelerde değişen gereksinimlere uyum sağlamak için geliştirilmiştir ve yazılım geliştirme süreçlerini planlı ve yapılandırılmış bir şekilde yönetmeyi hedefler.

- Temel İlkeleri:
  - İş Hedeflerine Odaklanmak
  - Zaman ve Bütçeyi Sabit Tutmak
  - İşbirliğine Dayalı Çalışma
  - Kaliteyi Kontrol Etmek
  - Artımlı ve İteratif Geliştirme
  - Değişikliklere Karşı Açık Olmak
  - Gereksinimleri Yeterli Düzeyde Belirlemek
  - Sürekli Geri Bildirim ve İyileştirme

- Scrum ve Kanban gibi diğer çevik metodolojilerden farklı olarak, daha fazla yapılandırılmış bir süreç sunar. Bu, projelerin planlı ve kontrol altında yürütülmesini sağlar.
- Belirli bir zaman ve bütçeyi sabit tutma ilkesi, diğer çevik yaklaşımlardan ayrılan önemli bir özelliğidir. Bu, projelerde daha fazla yönetim ve kontrol imkanı sağlar.

# Microsoft Solution Framework

- Yazılım geliştirme ve dağıtım süreçlerini optimize etmek, riskleri azaltmak ve ekiplerin daha verimli çalışmasını sağlamak amacıyla tasarlanmıştır.
- Esneklik ve ölçeklenebilirlik sağlayan bir yapıya sahiptir, bu nedenle çeşitli büyüklükteki projelerde kullanılabilir.

- İki ana bileşen üzerine kuruludur: **Süreç modeli** ve **ekip modeli**.
- Süreç Modeli:
  - Vizyon ve Kapsam
  - Planlama
  - Geliştirme
  - Dengeleme
  - Dağıtım
- Ekip Modeli:
  - Ürün Yöneticisi
  - Program Yöneticisi
  - Geliştirici
  - Test Uzmanı
  - Yayın Mühendisi
  - Kullanıcı Deneyimi Uzmanı

- MSF, daha yapılandırılmış bir süreç sunar ve belirli aşamaları olan bir süreç modeline sahiptir. Scrum ise, sprint döngüleri ve belirli rol tanımları ile daha çevik bir yaklaşımı temsil eder.
- MSF’de risk yönetimi ve proje planlaması daha ayrıntılı bir şekilde ele alınırken, Scrum’da odak noktası sürekli geri bildirim ve kısa döngülerde gelişimdir.
- MSF, belirli süreç aşamalarına ve rollerine sahip bir model sunarken, Kanban daha esnek bir iş akışı yönetimi sunar ve işleri görselleştirmek için kullanılan bir araçtır.
- Her iki çerçeve de projelerin yönetilmesi ve iş akışının optimize edilmesi için kullanılabilir, ancak MSF daha planlı ve yapılandırılmış bir yaklaşım sunar.

# Kaynakça

- Ian Sommerville, Yazılım Mühendisliği
- Fatih Yücalar, Emin Borandağ - Yazılım Mühendisliğinde Modern Yaklaşımlar
- <https://agilemanifesto.org/>
- <https://medium.com/@secilcor/scrum-nedir>
- Doç. Dr. Sedat AKLEYLEK'in ders notları