İSTANBUL ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ

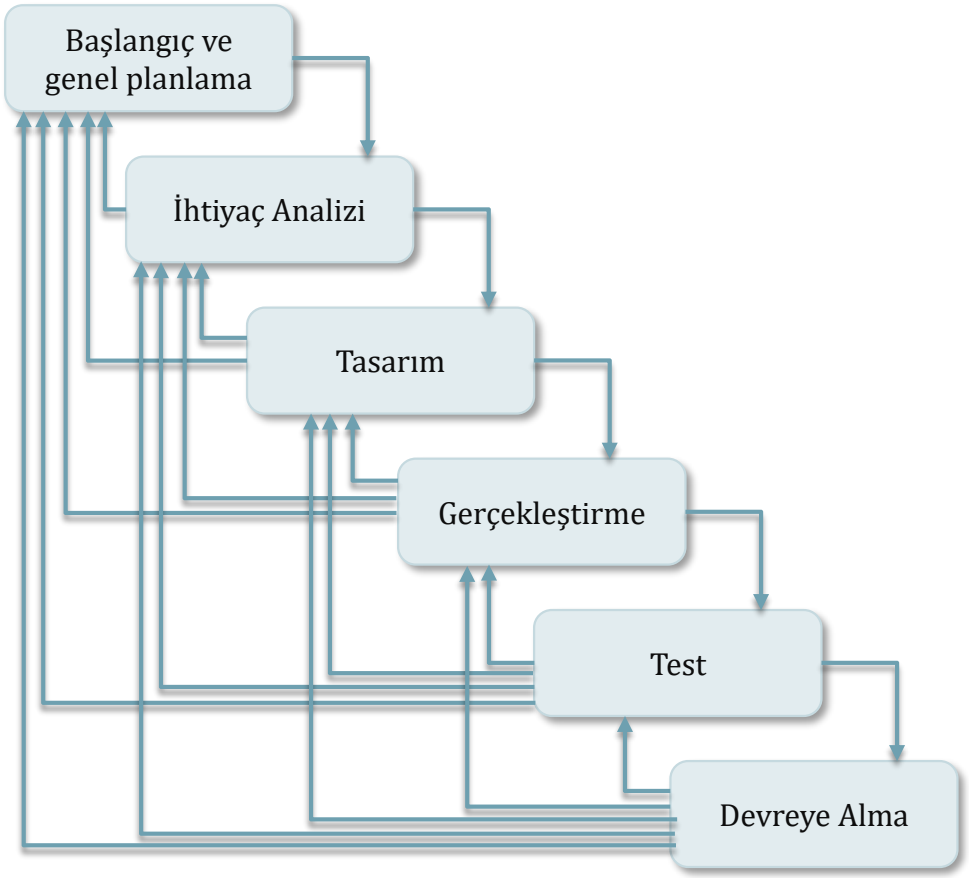
BİLGİSAYAR MÜHENDİSLİĞİ

BIMU4098 – BİLİŞİM PROJE GELİŞTİRME

Dr. Öğr. Ü. Emel ARSLAN

Yazılım Geliştirme Süreci

- Geliştirme aşamaları yazılım mühendisliğinde kısaca:
  - Planlama
  - İhtiyaç Analizi
  - Tasarım
  - Gerçekleştirme
  - Test
  - Devreye alma
  - Bakım



Yazılım Geliştirme Döngüsü

Yazılım Süreç Modelleri

Yazılım ihtiyacını karşılama hedefine doğru yürürken izlenecek farklı yollar, yazılım süreç modelleri olarak standartlaştırılmıştır. ***Süreç modeli*** **aşamaların sırasını ve aralarındaki geçiş kriterlerini belirler.**

Bir süreç modeli seçerek takip etmek,

- gerekli aşamaları hatırlatır ve
- planda önemli bir işlemin atlanmasına engel olur.

Hızlı ve güvenilir bir yazılım geliştirme için yazılım süreçlerinin modellenmesi ve yöntembilim konusunda büyük arayışlar vardır.

Yazılım Süreç Modelleri

1. Yazılımın analizden tