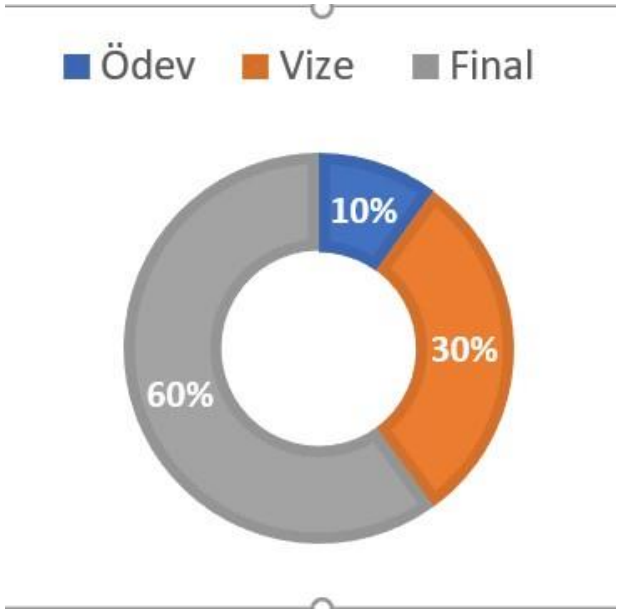
Yazılım Mühendisliği
1906003082015

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

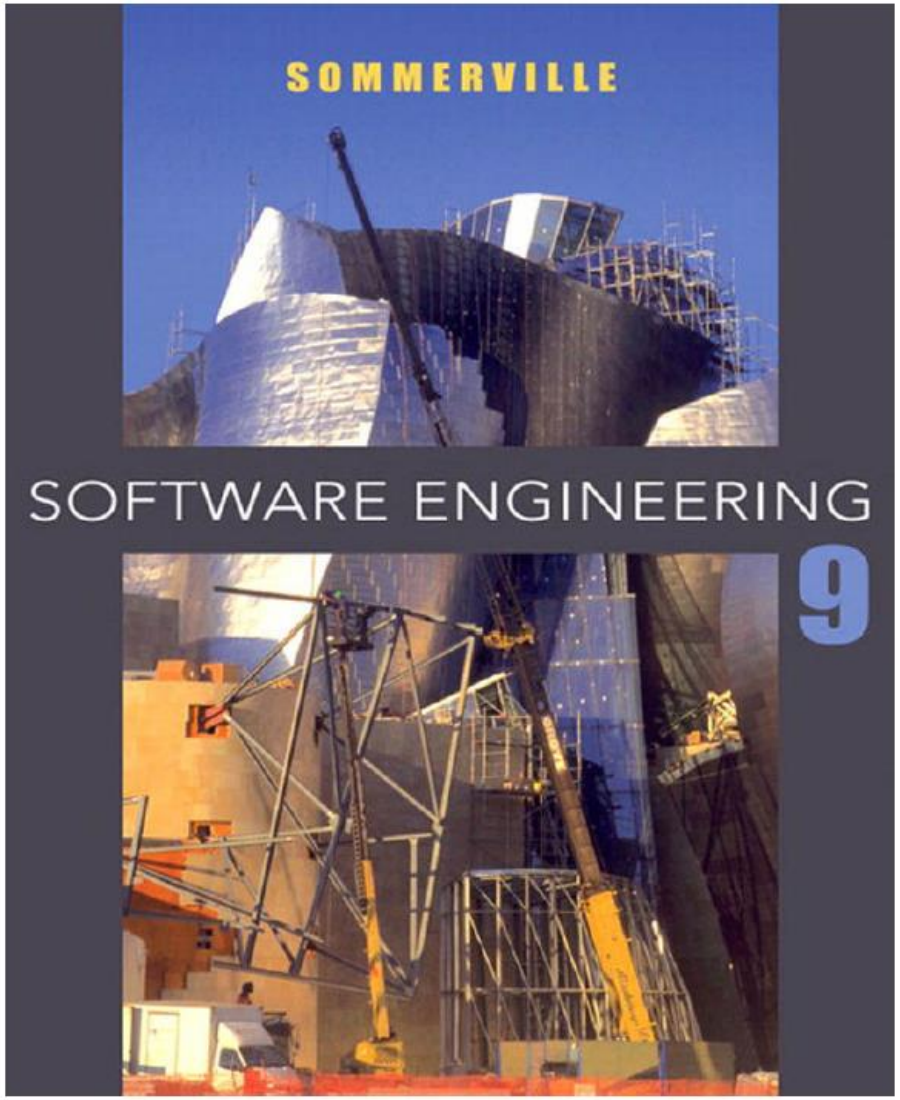
Giriş

Ders Günü ve Saati:
Salı: 09:15-13:00
Devam zorunluluğu %70



HAFTA	KONULAR
Hafta 1	Yazılım Mühendisliğine Giriş
Hafta 2	Yazılım Geliştirme Süreç Modelleri
Hafta 3	Yazılım Gereksinim Mühendisliği
Hafta 4	Yazılım Mimarisi
Hafta 5	Nesneye Yönelik Analiz ve Tasarım
Hafta 6	Laboratuar Çalışması: UML Modelleme Araçları
Hafta 7	Yazılım Test Teknikleri
Hafta 8	Ara Sınav
Hafta 9	Yazılım Kalite Yönetimi
Hafta 10	Yazılım Bakımı - Yeniden Kullanımı ve Konfigürasyon Yönetimi
Hafta 11	Yazılım Proje Yönetimi (Yazılım Ölçümü ve Yazılım Proje Maliyet Tahmin Yöntemleri)
Hafta 12	Yazılım Proje Yönetimi (Yazılım Risk Yönetimi)
Hafta 13	Çevik Yazılım Geliştirme Süreç Modelleri
Hafta 14	Yazılım Süreci İyileştirme, Yeterlilik Modeli (CMM)

Kaynaklar



2.BÖLÜM

Yazılım Geliştirme Süreç Modelleri

Yazılım Geliştirme Süreci Modelleri

- Yazılım geliştirme süreç modelleri, yazılım geliştirme süreci adımlarının hangi sırada ve düzende uygulanacağını tanımlar. Başlıca yazılım geliştirme süreç modelleri,
  - Gelişi güzel (Ad hoc)
  - Klasik doğrusal akış (Waterfall)
  - V Model
  - Evrimsel Süreç Modelleri
    - Artımlı (incremental) model
    - Helezon (Spiral) Model
  - Çevik (Agile) Modelleri
    - Özelliğe Dayalı Geliştirmeler Modeli
    - XP (Extreme Programming)
- Yazılım geliştirme süreç model tiplerinin seçimi, uygulama alanına, yazılım geliştiren ekibin yapısına ve kullanıcıların durumuna göre gerçekleştirilir.

GELİŞİ GÜZEL GELİŞTİRME

- Bu yaklaşım genellikle kaotik, düzensiz ve plansızdır veya planlandıkları zaman kolayca bozulabilirler. Tahminler, zamanlamalar yapılmasa bile pratikte çok az gerçekleşirler.
  - Herhangi bir model ya da yöntem yok.
  - Geliştiren kişiye bağımlı (belli bir süre sonra o kişi bile sistemi anlayamaz ve geliştirme güçlüğü yaşar).
  - İzlenebilirliği ve bakımı oldukça zor.
  - Genellikle tek kişilik üretim ortamı.
  - Basit programlama.

BAROK MODEL

- İnceleme
- Analiz
- Tasarım
- Kodlama
- Modül Testleri
- Alt sistem Testleri
- Sistem Testi
- Belgeleme
- Kurulum

Yaşam döngüsü temel adımlarının doğrusal bir şekilde geliştirildiği model.

70'li yıllar.

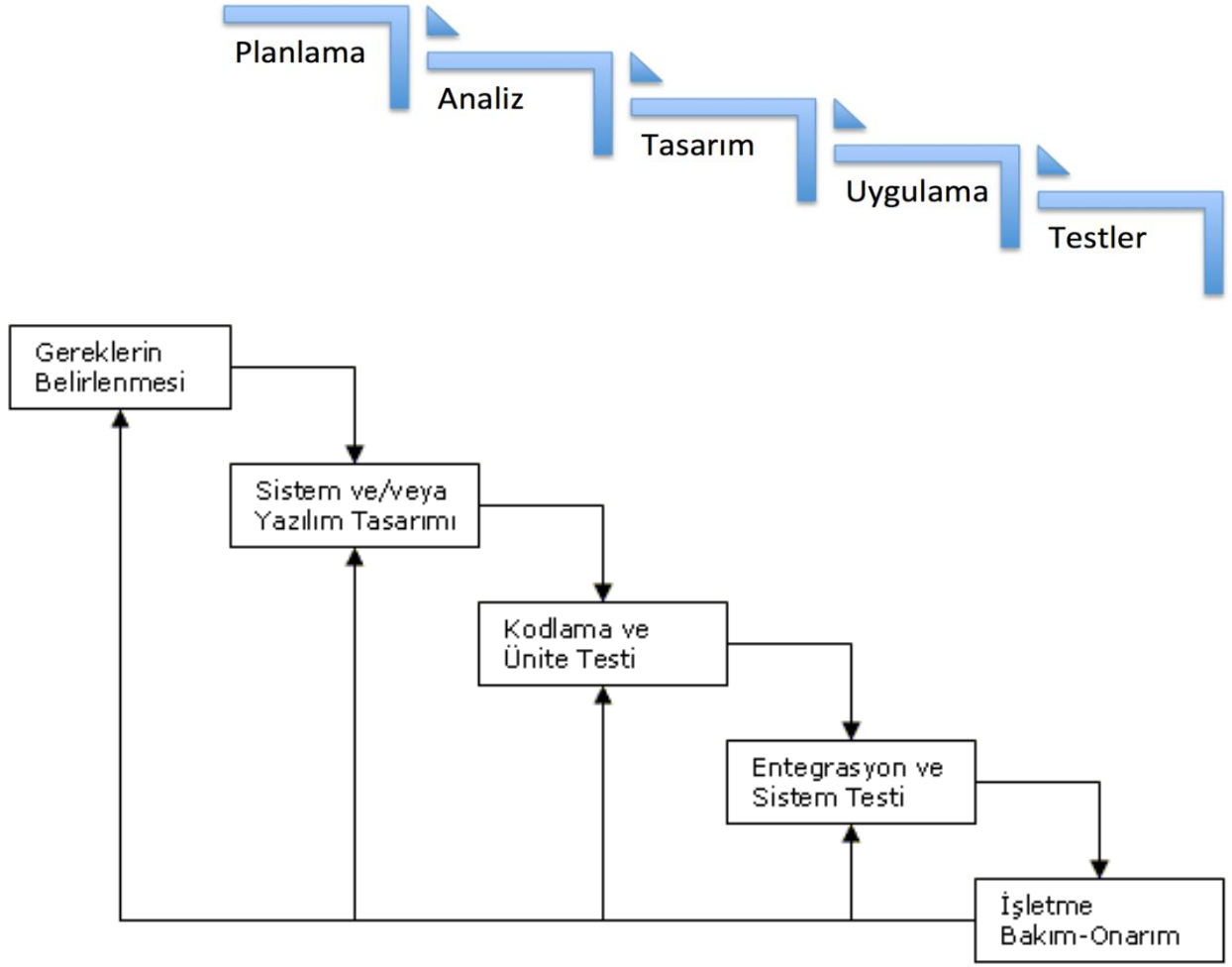
Belgelemeyi ayrı bir süreç olarak ele alır, ve yazılımın geliştirilmesi ve testinden hemen sonra yapılmasının öngörür.

Halbuki, günümüzde belgeleme yapılan işin doğal bir ürünü olarak görülmektedir.

Aşamalar arası geri dönüşlerin nasıl yapılacağı tanımlı değil.

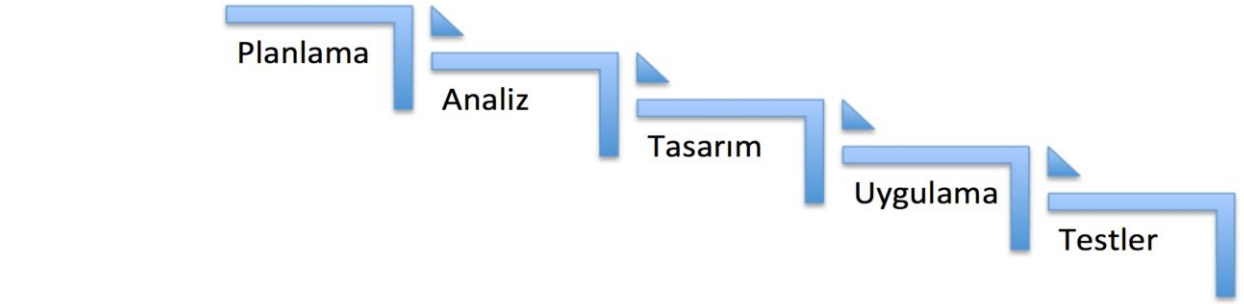
Şelale Modeli (Waterfall Model)

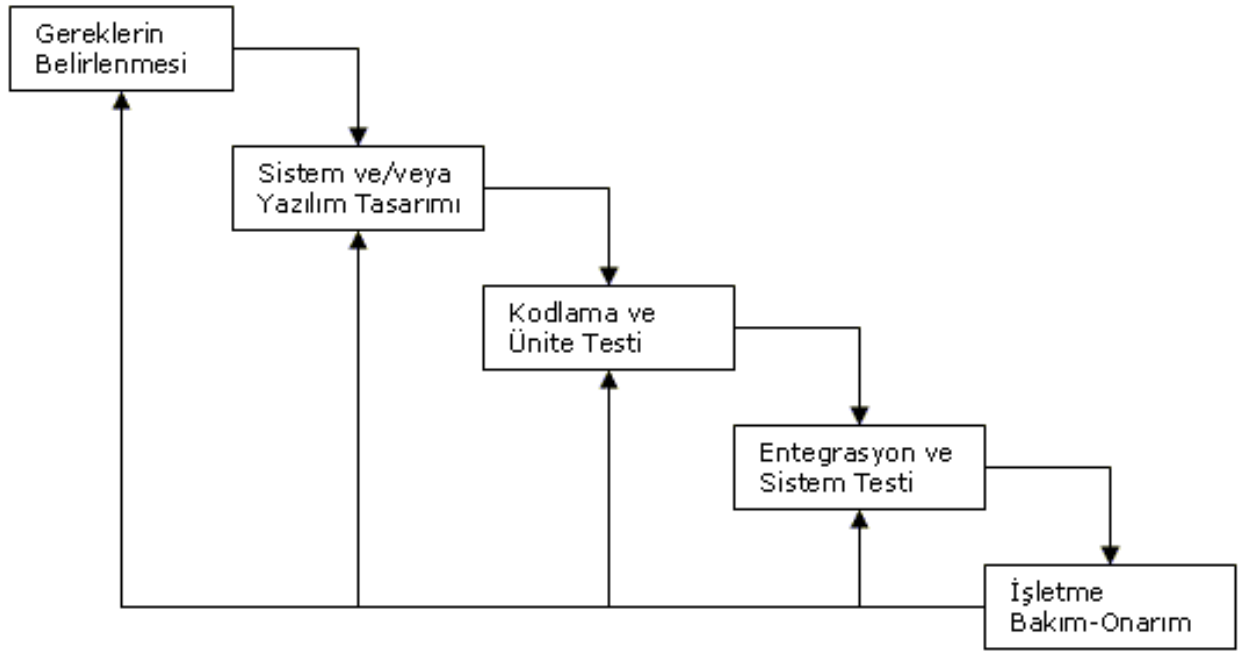
- İlk yayınlanan yazılım geliştirme süreç modelidir
- Proje yönetim modelleri genelde şelale modeli (waterfall model) ile başlar. Yazılım mühendisliğinde ilk anlatılan modellerden birisi budur.
- Statiktir.?!!
- Aşamaları:
- Bir sonraki aşama bir önceki aşama bitmeden başlamamaktadır.
- Dokümantasyon odaklıdır. (Maliyetli)
- Problemler son aşamaya bırakılır.



Şelale Modeli (Waterfall Model)

- Yaşam döngüsü temel adımları baştan sona en az bir kez izleyerek gerçekleştirilir.
- Barok modelin aksine belgeleme işlevini ayrı bir aşama olarak ele almaz ve üretimin doğal bir parçası olarak görür.
- İyi tanımlı projeler ve üretimi az zaman gerektiren yazılım projeleri için uygun bir modeldir.

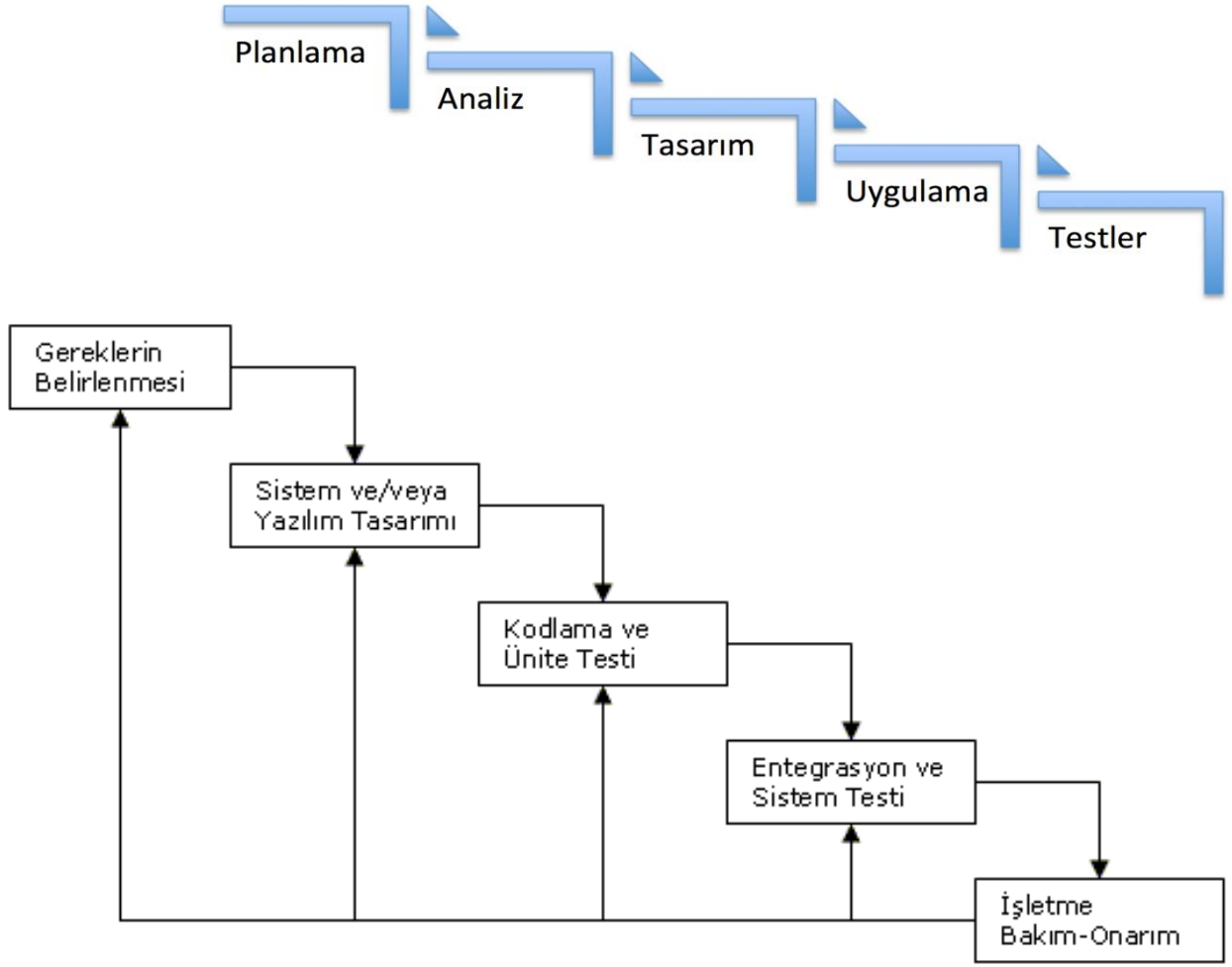




Şelale Modeli (Waterfall Model)

Tarihsel Gelişim:

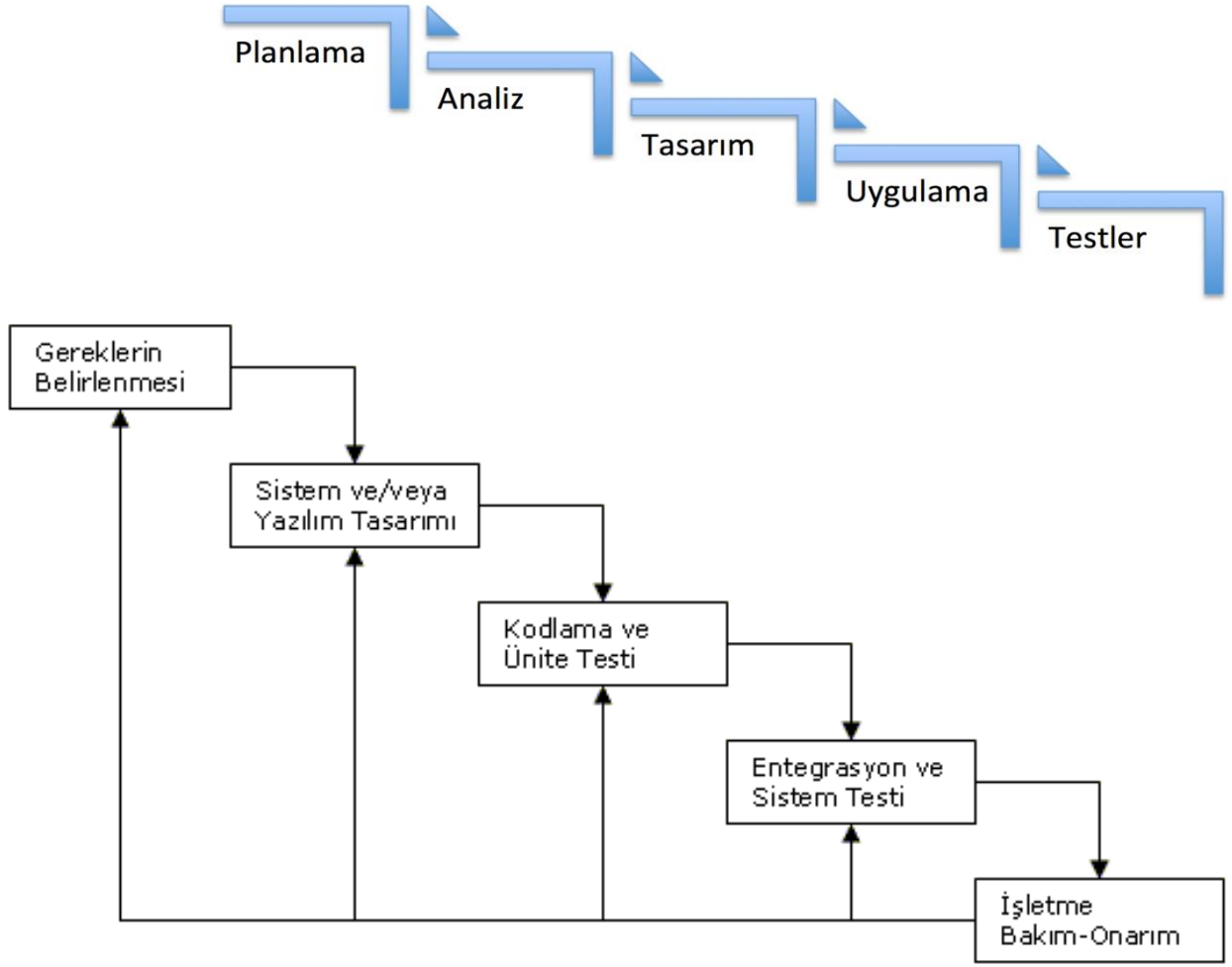
- İnşaat ve üretim alanları
- Benington, NMCAP, 1956 (Fikir)(«Production of Large Computer Programs»)
- Royce, 1970 (Fikir) («Managing the development of large software system»--» Generate System Requirements, Software Requirements, Perform Analysis, Program Design, Coding, Testing, then Operations.»)
- Barry W. Boehm, 1983 (Aşamaları detalandırılımıştır.)(«Software Engineering Economics»)



Şelale Modeli (Waterfall Model)

- Projeyi bu modelde olduğu gibi esnek olmayan safhalara ayirmak doğru değildir. Bu nedenle Şelale modelinin müşteri ihtiyaçlari tam ve çok iyi anlaşildiğinda kullanilmasi tavsiye edilmektedir. Sonuç olarak büyük sistemlerde, mühendislik projelerinde kullanilmaktadir.

- En basit, en etkili ve diğer modelleri temeli olan yöntemdir.



Şelale Modeli (Waterfall Model)-ELEŞTİRİ (SEKER-2015)

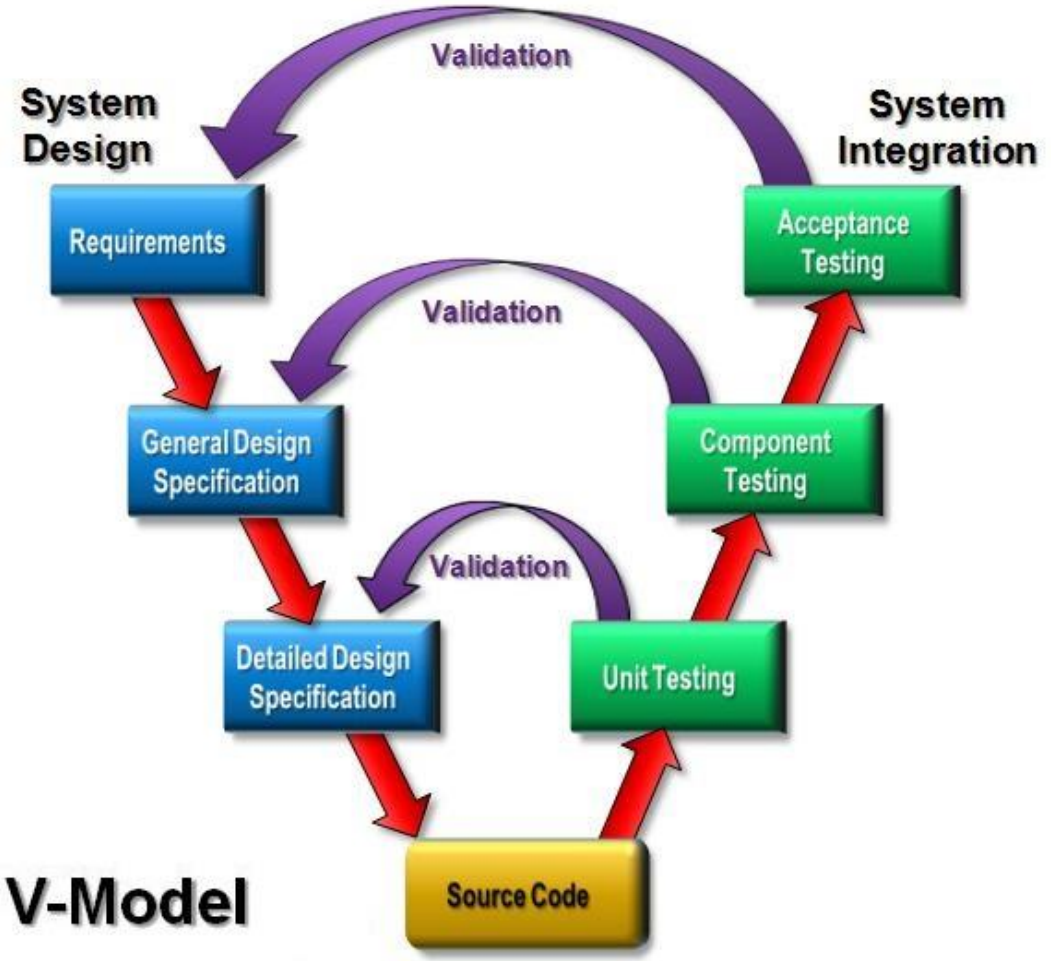
- Şelale modelinin en zayıf yönü olan ve geri dönüşe izin vermeyen yapı olmasıdır.
- David Parnas, “Mantikli bir sistemi aldatmak” (Parnas, 1986):
- “Çoğu sistemin detayları ancak sistem üzerinde çalışmaya başladıktan sonra anlaşılmaktadır. Yapılan işlemlerin bazıları, hata yaptığımızı göstermekte ve geri dönmemizi gerektirmektedir”
  - Bilişim teknolojileri konusunda tecrübesi olmayan paydaşların teknolojik olarak neler
  - yapılabileceğinden haberdar olmaması yüzünden isteklerinin değişime açık olması
  - Proje ihtiyaçlarının zaman içerisinde değişime açık olması
  - Proje paydaşlarının isteklerini doğru şekilde aktaramaması
  - Proje ekibinin acemiliği veya kodlama öncesi aşamalardaki kaynak eksiklikleri (nitelik ve nicelik
  - anlamındaki eksikler)
  - Projenin ilerideki şeklinin önceden kestirilememesi

Şelale Modeli (Waterfall Model)-ARTILARI (SEKER-2015)

- iki kere ölçüp bir kere biçmek yaklaşımının kurumda uygulanmasını sağlaması
- Proje dokümantasyonu hazırlanmaya zorlaması
- Hataların hangi aşamalarda ve kimin tarafından kaynaklandığının bulunmasını kolaylaştırması
- Her sonraki aşamaya geçmeden önce, önceki aşamanın tamamlanması, bu sayede önceki aşamada ayrılan kaynakların serbest kalması ve farklı projelere aktarılabilmesi

V MODEL

- Adımlar V şeklini oluşturduğu için V-Şeklinde Model denilmiştir.
- Şelale modeline benzer: ihtiyaç belirleme, üst seviye ve alt seviye tasarımlar vardır.
- Üst seviye de daha genel bakışa göre tasarım yapılır.
- Bu yaklaşımın diğer bir özelliği de, şekilden görüleceği üzere aynı hizada olan aşamaların birbirini karşılamasıdır. Detaylı tasarım yapıldı. Bunun hemen karşısında kodlama bittikten sonra birim testleri yapılır.



V MODEL

Sol taraf üretim, sağ taraf sınama işlemleridir.

V süreç modelinin temel çıktıları;

- Kullanıcı Modeli

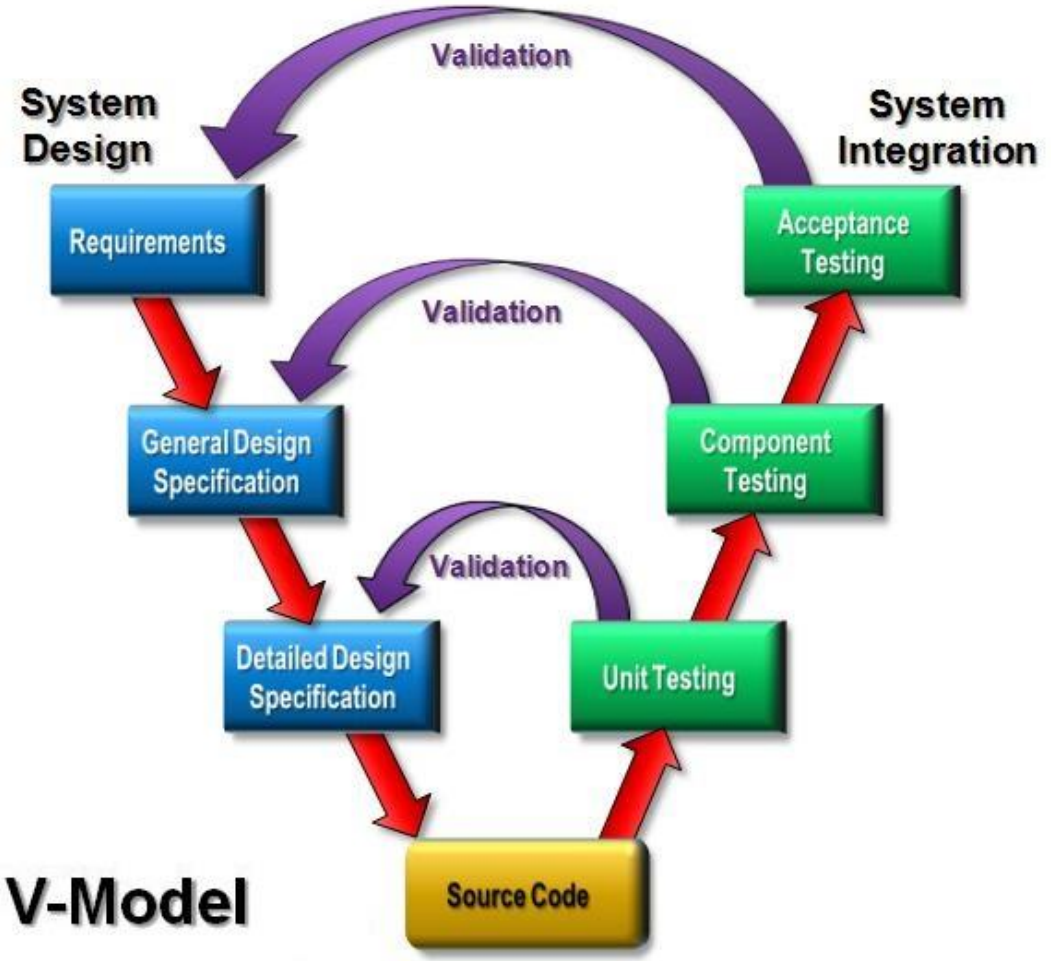
  Geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarılmaktadır.

- Mimari Model

  Sistem tasarımı ve oluşacak altsistem ile tüm sistemin sınama işlemlerine ilişkin işlevler.

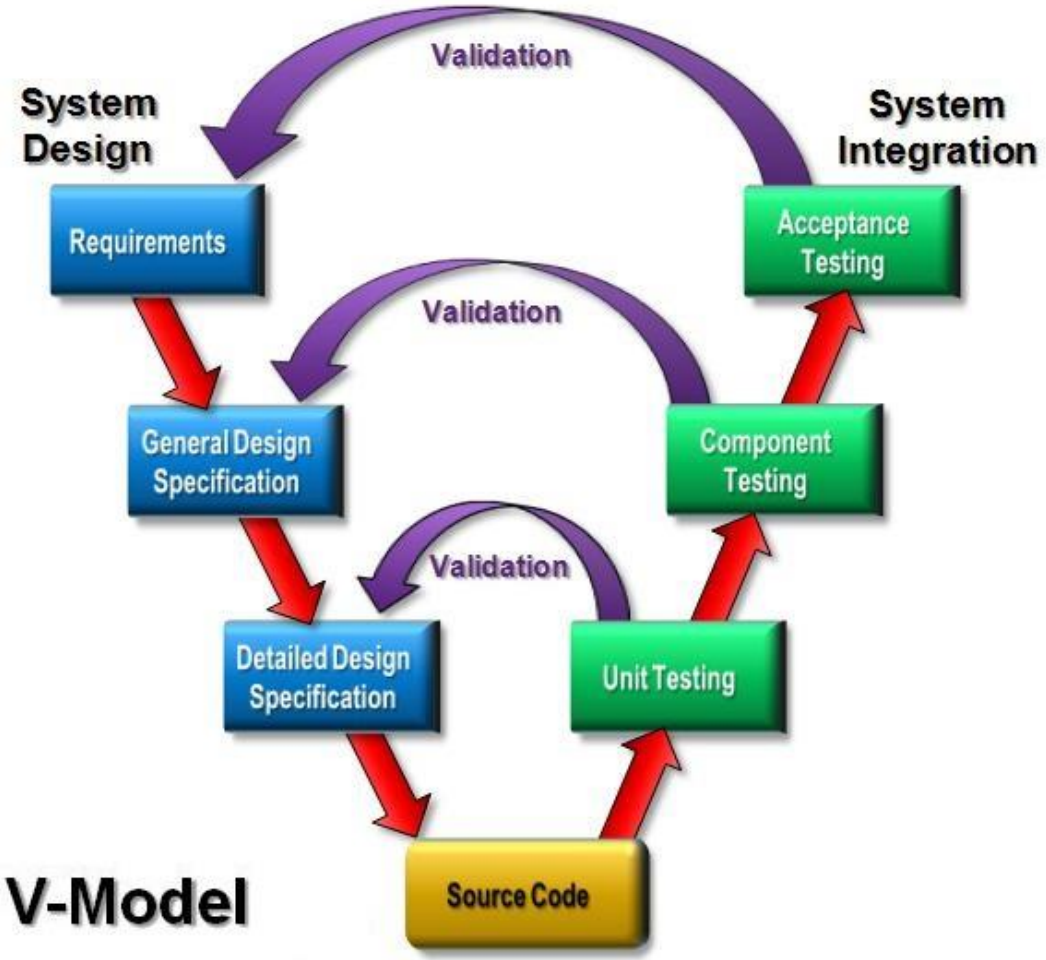
- Gerçekleştirim Modeli

  Yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar.



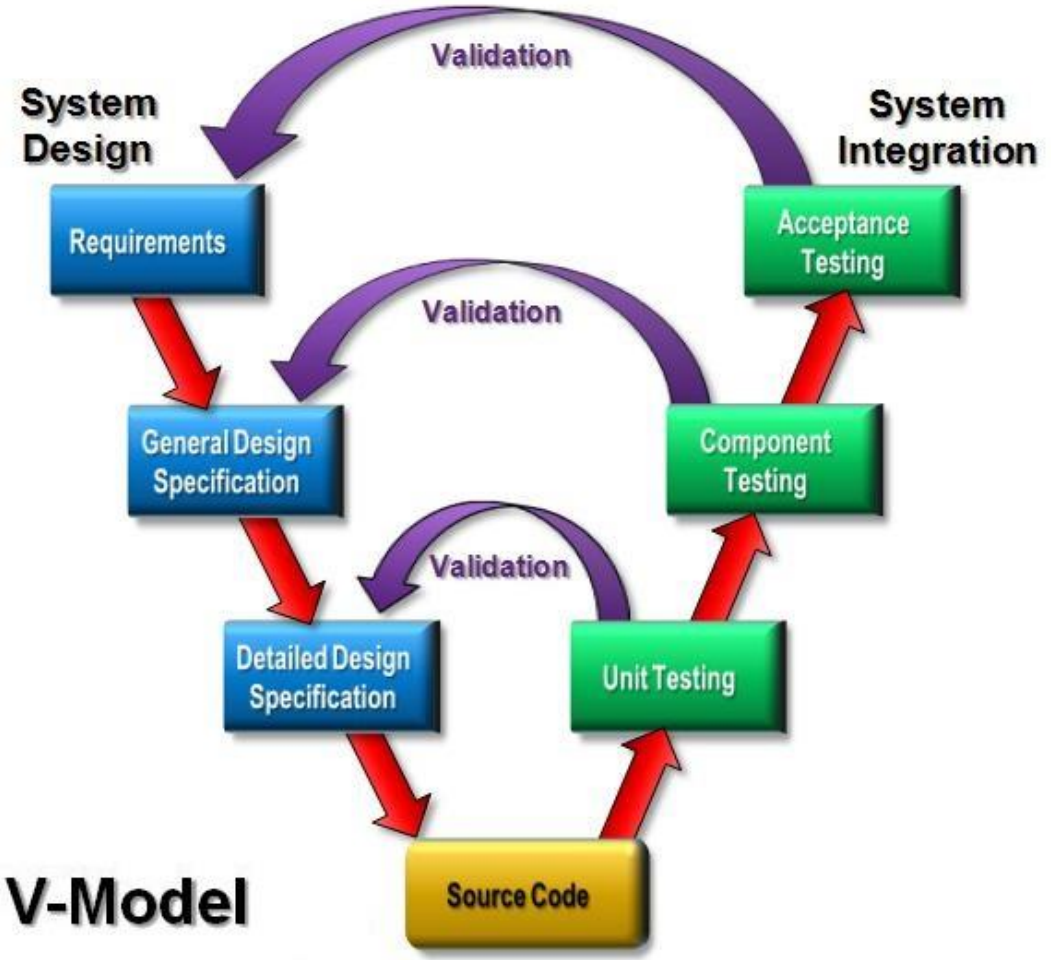
V MODEL

- Üretim modelinde aşağı yönlü, test modelinde yukarı yönlü bir yapı vardır.
- V-Şeklinde model aslında biraz daha alta inip yukarı çıkan ve her aşamasının karşılığının olduğu bir modeldir.
- “Bir şey yapıldı ama bunun karşılığı doğru mudur?” sorgulamasının yapıldığı, en alta kadar inip kodlamaya kadar inip sonra da yukarıya doğru sistemi taşıyan bir yapıya sahiptir.



V MODEL

- Belirsizliklerin az, iş tanımlarının belirgin olduğu BT projeleri için uygun bir modeldir.
- Model, kullanıcının projeye katkısını arttırmaktadır.
- BT projesinin iki aşamalı olarak ihale edilmesi için oldukça uygundur:
  - İlk ihalede kullanıcı modeli hedeflenerek, iş analizi ve kabul sınamalarının tanımları yapılmakta,
  - İkinci ihalede ise ilkinde elde edilmiş olan kullanıcı modeli tasarlanıp, gerçeklenmektedir.



Evrimsel Süreç Modelleri

- Çoğu yazılım projelerinde evrimsel bir süreç işlemektedir. Safhalar tekrarlanarak yazılım ürünü sürekli geliştirilmektedir. Sistemin küçük bir parçası başlangıçta ortaya çıkarılmakta ve sonradan kısa aralıklarla yayınlanan versiyonlarla geliştirilmektedir

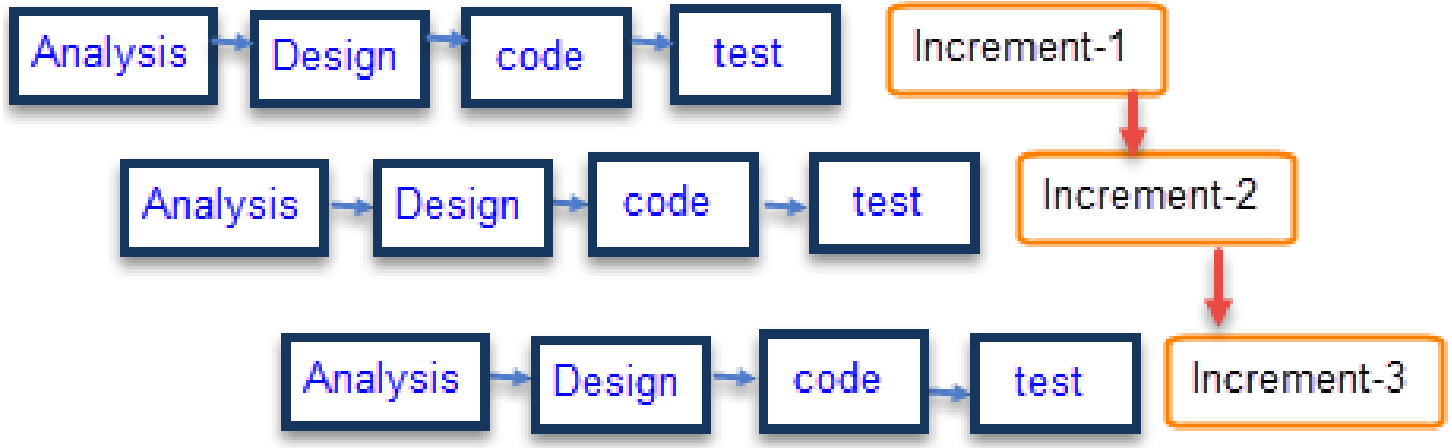
3 problemi vardır:

1. Yöneticiler ilerlemeyi görmek için rapor istemekte, fakat sistem çabuk geliştirildiğinden her versiyonu için doküman hazırlamak maliyetli olabilmektedir.
2. Sürekli değişim yazılım mimarisini bozmakta, değişiklikleri birleştirmek zor ve maliyetli olabilmektedir.
3. Özel araçlar ve teknikler gerektirebilmektedir. Hızlı gelişim için gereken bu araçlar, diğerleriyle uyumlu olmadıkları için bunları kullanacak insan yeteneğine ihtiyaç duyulmaktadır.

- Küçük (<=100.000 kod satırı) ve orta sistemler(<= 500.000 kod satırı) için önerilmektedir (Sommervile, 2000).
- Bu yaklaşım ile ilgili farklı 3 model öne çıkmaktadır. Bunlar Artımsal Model, Spiral Model ve RUP'tur.

Evrimsel Süreç Modelleri: Artımsal (Incremantal) Model

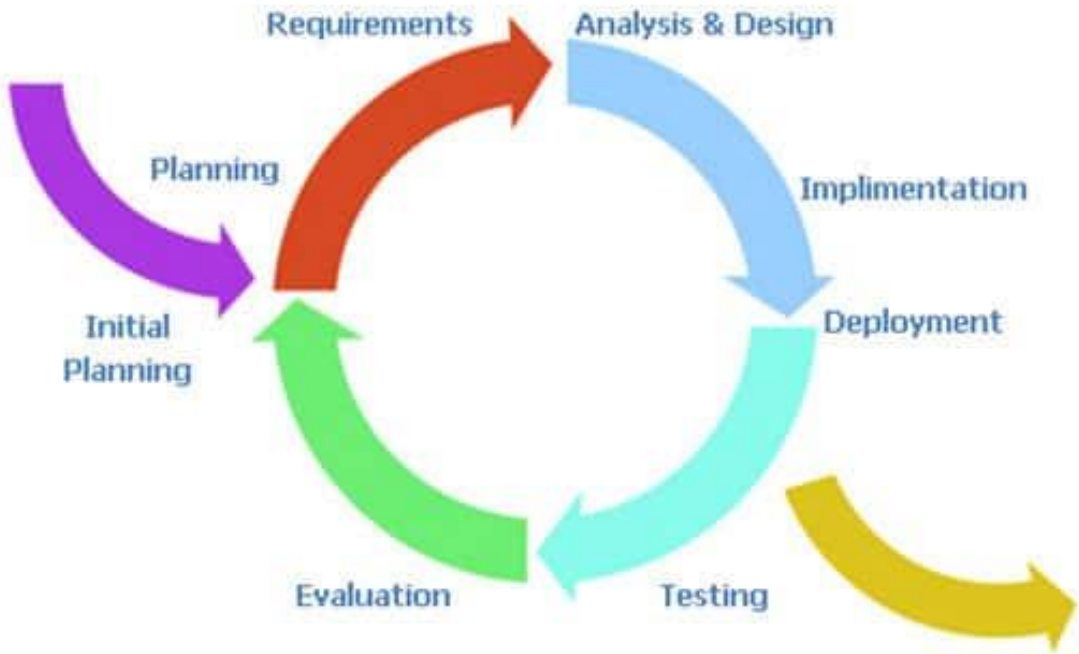
- Artımsal modelde, tek teslimatta tüm sistemi teslim etmektense, sistemi fonksiyonel birimlere ayırtıp, teslimatları artımsal fonksiyonel birimler halinde yapmak tercih edilir.



Incremental Model

Evrimsel Süreç Modelleri: Artımsal (Incremantal) Model

- Modelde bir döngü söz konusudur.
- Planlama aşaması ile başlar.
- Planlamalardan geçip bu planlamanın ardından bir döngüye girilir. İstek analizleri, analizler, tasarımlar yapılır ve uygulamaya geçilir. Uygulamadan sonra iki durum söz konusudur. Birincisi canlıya geçiş durumudur.
- Deployment edilir ve tabi problemler her aşamada olduğu gibi bu aşamada da yaşanır. Ama bu problemler testlerin birer parçası olarak görülebilir.
- Bir yandan da testlerimiz devam eder. Bu problemler sistemi besler. Testlerden ve canlıdan gelen bilgilere göre tekrar değerlendirilmeye tabii tutulur.
- Sonra tekrar istek analiz, analizler, canlıya bir bilgi verilmesi ve tekrar testler gibi süren bir döngü vardır. Bu sırada planlamada değişiklikler olabilir



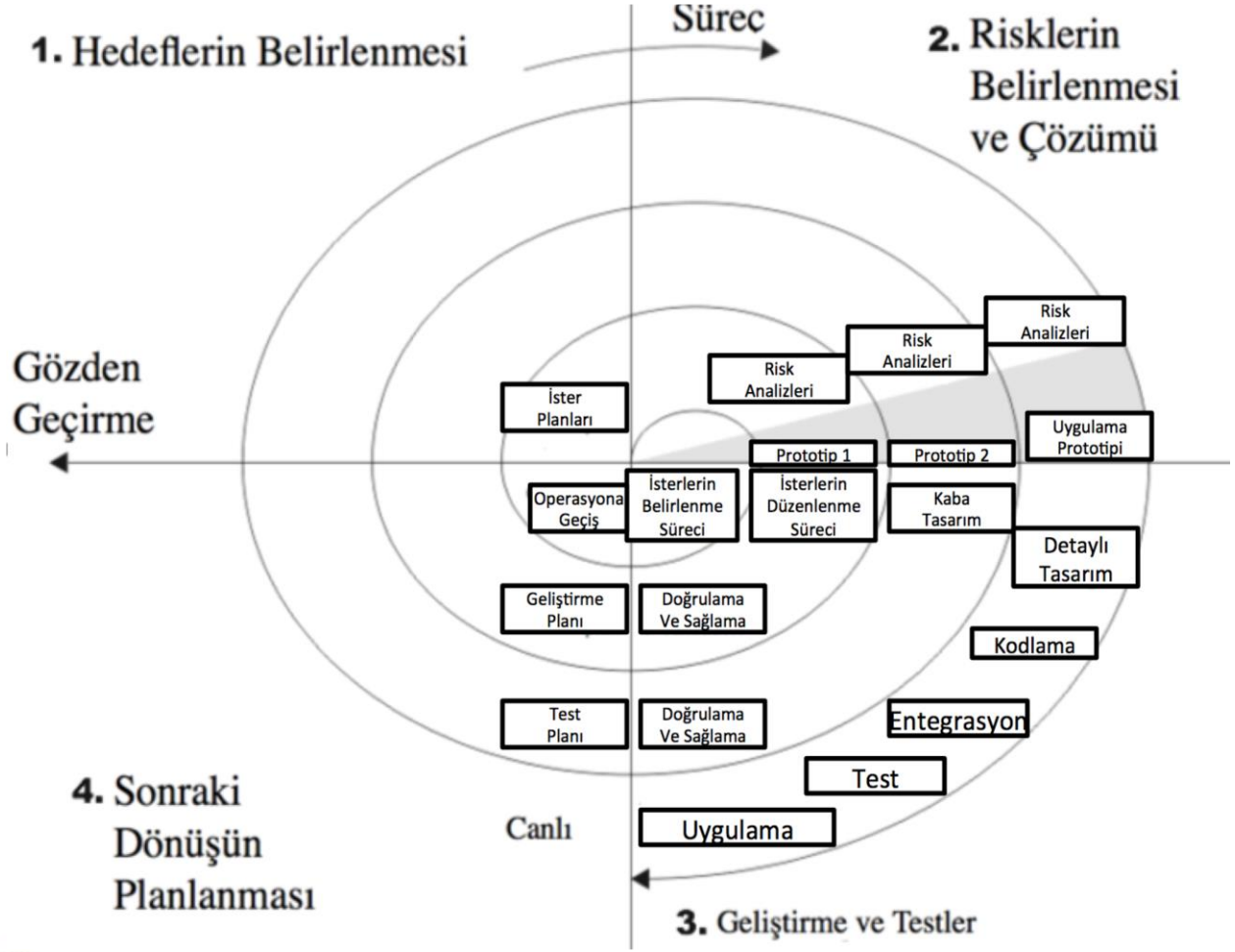
Evrimsel Süreç Modelleri: Artımsal (Incremantal) Model

- Müşteri, değer almak için tüm sistemin bitmesini beklemez. Kritik ihtiyaçlarını karşılayan ilk geliştirme ile yazılım kullanılmaya başlanır.
- Müşteri, prototipleri kullanarak deneyim kazanır ve gelişimin devamı için ihtiyaçlarını bildirir. Müşteri, süreç içerisinde daha fazla yer alır.
- Daha az risklidir. Başarısızlıkların tüm projeye yansıması engellenmektedir.
- Öncelikli servisler ilk verilir ve diğerleri daha sonra entegre edilir.

Müşterinin sistemin önemli parçalarında hata ile karşılaşma olasılığı çok düşüktür. Çünkü önemli parçalar en çok teste tabi tutulan bölümlerdir (Sommervile, 2000).

Evrimsel Süreç Modelleri: Spiral Model Model

- Spiral model, diğer modellerden farklı olarak süreci oluşturan aşamalardan tekrar tekrar geçilmesini ve her geçişte projenin ilerleme kat etmesini hedeflemektedir.
- Bu spirali de 4'e böldüğümüzde 4 aşaması vardır.
- 1. Hedeflerin belirlenmesi
- 2. Risklerin belirlenmesi ve çözümü
- 3. Geliştirme ve testler
- 4. Sonraki dönüşün planlanması



Evrimsel Süreç Modelleri: Spiral Model Model

1. Planlama

   Üretilecek ara ürün için planlama, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirme

2. Risk Analizi

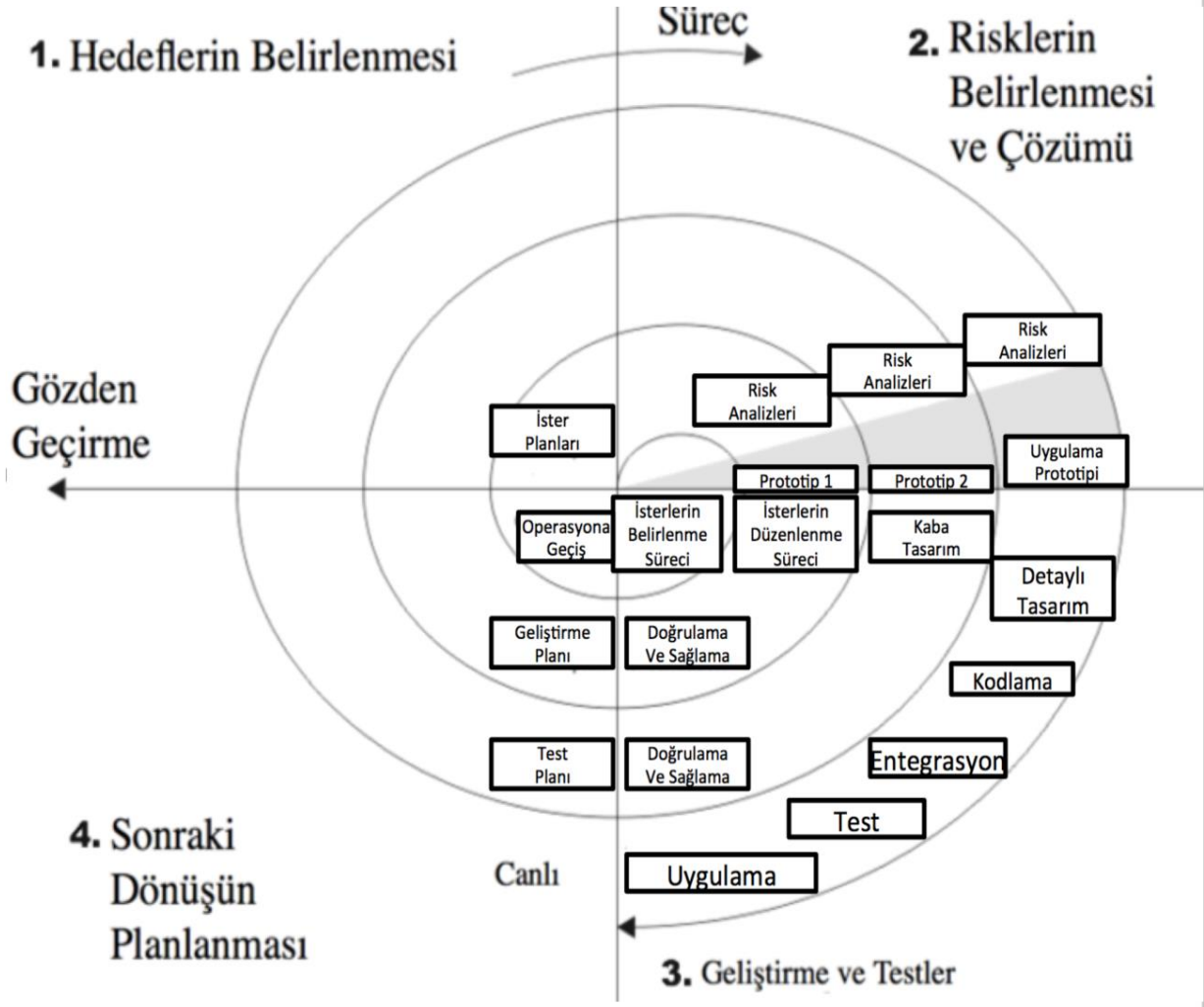
   Risk seçeneklerinin araştırılması ve risklerin belirlenmesi

3. Üretim

   Ara ürünün üretilmesi

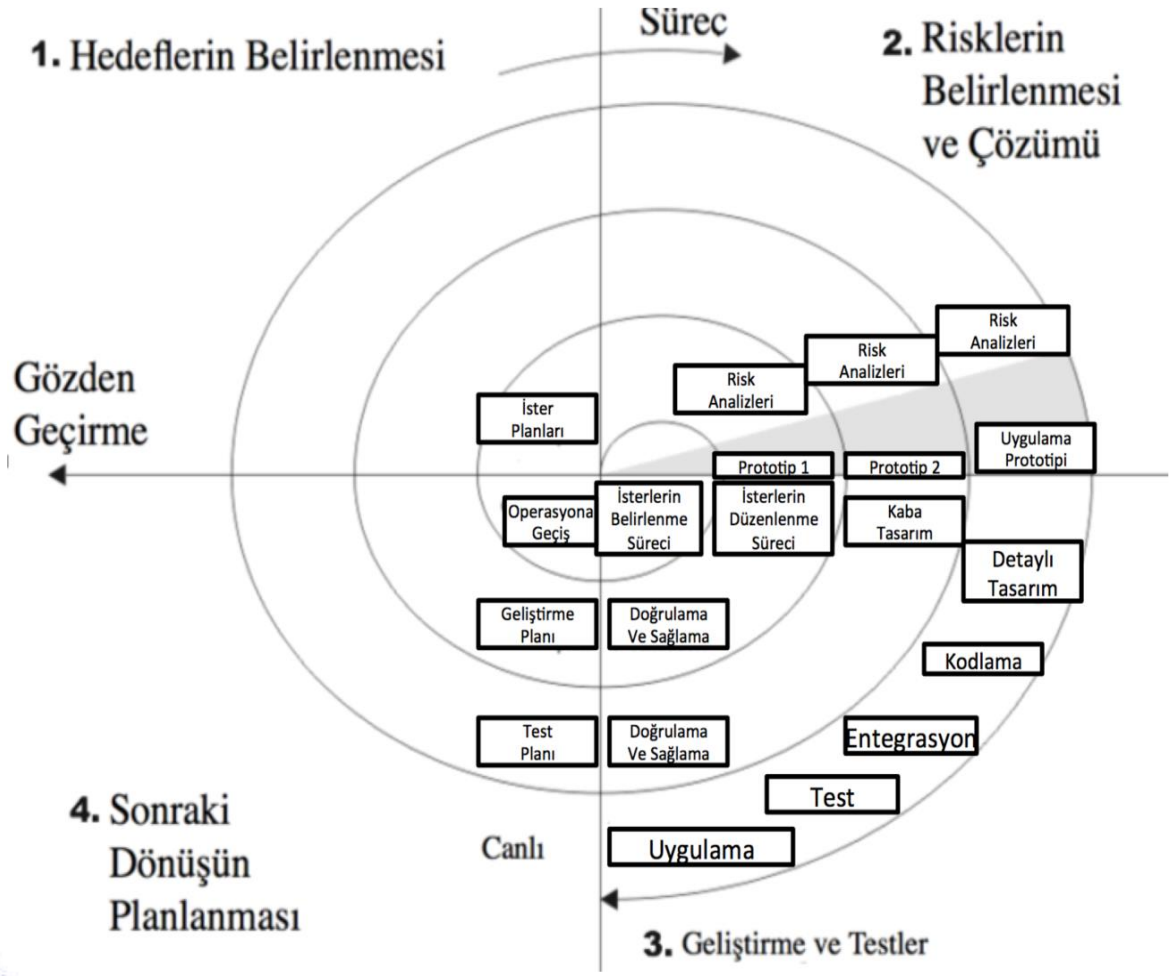
4. Kullanıcı Değerlendirmesi

   Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmeler



Evrimsel Süreç Modelleri: Spiral Model Model

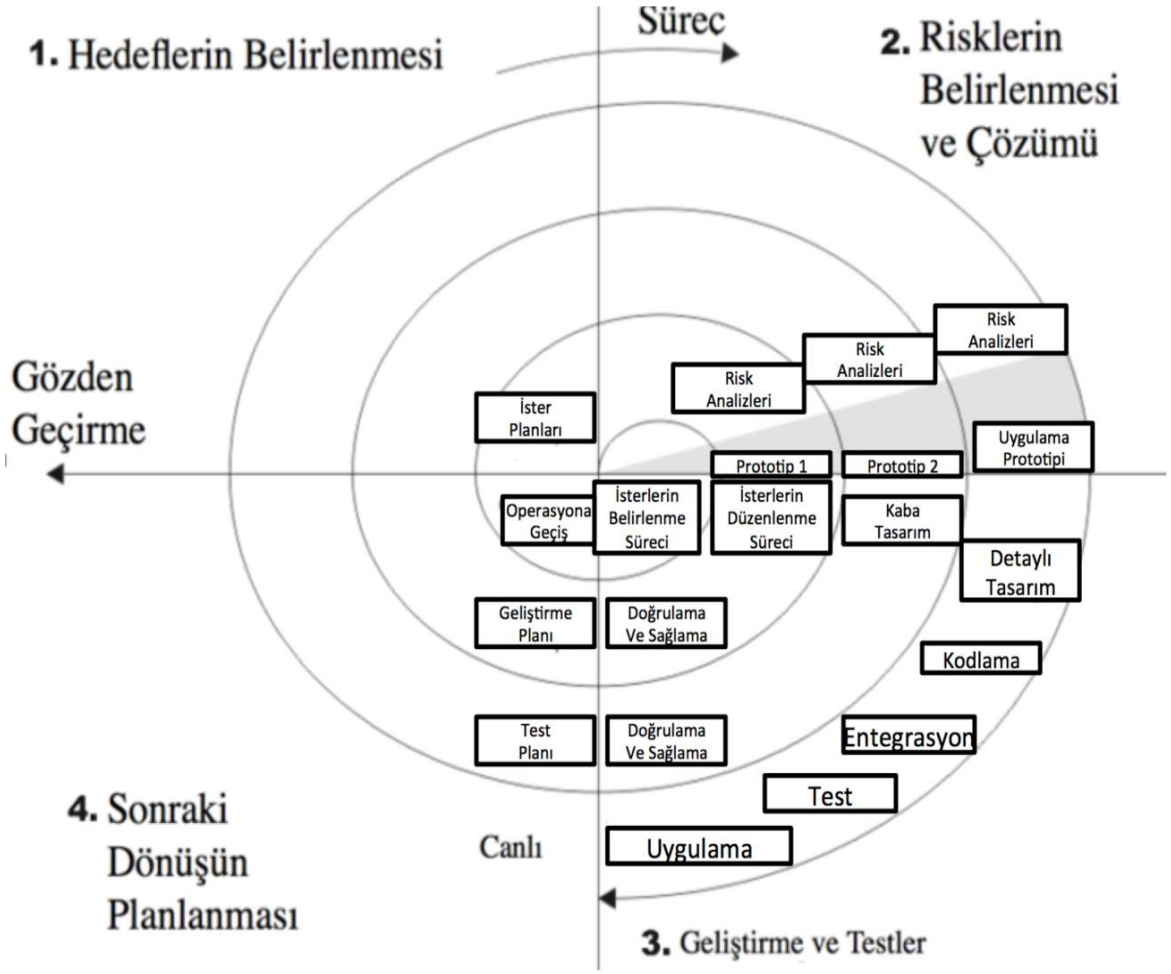
- Modeli oluşturan her döngü (sistemin olabilirliği, ihtiyaç tanımlama, tasarım,...)
  - amaçların, sınırların tanımlanması,
  - risklerin belirlenmesi,
  - gelişim ve geçerliliğin sağlanması,
  - bir sonraki seviyenin planlanması, aşamalarına tabi tutulur.
  - **Risk Analizi Olgusu ön plana çıkmıştır.**
  - Her döngü bir fazı ifade eder. Doğrudan tanımlama, tasarım,... vs gibi bir faz yoktur.
  - Yinelemeli artımsal bir yaklaşım vardır.
  - Prototip yaklaşımı vardır.
- Odaklandığı nokta, bir sonraki riskleri azaltmaktır. Risk yönetimi kullanılması, farkını oluşturur (Sommervile, 2000).



Evrimsel Süreç Modelleri: Spiral Model Model

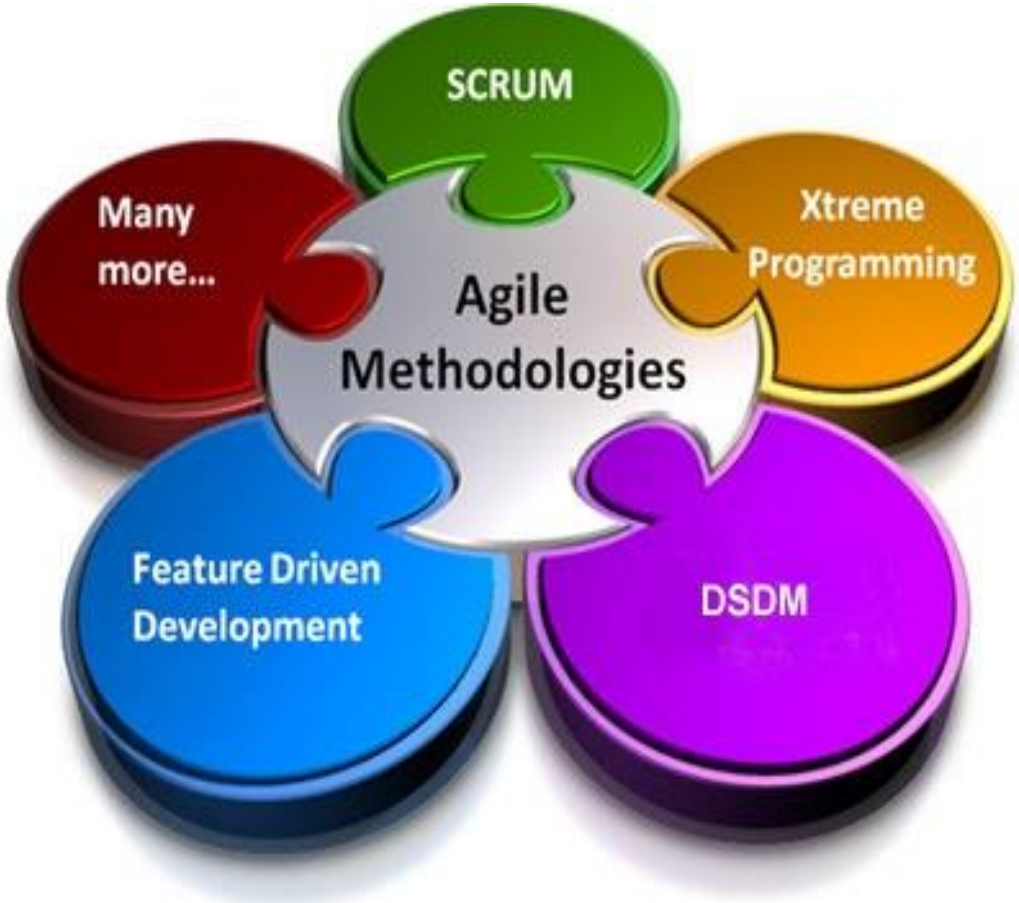
1. Kullanıcı Katkısı
   Üretim süreci boyunca ara ürün üretme ve üretilen ara ürünün kullanıcı tarafından sınanması temeline dayanır.

   Yazılımı kullanacak personelin sürece erken katılması ileride oluşabilecek istenmeyen durumları engeller.
2. Yönetici Bakışı
   Gerek proje sahibi, gerekse yüklenici tarafındaki yöneticiler, çalışan yazılımlarla proje boyunca karşılaştıkları için daha kolay izleme ve hak ediş planlaması yapılır.
3. Yazılım Geliştirici (Mühendis) Bakışı
   Yazılımın kodlanması ve sınanması daha erken başlar.

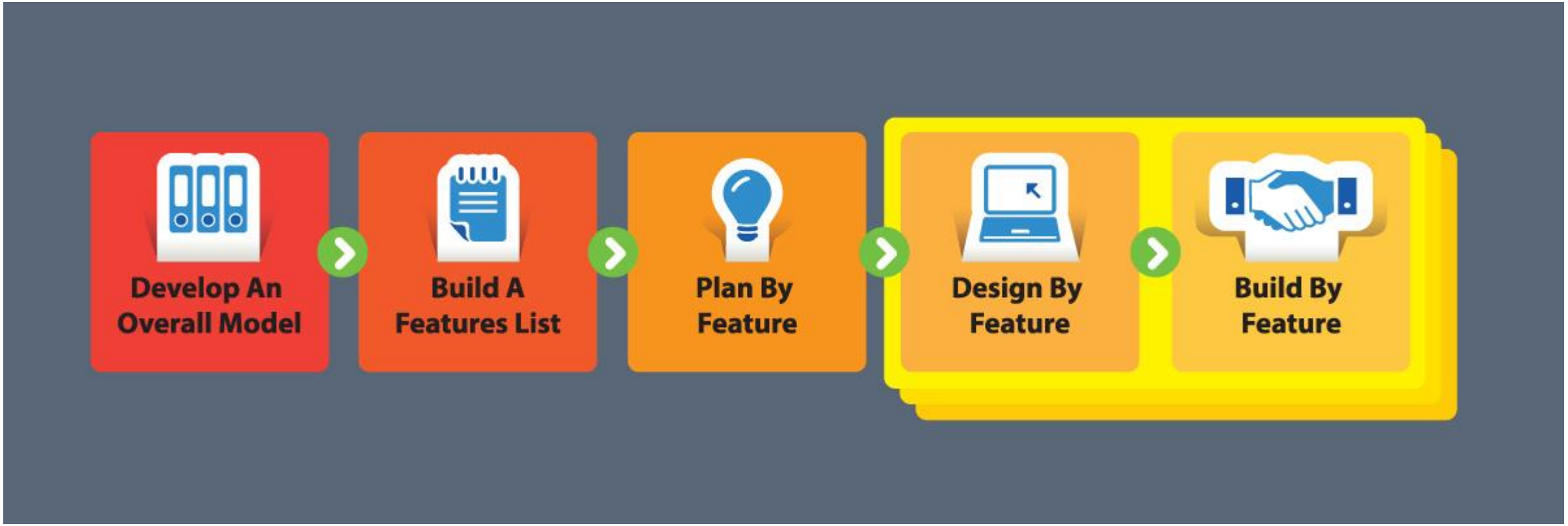


Çevik Metotlar

- Yazılım süreç modellerinde günümüzdeki tercih edilen durum daha adapte edilebilir ve daha esnek çalışma yollarıdır. Kesin tanımlanmış büyük organizasyonel proseslerden taslağı çizilebilir, esnek, çevik proseslere hareket başlamıştır (Kettunen ve Laanti, 2005).
- Çevik, projelerde insan odaklı bir yaklaşımı içerir. İnsanların değişime etkin bir şekilde karşılık vermesini sağlar. Bu da bu projeden fayda sağlayacak olanların ihtiyaçlarını karşılayabilen çalışma ortamlarının yaratılmasına imkan verir.
- Bu kategoride süreçler üst düzeyde tanımlanır. Genellikle sinerjik çözümler veya felsefeler olarak sunulur.
- Özelliğe Dayalı Geliştirme Modeli (Feature Driven Development) (Palmer and Felsing 2002) ve XP Metodolojisi bu kategorideki süreçlere birer örnektir (Karadağ, 2002).



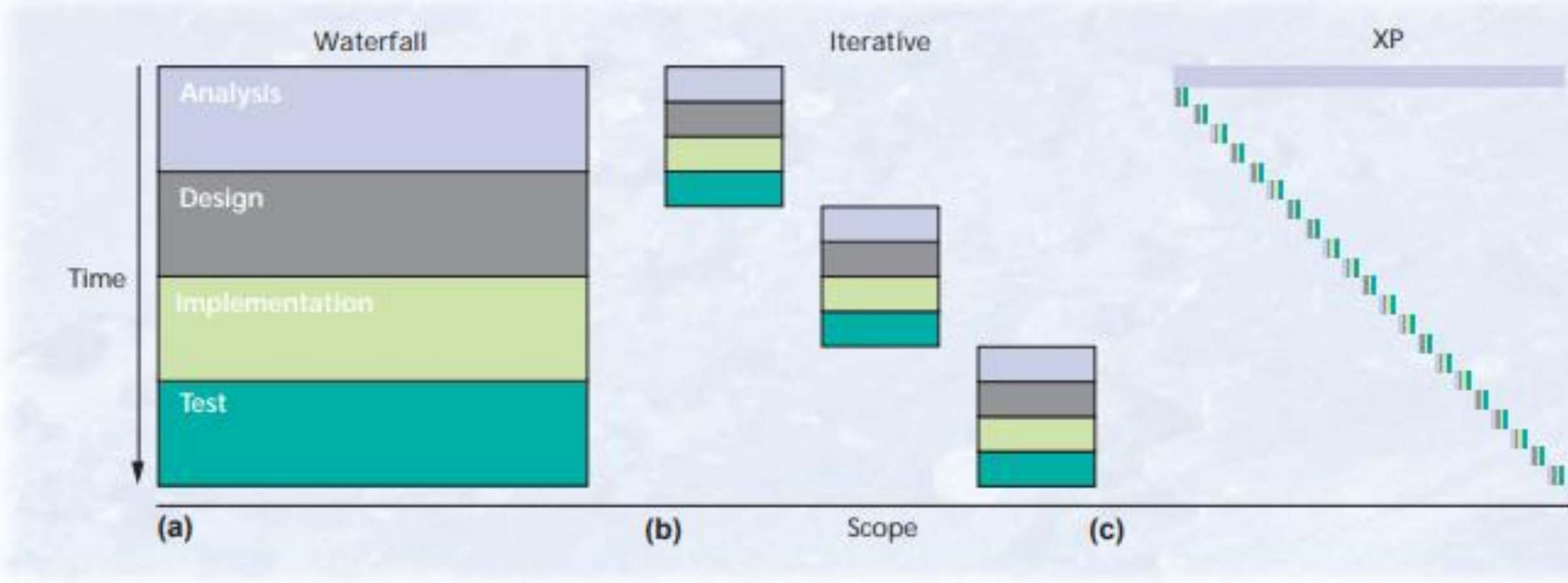
Çevik Metotlar: Özelliğe Dayalı Geliştirme Modeli (FDD - Feature Driven Development)



Çevik Metotlar: Özelliğe Dayalı Geliştirme Modeli (FDD - Feature Driven Development)

- FDD, modelce yönlendirilen kısa tekrarlamalı bir yazılım geliştirme sürecidir. FDD, 5 süreci içerir:
- • Bütünüyle bir modeli geliştirme
- • Özellik listesini oluşturma
- • Özelliklerin planı
- • Özellikleri tasarlama
- • Özellikleri gerçekleştirme
- FDD işlevlerinin basitliğinin bir yararlı özelliği, yeni elemanların kolaylıkla ekibe katılabilmesidir. Bu yöntem, sık ve dikkate değer sonuçlar üretir. İlave olarak, FDD planlama stratejilerini içerir ve hatasız bir ilerleme izleme sağlar (Karadağ, 2002).

Çevik Metotlar- XP METODU (BECK 1999)



Çevik Metotlar- XP METODU (BECK 1999)

- XP Metodoloji; “Hafif” metodolojilerden en iyi bilinendir.
- XP, küçük, aynı yerde yerleşik takımlarda çok iyi çalışır. Küçük, birlikte çalışan proje takımları (4-10 kişilik) için uygundur (Kettunen ve Laanti, 2005).
- XP’nin temel uygulaması, tekrarlama içeren hızlı uygulama geliştirme (Rapid Application Development) metotlara benzer.
- Bu iki haftalık dilimleri, sık güncellemeleri, teknik özellikleri ve işin bölünmesini içerir.
- **Dolayısı ile, müşteri ile güçlü bir ilişki önemli bir gerektir.**

Çevik Metotlar- XP METODU (BECK 1999)

- XP metodolojinin 4 anahtarı vardır:
  - İletişim
  - Geri Bildirim
  - Basitleştirme
  - Cesaret

XP programcıları müşterileri ve takım üyeleri ile devamlı iletişim halindedir.
Tasarımlarını basit tutarlar.
XP ile diğer metodolojiler arasında gözlenen bir diğer fark onun testlerdeki gücüdür.
XP, küçük geliştirme ekiplerinin, çabuk ürün verme ve değişimleri için tasarlanmıştır.

YAZILIM GELİŞTİRME SÜREÇLERİ- ÖZET

- Kompleks projelerde ise tek bir model seçmek en iyi yol değildir. Hibrid modeller, farklı modellerin özelliklerini dengelemek daha iyi bir seçenektir. Ürünün farklı parçalarının değişen karakteristiklerine bağlı olarak farklı modeller kullanılabilmektedir.

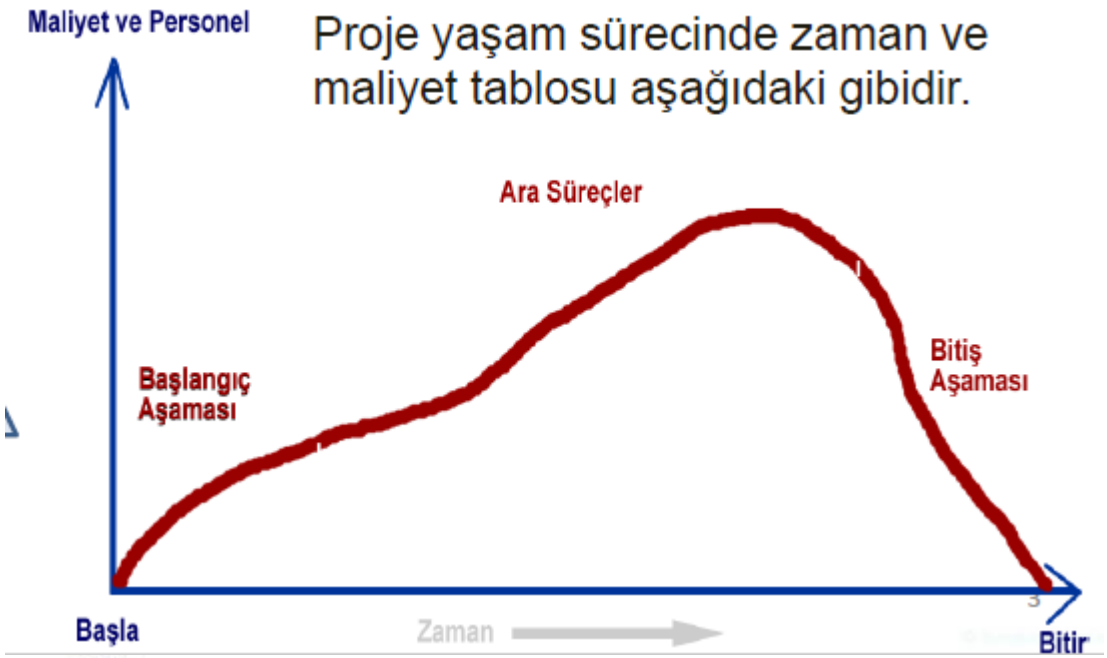
KAYNAKLAR

•[1] Tian , J. (2005), Software Quality Engineering Testing, Quality Assurance, and Quantifiable Improvement, IEEE Computer Society/ John Wiley & Sons, Inc.
•[2] S.E. Seker, (2015), Yazılım Geliştirme Modelleri ve Sistem Yazılım Yaşam Döngüsü, YBS Ansiklopedi, 2,3,18.
•[3] Zuhal Gül, (2006), Yazılım Geliştirme Sürecinin İyileştirilmesi Ve Türkiye Uygulamaları, YL Tezi, İstanbul Teknik Üniversitesi

YAZILIM KALİTE VE KONFİGÜRASYON YÖNETİMİ

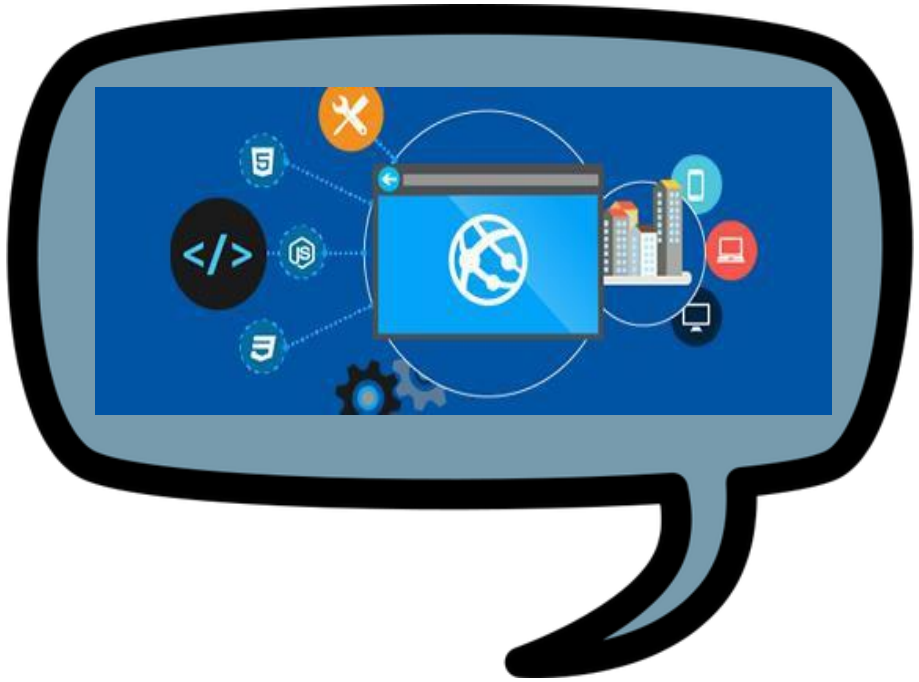
Uzun yıllardır proje yönetimi ile ilgili farklı yöntemler geliştirilmektedir. Özellikle 1990 yılından sonra yazılım proje yönetiminin önemi daha çok anlaşılmıştır. Yazılım Projelerinin 3 ana unsuru vardır.

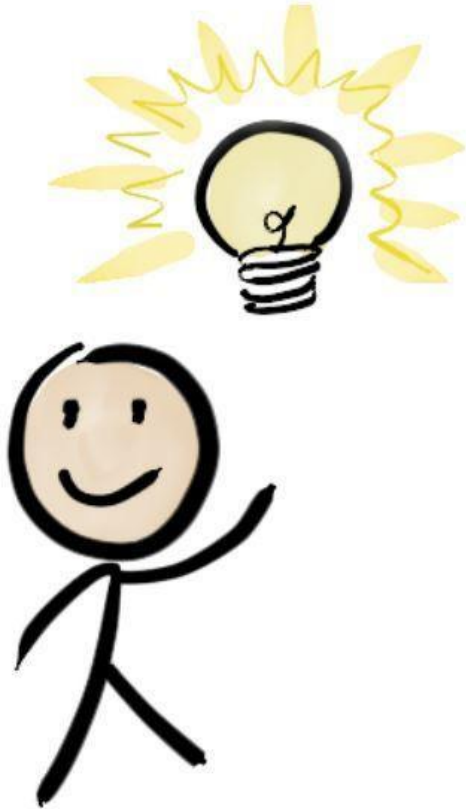
1. KALİTE
2. MALİYET
3. ZAMAN



1.Günümüzde Yazılım Projelerinin Durumu

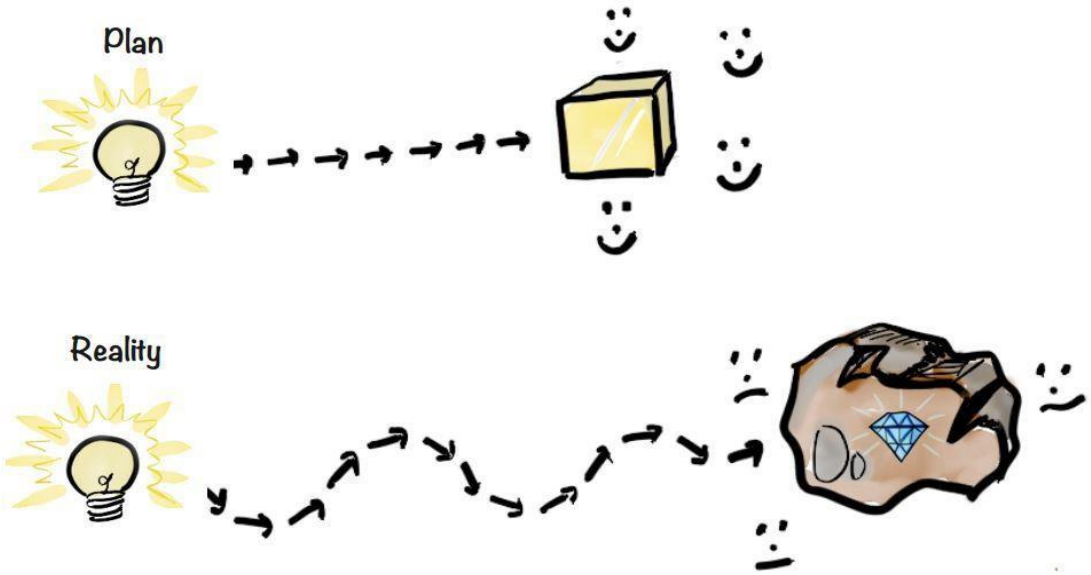
- Birçok proje harika bir fikir ile başlar!





1.Günümüzde Yazılım Projelerinin Durumu

- Bu projelerin büyük bir kısmının başarısız olması muhtemeldir!



1.Günümüzde Yazılım Projelerinin Durumu

- Gartner Institute'un BT sektörü araştırmasına göre: BT projelerinin

  %74'ü başarısız ya da maliyet/zaman hedeflerini aşıyor. BT projelerinin %51'i bütçesini %200 oranında aşıyor ve hedeflenen özelliklerin %75'ini karşılayabiliyor.

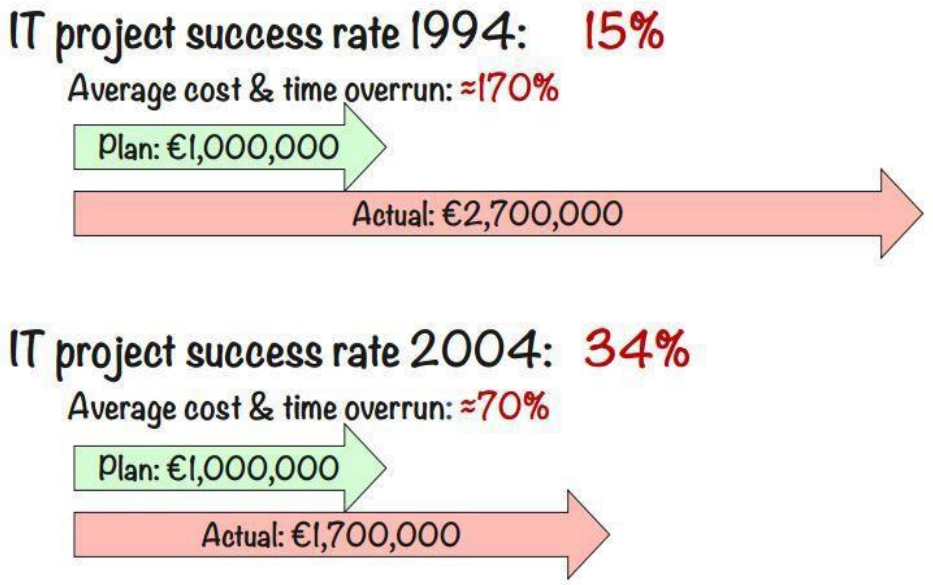
  Standish grubun 2000 yılında gerçekleştirdiği bir araştırmaya (Chaos in the new Millenium 2000) göre yazılım projelerinin başarıya ulaşma oranı %28 olarak veriliyor. Diğerleri ya başarısız (%23) ya da zorlanmış (%49) projelerdir. Aynı araştırma yazılım projeleri özelinde de proje maliyetlerinin tahmin edilenin üzerinde olduğu veya zaman aşımı olduğu ya da niteliklerin istenilene tam uygun olmadığını gösteriyor.

- Gartner Group'un (Technowledge SM 99 Presentation) yapmış olduğu bir araştırmaya göre BT projelerinin %70'i beklenen faydayı sağlamıyor.

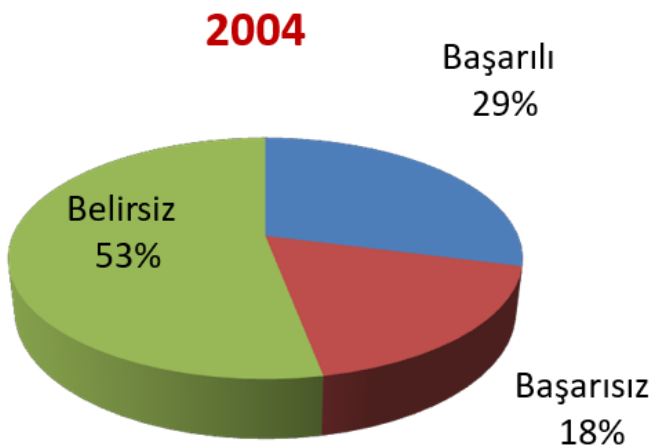
  Gartner Institute'un 2001 BT sektörü araştırmasına göre: Amerika'da her yıl başarısız BT projeleri için 75 milyar dolar harcanıyor .

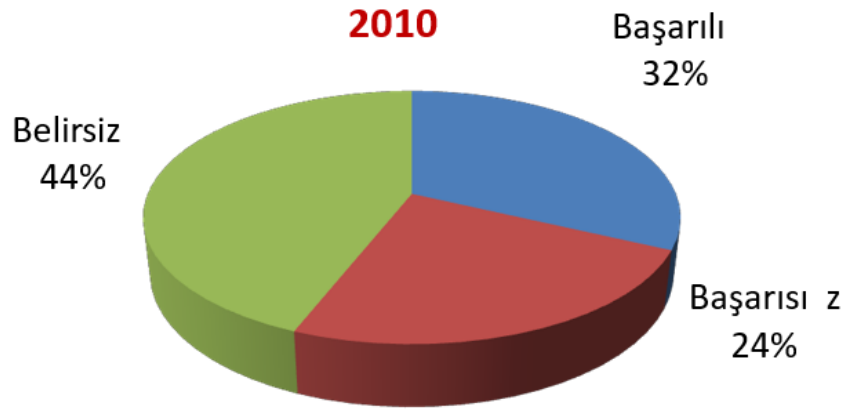
1.Günümüzde Yazılım Projelerinin Durumu

- Birçok Bilgi Teknolojisi projesi başarısız olmuş veya gecikmiştir. The Standish Group, 10 yıl içerisinde 40.000'den fazla proje üzerinde çalışmıştır.



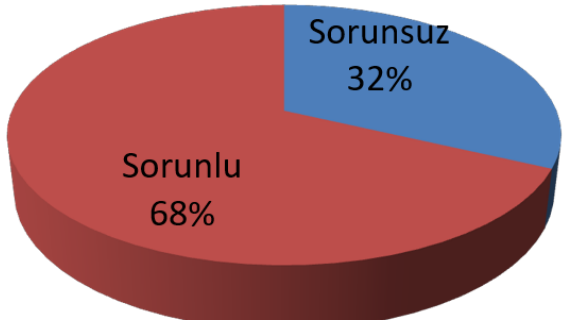
1.Günümüzde Yazılım Projelerinin Durumu





Tahmini Yıllık Zarar : 55 Milyar $

Proje Yönetimi açısından;



1.Günümüzde Yazılım Projelerinin Durumu

Ülkemizde durum nasıl?

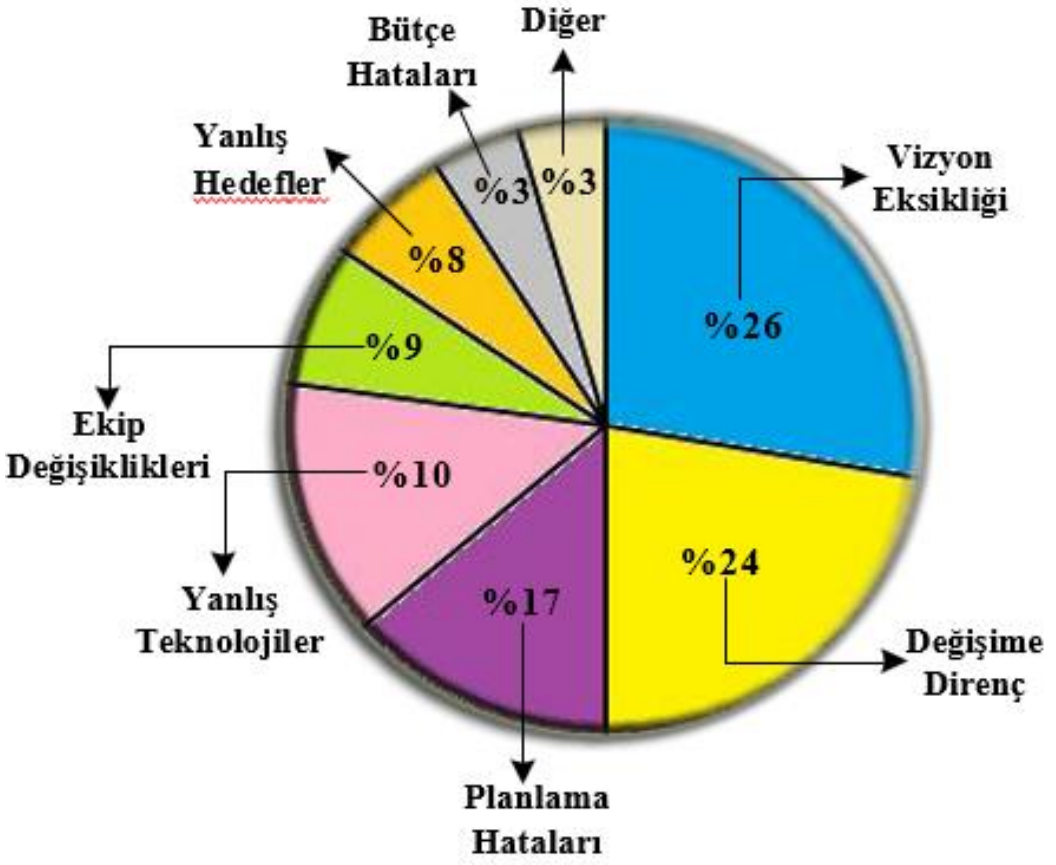
Durum	Oran
Tam başarılı	%4-5
Kısmen başarılı	%45-50
Çöpe gidenler	%50

Agile Turkey

Örneğin, ülkemizde geliştirilen projelerin başarıya ulaşmasına katkı sağlamak amacıyla, Araştırma Destek Programları Başkanlığı (ARDEB) tarafından desteklenen projelerin çıktı, sonuç ve etkilerini nicelik ve nitelik olarak artırmak amacıyla yüksek başarı ile sonuçlanan projelerin yürütücü ve araştırmacılarını ödüllendirmek için TÜBİTAK tarafından belirlenen ölçütler ve değerlendirme yöntemine göre hesaplanarak, proje ekibine (yürütücü ve araştırmacılara) TÜBİTAK Proje Performans Ödülü (PPÖ), denilen bir teşvik ödülü verilmektedir.

1.Günümüzde Yazılım Projelerinin Durumu

- Başarısızlığın ana sebepleri:
  - Müşterinin isteklerini doğru analiz edememek;
  - Proje için uygun ekibi kuramamak;
  - Yanlış teknoloji ve mimari seçimleri;
  - Geleneksel yöntemlerin eksiklikleri;
  - Müşteriyle iletişimden kaçınmak vs.



ÇEVİK YAZILIM YÖNTEMİ

Çevik yöntemler(ÇY)'lerin benimsediği ve Çevik Manifesto ile özetlenen ilkeler ve değerler bütünü, yazılım gelistirme dünyası müşteri beklentilerine cevap vermek için çare olarak son yıllarda artan oranda kullanılmaya baslanan yeni bir yaklasım olarak karsımıza çıkmaktadır. Yalın düsünce, müşteriye değer sağlamayan faaliyetleri; zaman, kaynak ve para israfı olarak görür ve sürecin etkinlesmesi için israfın yok edilmesini gerektirir.

Temel ilke olarak ;

- Bireyler ve onlar arasındaki etkilesimi süreç ve araca tercih etmek,
- Çalısan bir yazılımı detaylı bir dokümantasyona tercih etmek,
- Müsteri ile isbirliğini, anlasma görüsmelerine tercih etmek,
- Değisikliklere istenildiği anda cevap verebilmeyi sınırları belirli bir plana tercih

etmek.

- **Kendi kendine organize olan takımlar kurmak ?**

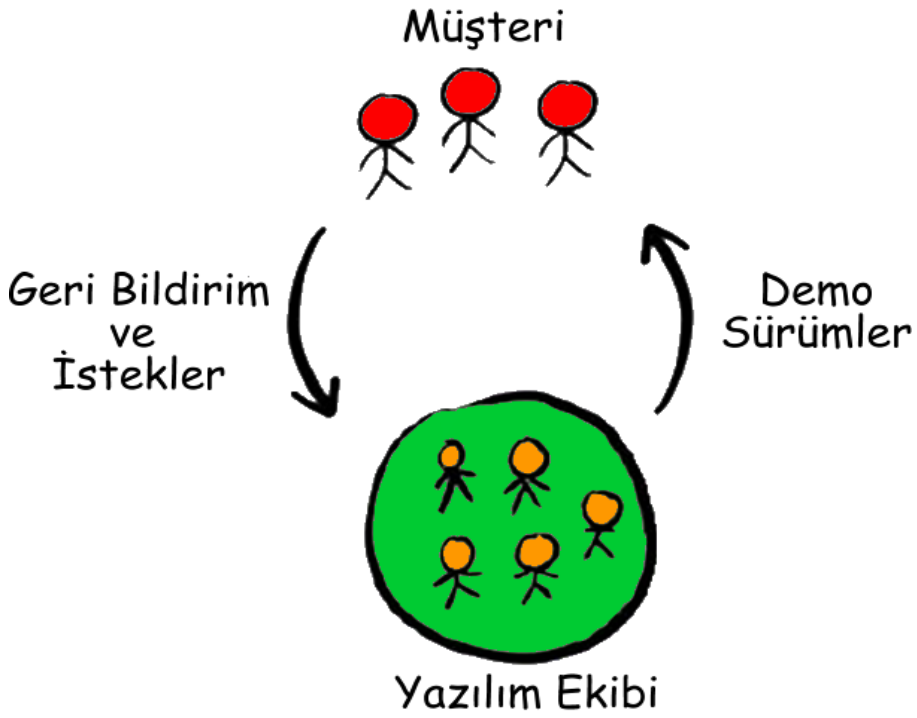
ÇEVİK YAZILIM YÖNTEMİ

- Hızlı, devamlı ve kullanışlı yazılım üreterek müşteri memnuniyeti sağlamayı amaçlar.
- Geliştiriciler ile iş adamları arasında günlük ve yakın işbirliği bulunmalıdır.
- Çalışan yazılım gelişimin en önemli ölçüsüdür.
- Taleplerdeki geç değişikliklerin de memnuniyetle karşılanır.
- Yüz yüze görüşme iletişimin en güzel yoludur.
- Kendi kendini organize eden takım yapısı gereklidir.
- Basitlik önemlidir.

ÇEVİK YAZILIM YÖNTEMİ

- ***Tekrarlamalı*** ve ***artımsal*** bir ürün geliştirme yöntemidir.
- **Bireyler ve etkileşimi**, süreç ve araca tercih eder.
- **Çalışan bir yazılımı**, detaylı ürün belgelendirmeye tercih eder.
- **Müşteri ile işbirliğini**, sözleşmedeki kesin kurallara tercih eder.
- **Değişikliklere uyum sağlayabilmeyi**, belirli bir plana tercih eder.

ÇEVİK YAZILIM YÖNTEMİ



- Çevik yazılım metodu, kısa vadeli planlar ve küçük parçalar halinde yazılımın geliştirilmesini ön görür. Yazılımın geliştirilmesindeki geri dönüş (feedback) ve değişikliklere uyum sağlamak son derece önemlidir. Her yapılan yineleme yazılımı hedeflenen adıma bir adım daha yakınlaştırır. İstenilen sonuca ulaşmak adına birden çok yineleme gereklidir.

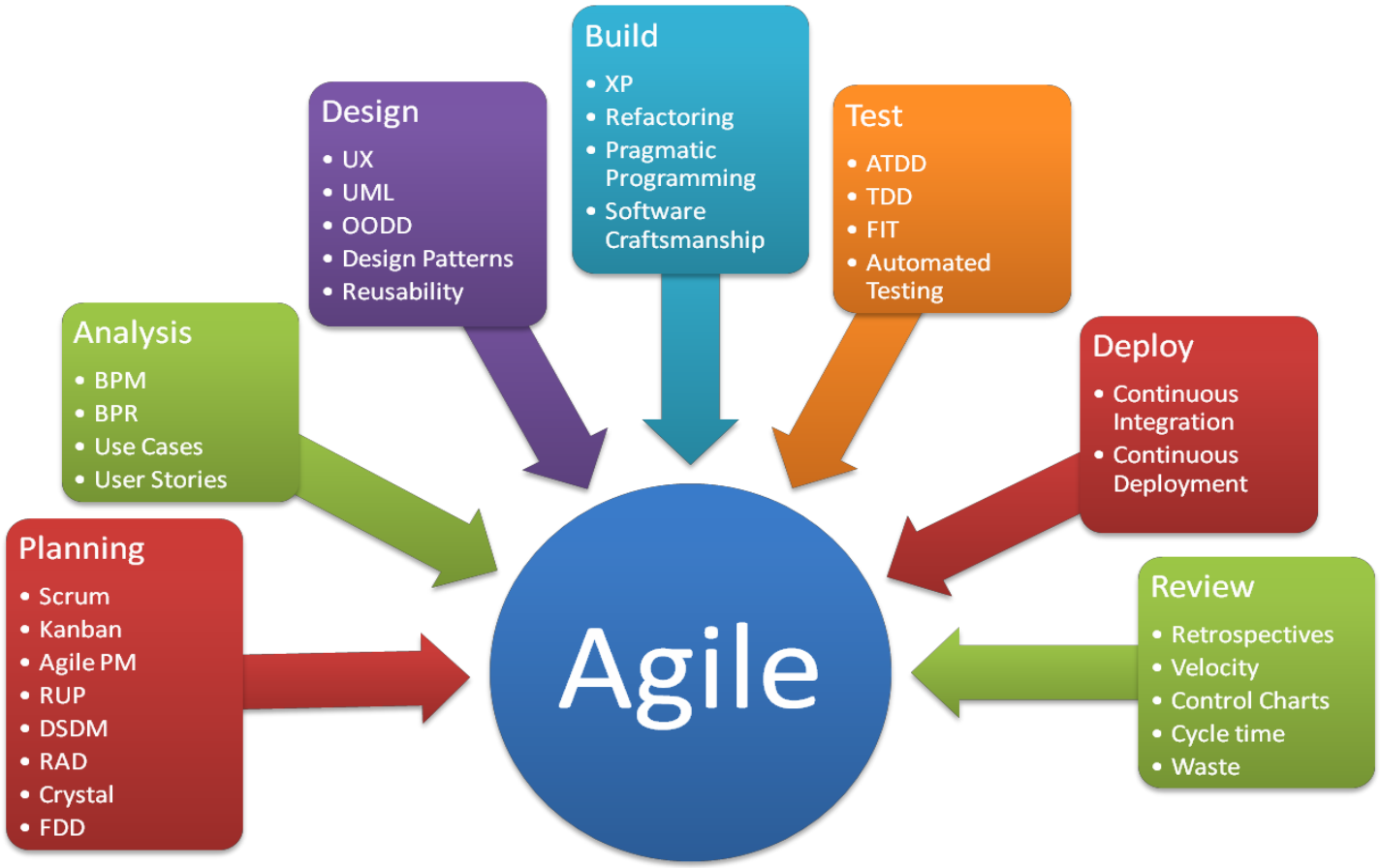
ÇEVİK YAZILIM Süreç Modeli

- Çevik modeller, mevcut geleneksel modellere alternatif olarak, 1990'larda ortaya çıkmaya başlamıştır.
- 1950'lerdeki üretim alanında verimliliğin artırılması için geliştirilen yalın yaklaşımların, yazılım sektöründe bir uzantısı olarak ortaya çıkmıştır.

- Çevik modeller kapsamında; yazılım sistemlerini etkili ve verimli bir şekilde modellemeye ve belgelendirmeye yönelik, pratiğe dayalı yöntemler yer alır.

- Bu modelleme biçiminin; kapsadığı değerler, prensipler ve pratikler sayesinde geleneksel modellemelere metotlarına göre yazılımlara daha esnek ve kullanışlı biçimde uygulanabileceği savunulmaktadır.

ÇEVİK YAZILIM Süreç Modeli



Çevik Yazılım Geliştirme Manifestosu

2001 yılında, dünyanın önde gelen çevik modellerinin temsilcileri, ortak bir zeminde buluşabilmek adına bir araya gelerek "çevik yazılım geliştirme manifestosu" nu yayınlamışlardır. Bu manifestoya göre;

- **1. Hızlı, devamlı ve kullanıslı yazılım gelistirerek müsteri memnuniyeti sağlamak amaçlardandır.** Bu, projenin ilk asamalarından itibaren sürekli kod teslimleri ile yapılır ve müsterinin yazılımı çok önceden kullanmaya baslayarak değer sağlamasına olanak sağlanır. Günümüzde ÇY'lerin yaygınlasmasının baslıca nedenlerden biri, yapılan yatırımların hızlı geri dönüsünün olmasıdır .
- **2. Değisiklik talepleri, projenin ilerleyen asamalarında olsa dahi kabul edilir.** Amaç müsterinin ihtiyaçlarını karsılayan, onlara yarar sağlayacak, gerçek değer katacak yazılımı gelistirmektir ve ihtiyaçlarda meydana gelen değisiklikler projenin sonraki asamalarında dahi yazılıma aksettirilmelidir. Test güdümlü gelistirme, kapsamlı otomatik testler, sürekli entegrasyon, basit tasarım, kodun yeniden düzenlenmesi(refactoring) gibi uygulamalar sayesinde değisikliklerin getireceği maliyetler minimuma indirilir ve süreç değisikliklere daha hızlı uygun hale getirilir.

Çevik Yazılım Geliştirme Manifestosu

- 3. Sık sık çalışan yazılım teslim edilmelidir. Bu aralıklar tipik olarak 1-4 hafta arasıdır. Bu sayede sürekli geri beslenim sağlanır ve müsterinin istekleri doğrultusundayazılım evrimleseresek gelisir.
- 4. Gelistiriciler ile diğer alan uzmanları arasında günlük ve yakın işbirliği bulunmalıdır. Farklı roller arasında duvarlar örülmez. Rol bazlı ekipler yerine yazılım özelliklerine göre ekipler olusturulur. Testçi, analist, yazılım gelistirici aynı ekibin içinde çalısır ve sürekli iletisim halindedir.
- 5. Projeler motive bireyler çevresinde kurulur. Ekip kendi kendine organize olacak yetkiye sahiptir.
- 6. Yüz yüze görüsme iletisimin en güzel yoludur.
- 7. Çalısan yazılım parçaları, gelisimin en önemli ölçüsüdür.

Çevik Yazılım Geliştirme Manifestosu

- 8. Sürdürülebilir bir hızı sağlamaya çalısır. Planlamaların sağlıklı olması için ekibin is teslim hızının güvenilir olması gerekir. Örneğin fazla mesailer gibi yöntemlerle ekibin hızını geçiçi olarak arttırmak tercih edilen yöntemlerden değildir.
- 9. Teknik mükemmelliğe katılım ve iyi tasarım çevikliği gelistirir. Teknik açıdan mükemmel, sade çözümler olusturulmasına özen gösterilir. En iyi tasarım çabuk genisleyebilen tasarımdır.
- 10. Basitlik önemlidir. Sadelik anlayısı akla gelen ilk bastan savma çözümü uygulamak yerine anlasılması ve sonradan değistirilmesi kolay, maliyeti en düsük ve o anki gereksinimleri karsılayan çözümü kullanmaktır.
- 11. En iyi mimariler, ihtiyaçlar ve tasarım kendi kendine organize ekiplerce ortaya çıkarılır.
- 12. Ekip kısa sürelerle toplanır, çalısma yöntemlerini gözden geçirir. Daha etkin ve etkili çalısmak için geçmisi kapsayan (retrospective) biçimde yapılan toplantılarla gözden geçirir.

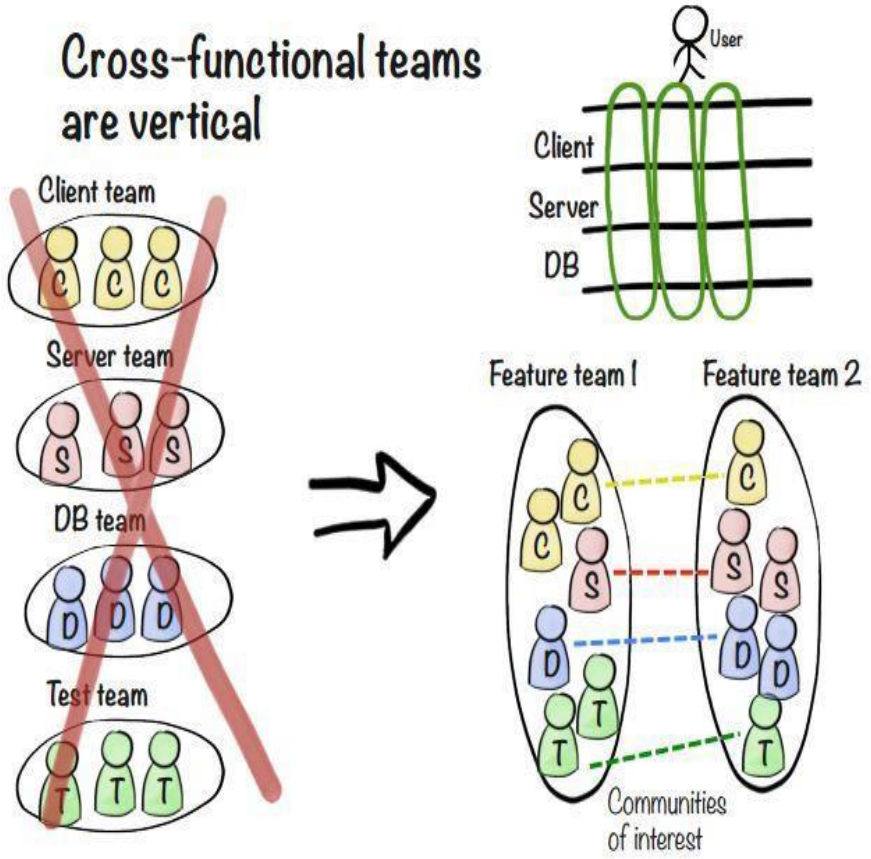
Çevik Yöntemlerde amaç, operasyonel maliyetleri ve döngü zamanını kısaltıp, kaliteyi arttırmaktır.

Hangi Durumlarda Kullanılabilir?

Bu metotların kullanılmasının en uygun olduğu durumlar şunlardır:

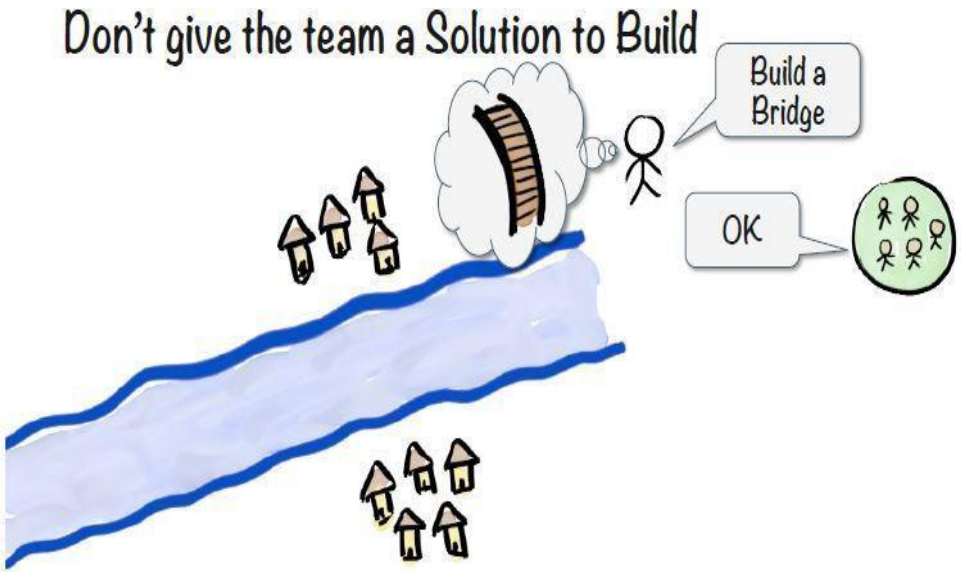
- Projenin yazılım evresinde müşteriden gelebilecek talep değişikliklerinin tahmin edilemez olması.
- Projenin parçalarının önce tasarlanıp ardından hemen geliştirilmesinin gerekmesi ve önceden ne yapılacağını, detaylı yol haritasını ve tasarımını tahmin etmenin çok güç olması.
- Analiz, tasarım ve test etme süreçlerinin ne kadar zaman alacağının önceden bilinememesi.
- Yazılım ekibinin birlikte çalışmak, hiyerarşiye önem vermemek, sağlam iletişim kurmak gibi özelliklere sahip olması.

Çevik MODEL Takımları



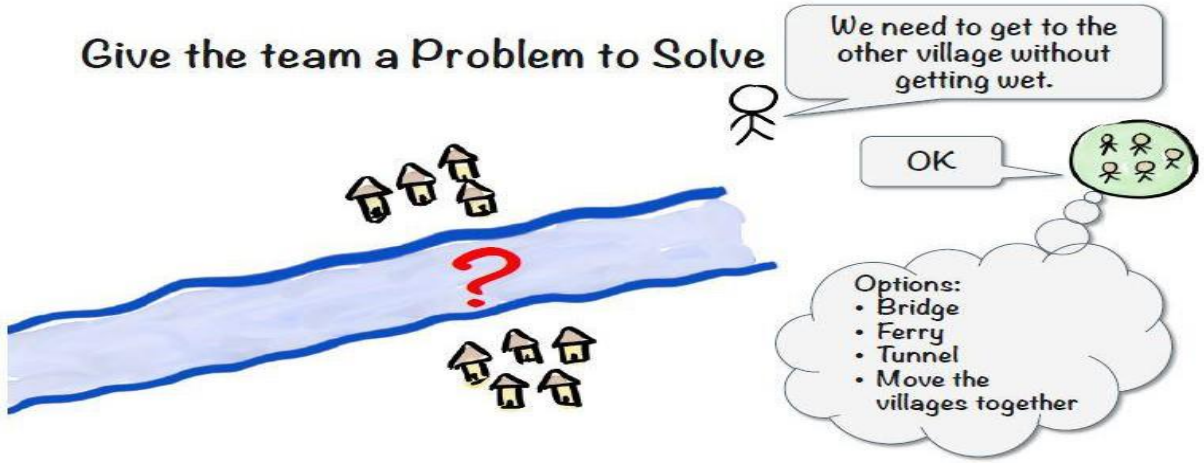
- Biraraya gelmiş,
- Kendi kendilerine organize olan,
- Çapraz fonksiyonlu,
- İşine odaklanmış,
- Hedefleri net olan,
- Teslim edilebilecek düzeyde ürün ortaya koyabilen
- Küçük(3-7 kişilik) gruplar.

Çevik MODEL Takımları



Takımlara çözümü söylemeyin!

Sorunu söyleyin, onlar çözümü üretsin!



Çevik Yazılım Modelinin Diğer Modellerden Farkı

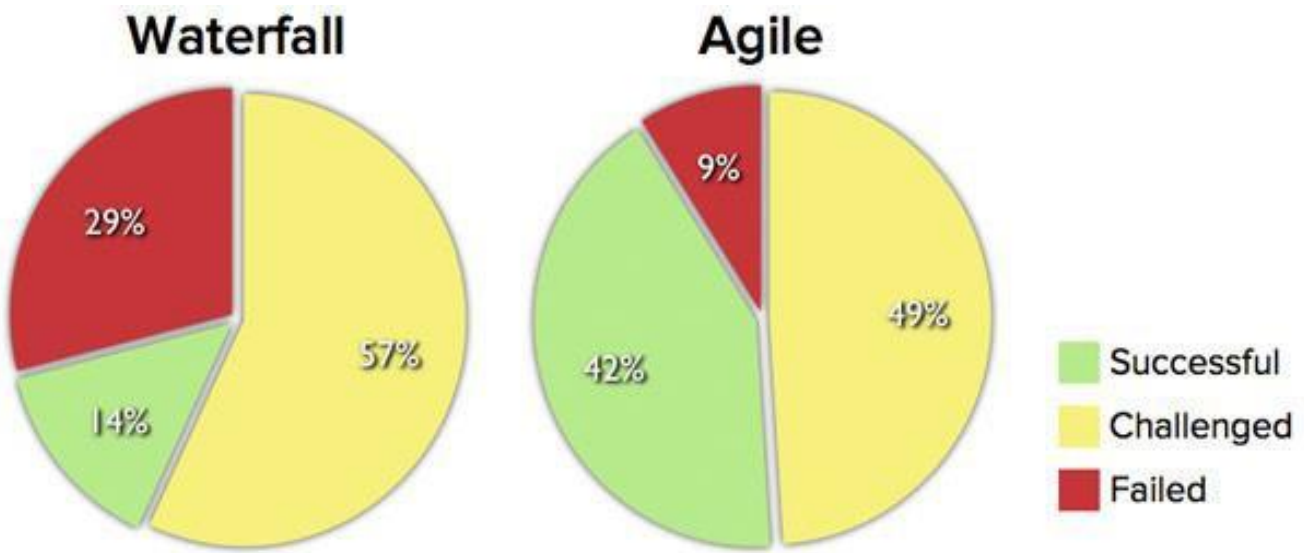
Geleneksel Yöntemlerde

- Müşteriler ne istediğini iyi bilir.
- Geliştiricilerneyi,neşekildeüreteceklerinііyibilir.
- Bu yol boyunca hiç bir şey değişmeyecektir.

Çevik Yöntemlerde

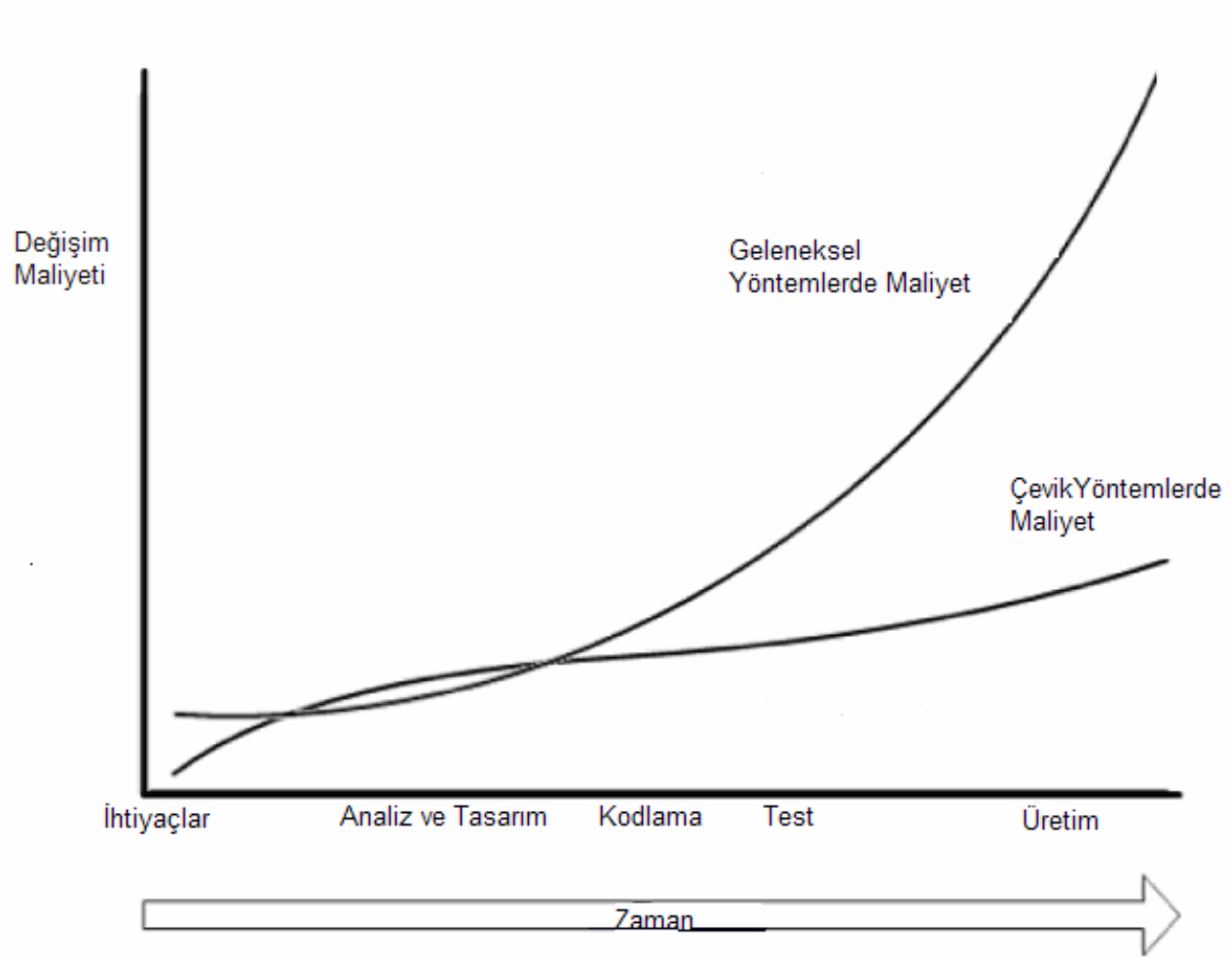
- Müşteriler ne istediğini keşfeder.
- Geliştiriciler neyi nasıl üreteceğini keşfeder.
- Bu yol boyunca birçok değişiklik yapılabilir.

Çevik Yazılım Modelinin Diğer Modellerden Farkı



Source: The CHAOS Manifesto, The Standish Group, 2012.

Çevik Yazılım Modelinin Diğer Modellerden Farkı



Çevik Yazılım Modelinin Diğer Modellerden Farkı

Ölçüm	Çevik Modelleme	Çağlayan Modeli
Planlama ölçeği	Kısa dönemlik	Uzun dönemlik
Müşteri ile geliştirici arasındaki mesafe	Kısa	Uzun
Özelleştirme ve uygulama arasındaki zaman	Kısa	Uzun
Sorunları keşfetmek için zaman	Kısa	Uzun
Proje tamamlanma riski	Düşük	Yüksek
Değişikliklere uyum yeteneği	Yüksek	Düşük

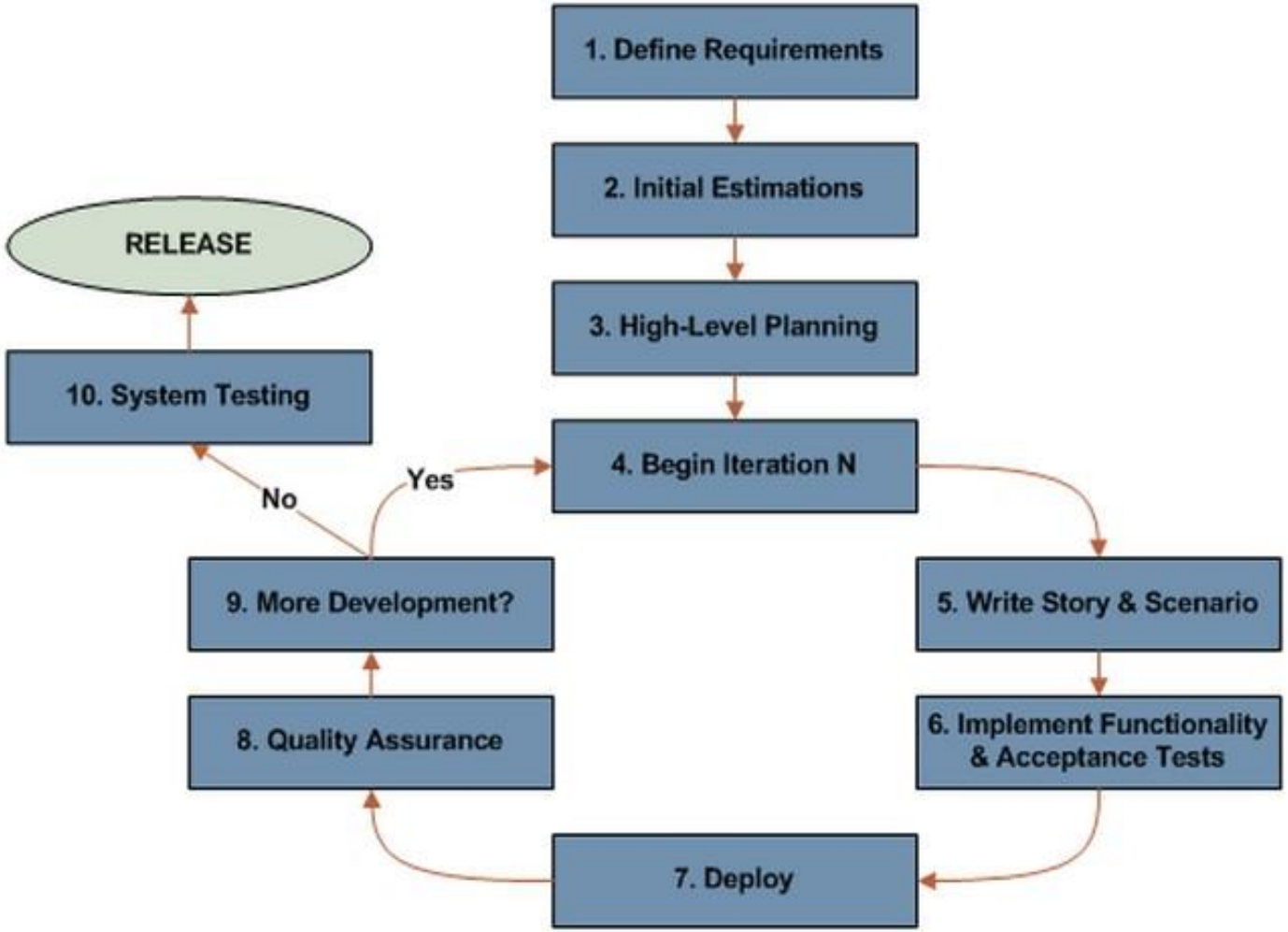
ÇEVİK metotların problemleri

- Sürece dahil edilen müşterilerin ilgisini sürekli kılmak zor olabilir.
- Takım üyeleri çevik metotları tanımlayan yoğun karışmaya uygun olmayabilirler.
- Önceliklerin değişimi birden fazla stakeholder(ortak) olması durumunda zordur.
- Sadeliği koruma fazladan iş gerektirir.
- Tekrarlanan geliştirmeye farklı yaklaşımlar olduğunda sözleşme bir problem olabilir.

80 - 20 Pareto Kuralı:

- Bir projenin %80 'lik kısmı, proje süresinin %20 sinde tamamlanır. Geriye kalan %20'lik iş ise zamanın % 80'ini alır.

Çevik Yazılım Geliştirme Süreci



Çevik Yazılım Geliştirme Modelleri

Çevik modeller tam bir yazılım süreci değildir ama kapsamlı yazılım geliştirme yöntemlerini tamamlayıcı niteliktedir. Başlıca çevik yazılım geliştirme süreç modelleri:

- Uçdeger Programlama (“Extreme Programming – XP”)
- Adaptif Yazılım Geliştirme (”Adaptive Software Development -ASD”)
- Dinamik Sistem Geliştirme Metodu (”Dynamic System Development Method”)
- SCRUM
- CRYSTAL
- Özellik Güdümlü Gelistirme (“Feature-Driven Development – FDD”)
- Çevik Tümlesik Süreç (“Agile Unified Process – AUP”)
- Çevik bilgi Metodu (Agile Data Method)

Uçdeğer Programlama (Extreme Programming – XP)

- Uçdeğer programlama, Kent Beck tarafından 1999 yılında bir yazılım geliştirme disiplini olarak ortaya çıkarılmıştır.
- Tüm gereksinimler senaryolar şeklinde oluşturulur. Daha sonra senaryolar işlere bölünür.
- Yazılımcılar çiftler halinde çalışır ve her iş için test de geliştirir. İşleri sonra
- tümleştirir.
- Sistemin müşterisi de geliştirici takımın devamlı bir parçasıdır.
- 4 prensip etrafında toplanır:
  1. Basitlik,
  2. İletişim,
  3. Geri bildirim,
  4. Cesaret

Uçdeğer Programlama (Extreme Programming – XP)

12 temel pratiğin birlikte uygulanmasını gerektirir:

1. Planlama oyunu
2. Kısa aralıklı sürümler
3. Müşteri katılımı
4. Yeniden yapılandırma (“refactoring”)
5. Önce test (“test first”)
6. Ortak kod sahiplenme
7. Basit tasarım
8. Metafor
9. Eşli programlama (“pair programming”)
10. Kodlama standardı
11. Sürekli entegrasyon (“continuous integration”)
12. Haftada 40 saat çalışma

Uçdeğer Programlama (Extreme Programming – XP)

Bazı özellikleri

Takımın bilgisayarları kübiklerle bölünmüş büyük bir odanın ortasında yer alır.

Bir müşteri temsilcisi geliştirici takımlarla devamlı beraber çalışır.

Hiç kimse aynı iş için peşpeşe iki haftadan fazla çalışamaz.

Takım içerisinde özelleşme yoktur. Herkes belirtim, tasarım, kodlama ve test aşamalarında görev yapar.

Parçalar geliştirilmeden önce ayrı büyük bir tasarım aşaması yoktur. Onun yerine parçalar geliştirilirken tasarım da değiştirilir.

Küçük ve orta ölçekli projelerde kullanılırlar

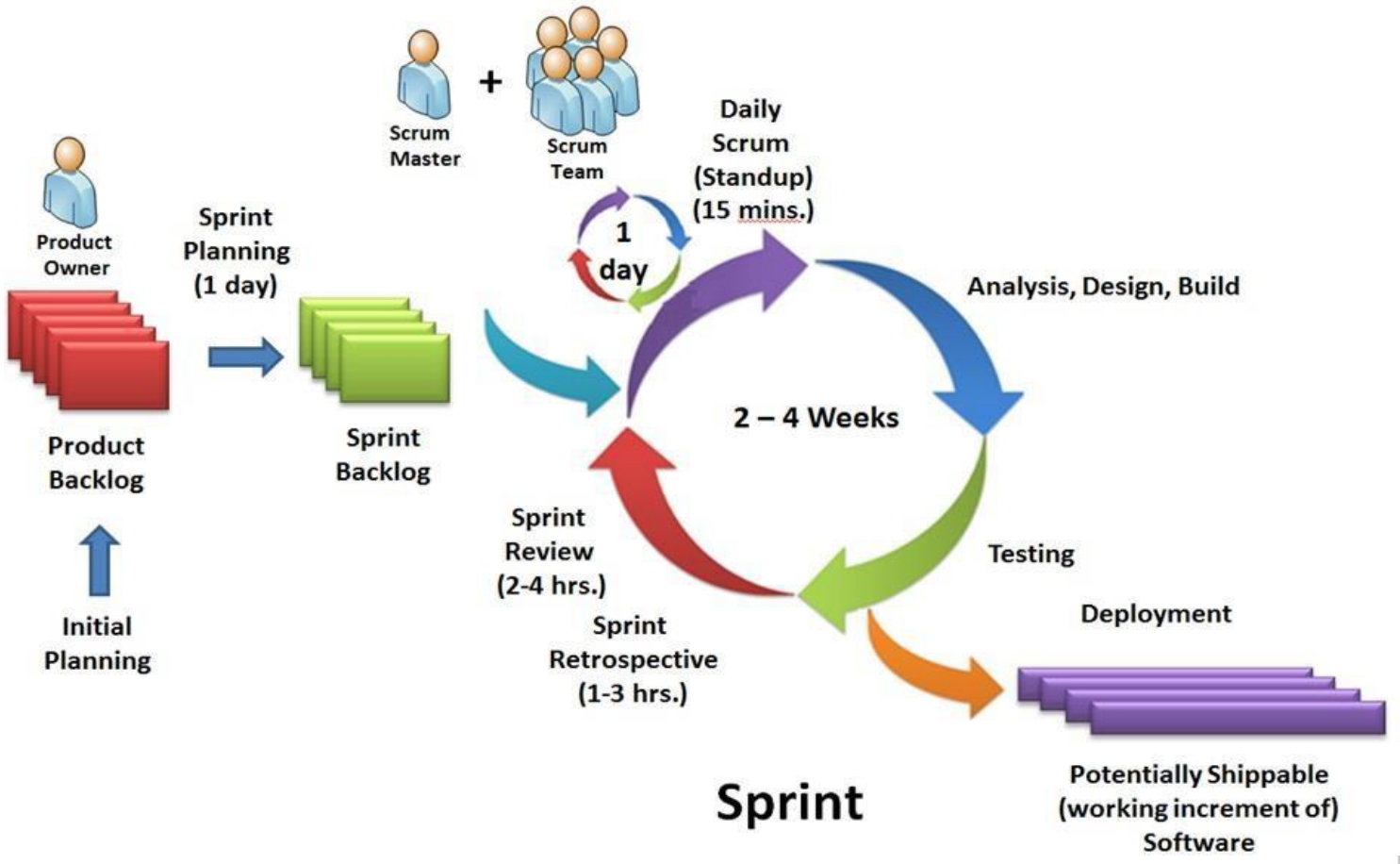
Kullanıcın gereksinimleri belirsiz ya da değişkense kullanışlıdırlar.

Scrum

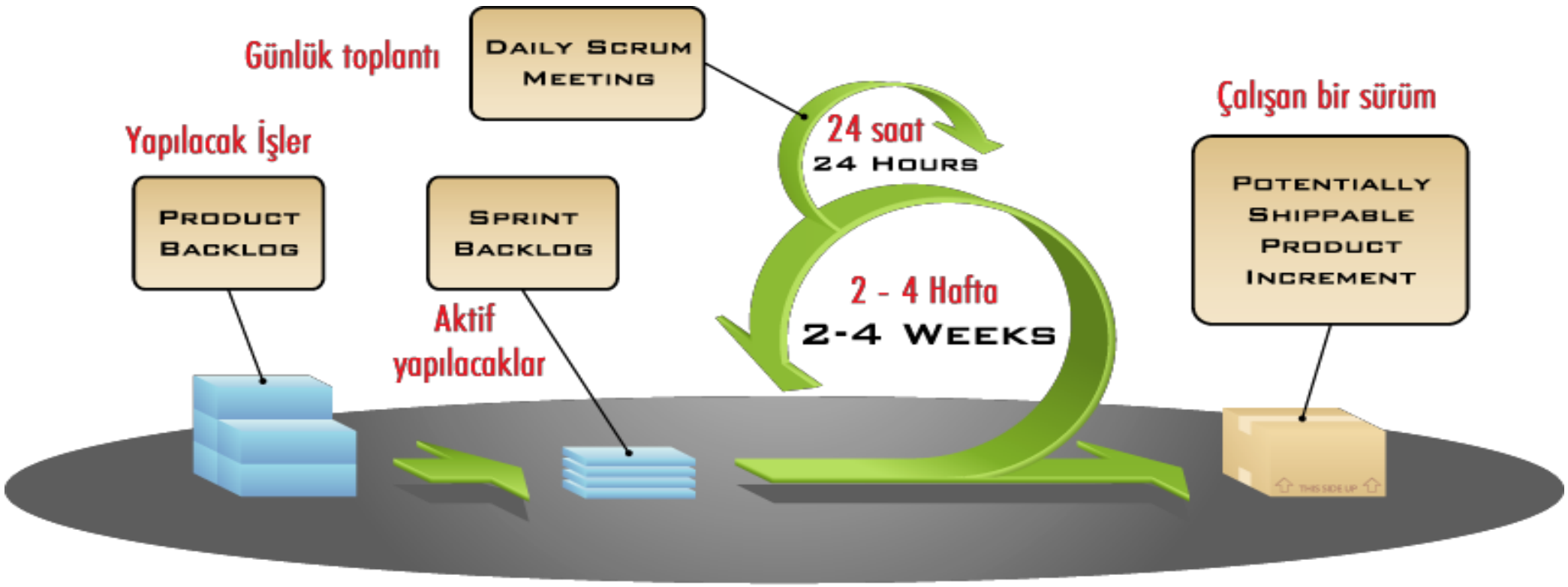
- En genel ve en bilinen Agile frameworküdür.
- Jeff Sutjerland ve Ken Schawaber tarafından 1990'ların ortalarında geliştirilen, proje yönetimi ve planlama ile ilgili yöntemlere odaklı olan ve mühendislik detayları içermeyen bir modeldir.
- Yazılımı küçük birimlere (sprint) bölerek, yinelemeli olarak geliştirmeyi öngörür.
- Bir yinelemenin tanımlanması 30 günden fazla sürmemekte ve günlük 15 dakikalık toplantılarla sürekli is takibi yapılır.
- Karmaşık ortamlarda adım adım yazılım geliştiren küçük ekipler (<20) için uygundur.
- Yüksek seviyede belirsizlik arz eden projelerde, son yıllarda daha yaygın biçimde uygulandığı ve başarılı sonuçlar ürettiği bildirilmiştir.

Scrum

- Geliştirme işi "paket"lere bölümlenmiştir.
- Test ve belgelendirme ürün oluşturuldukça süregelendir.
- İş "sprint" ler halinde gerçekleştirilir ve var olan gereksinimlerin "backlog" u olarak çıkarılır.
- Toplantılar çok kısadır ve bazen başkan bile içermez.
- Zaman aralıklı olarak müşteriye "demo" lar sunulur.



Scrum


Günlük toplantı
DAILY SCRUM MEETING
Yapılacak İşler
PRODUCT BACKLOG
SPRINT BACKLOG
Aktif yapılacaklar
24 saat
24 HOURS
2 - 4 Hafta
2-4 WEEKS
Çalışan bir sürüm
POTENTIALLY SHIPPABLE PRODUCT INCREMENT
COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

Scrum

- Backlog:

➢Müşteriden ve son kullanıcıdan gelen gereksinimleri içerir.

➢"Ne yapacağız" sorusunun yanıtını içerir.

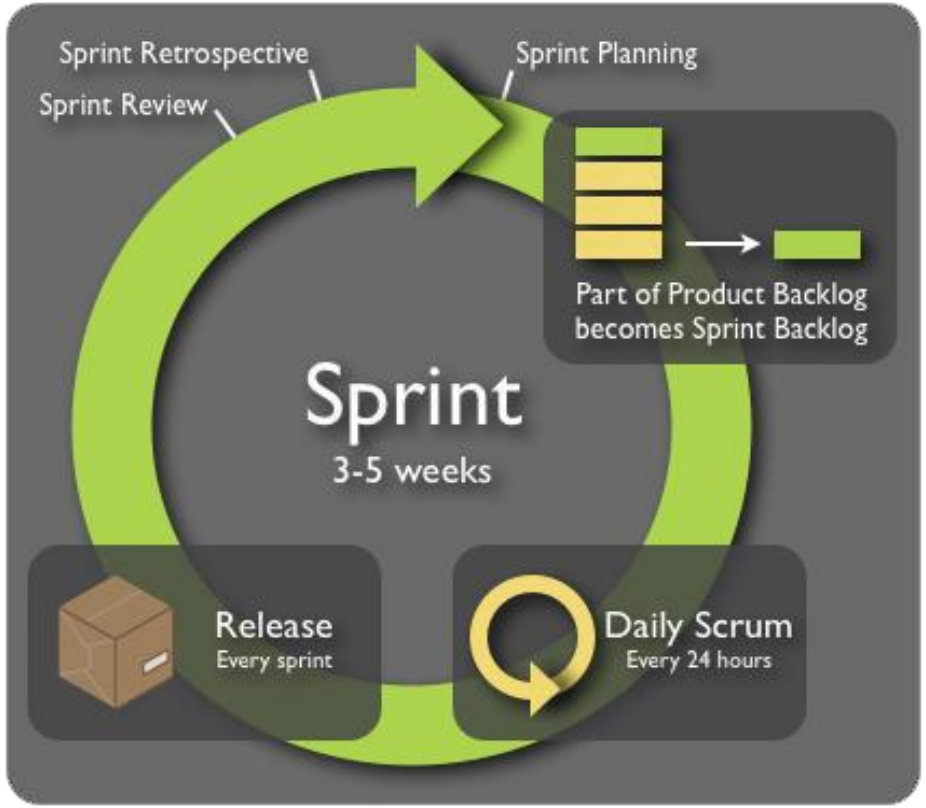
➢Herkese açık ve herkes tarafından müdahale edilebilir.

➢Risk, iş değeri, zaman gibi kavramlara göre ürün sahibi tarafından sıralandırılır.

➢User Story'lerden oluşur.

Scrum

- Sprint:
  - Belirli bir süreye sahiptir.
  - Sonunda ortada değeri olan bir çıktı olmalıdır.
  - Toplantılarla içerik belirlenir.
  - Sprint süresi boyunca her gün toplantılar yapılır.



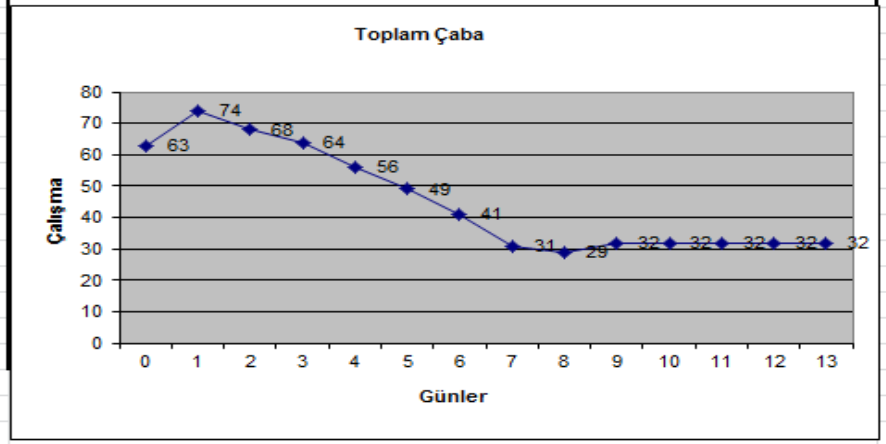
Scrum

- Sprint:

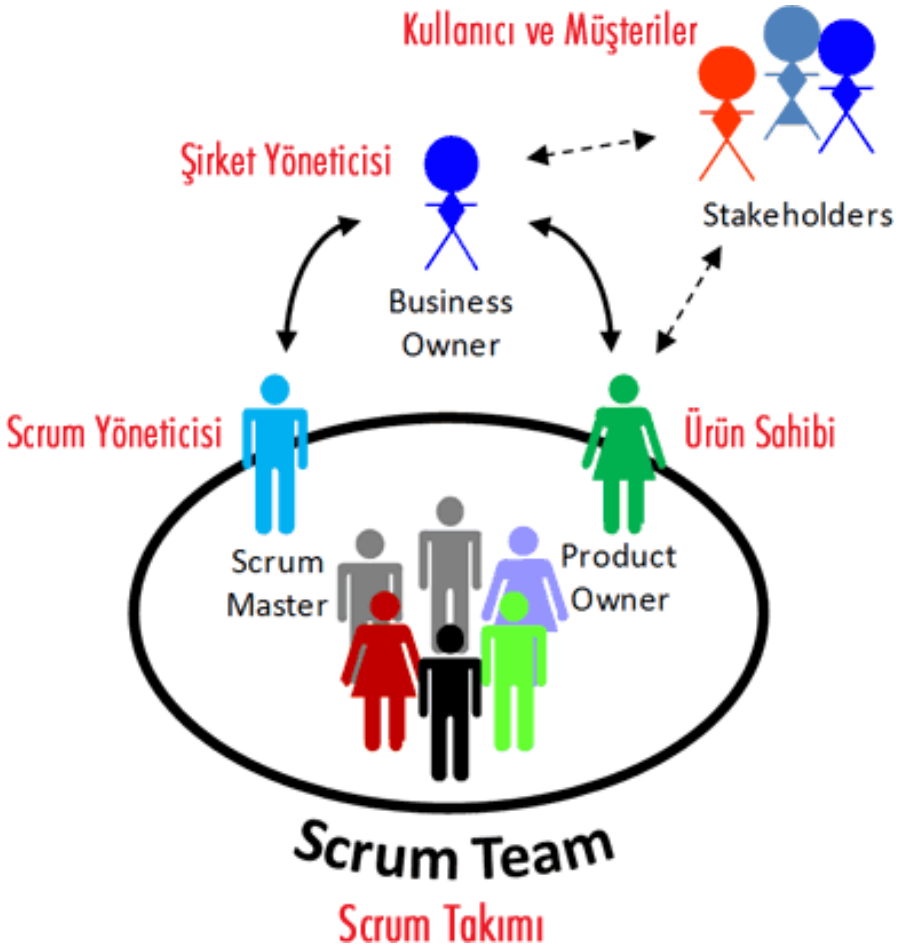
Örnek Bir Sprint (Koşu) Dokümanı (Sprint Backlog)

3. Koşu (Sprint) Gerçek Hava Durumu Eklentisi

Hikaye ID	Hikaye / Görev	Gün / Çaba													
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
		63	74	68	64	56	49	41	31	29	32	32	32	32	32
10	**Hava sağlayıcı sisteminin bir günlük sıcaklık verilerinin getirilmesi**														
	Sunucuya bağlamak ve kimlik doğrulaması	4	16	12	8	3	3	3	3	3	3	3	3	3	3
	Data klasörüne bağlanmak	8	7	7	7	4	0	0	0	0	0	0	0	0	0
	Veriler içerisinden sıcaklık değerinin ayrıştırılması	6	6	4	4	4	1	1	1	1	1	1	1	1	1
	Sıcaklık verileri istemciye iletmek	16	16	16	16	16	16	8	2	0	0	0	0	0	0
11	**Sağlayıcı, yağmur, kar, vb detayları getir**														
	Kar ve yağmurların veriler içinden alınması	4	4	4	4	4	4	4	0	0	0	0	0	0	0
	Kar / yağmur veri istemciye iletilmesi	4	4	4	4	4	4	4	4	4	0	0	0	0	0
	İstemci ekranında yenileme										3	3	3	3	3
	Refactor sunucu kodu										4	4	4	4	4
12	**Serverdanı birkaç günlük verilerin getirilmesi**														
	Serverdan birkaç gün verilerin getirilmesi	10	10	10	10	10	10	10	10	10	10	10	10	10	10
	Istemciye birkaç günlük veriyi iletmek	3	3	3	3	3	3	3	3	3	3	3	3	3	3
13	**Otomatik yenileme özelliği**														
	4 saatte 1 kez sunucudan verileri güncelle	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	İstemciye veri güncelleştirmesi yapmak	2	2	2	2	2	2	2	2	2	2	2	2	2	2



Scrum Rolleri



Scrum Tanımları

Şeffaflık

Sürecin tümleriyle geliştiriciye görünür olmasıdır. Böylece ortak bir görüş ortaya konur. Örneğin, "bitti" takımdakilerin artık işin sonuna gelindiğinde hep bir ağızdan söyledikleri bir ortak bir cümledir. Çalışılan sürecin bittiği anlamına gelir.

Denetim

Scrum, kullanıcılarına yönelik istenmeyen sapmaları tespit etmek için sürecin ve programın sürekli denetim altında tutulması gerekir. Scrum karmaşık ürün gelişimini basite indirgemek için yapılandırılmış bir çerçeve olarak düşünebiliriz

Uyum

Yazılan programların talebe uygun olup olmaması durumudur. Herhangi bir saptama söz konusu ise bu açıklar giderilir ve talebe uyumlu hale getirilir. Saptamaları aza indirmek kısa bir sürede yapılmalıdır.

Scrum Takımı

Ürün Sahibi, Geliştirme Ekibi ve Scrum Master'dan oluşur. Takım kendi kendini örgütler. Böylece kendi içerisinde uyum içinde olan takımlar daha başarılı sonuçlar alırlar. Scrum'ın takım modeli esneklik, yaratıcılık ve verimliliği optimize etmek için tasarlanmıştır.

Ürün Sahibi (Product Owner)

Ürünün değerinden ve gerikaydından (Backlog) sorumlu olan kişidir . Ürün Backlog öğeleri açık olmalıdır. Bunuda sağlayacak olan kişi Product Owner'dir. Sorumluluğun büyük bir kısmı Product Owner'dadır. Takımda Product Owner tektir. Product Owner'in başarılı olması için, tüm organizasyonun kendi kararlarına saygı göstermesi gerekir. Geliştirme Ekibi, Product Owner'in kararlarına göre hareket eder .

Scrum Tanımları

Sprint Backlog

Takım belirlenir ve "Sprint Planning" dediğimiz en fazla 4 hafta sürecek olan küçük döngülerle "Sprint" ile işe başlanır. Her döngü için ürün gerikaydından önemli gereksinimler seçilerek sprint gerikaydı (Sprint Backlog) oluşturulur ve sprint boyunca bu gereksinimler geliştirilir.

Product Backlog

İlk önce müşteriye yani ürün sahibine (Product Owner), istediği ürün gereksinimlerinin neler olduğu sorulur ve bu ihtiyaçlar çıkartılır (Product BackLog). Product BackLog içerisinde maddeler en önemlisinden en aza doğru sıralanır. Belirli bir periyot içerisinde belirtilen gereksinimleri karşılayan tam olarak bir portatif müşteriye teslim edilir .

Sprint Retrospective

Sprintte bir sonraki aşama için iyileştirme fırsatı yakalanır. Bu bir aylık sprint için toplamda 3 saatlik toplantıdır. Burada takım işini kolaylaştırmak için planlar oluşturulur. Böylece gelişim süreci ve sonraki Sprinti daha etkili ve zevkli hale getirmek için uygulamalar geliştirme ortamı yaratılır. Bu toplantılar sonucunda ürün kalitesini arttırma yolları planlanmış olur.

Scrum Tanımları

Scrum Ustası(The Scrum Master)

Scrum'dan sorumlu olan kişidir. Scrum Master, Scrum Takım teorisi ve uygulamaları kurallara uyarak sağlar. Scrum Master takım için liderdir. Scrum Master, ekip tarafından yaratılan değeri maksimize etmek için çalışır. Scrum Master, çeşitli yollarla Ürün Sahibine yardım eder . Bunlar:

- Ürün Backlog yönetimi bulma teknikleri
- Geliştirme ekibinin vizyon ve hedeflerini belirlemek
- Açık ve özlü Ürün Backlog öğeleri oluşturmak için geliştirme ekibine temel yolların öğretilmesi
- Anlaşılabilir bir ortamda uzun vadeli ürün planlamaları yapmak
- Çevikliği anlama ve uygulama

Scrum Master, geliştirme ekibinde hizmet eder:

- Örgütlenmeye koçluk eder
- Geliştirme ekibi için değeri yüksek ürünler oluşturmak
- Geliştirme ekibinin ilerlemesine engel olacak şeyleri ortadan kaldırmak
- Talebe göre Scrum olaylarının kolaylaştırılması

Scrum Master organizasyon içinde çalışır:

- Scrum'ın belirlenmesinde koçluk eder
- Örgüt içinde planlama yapar
- Takımın verimliliğini arttırır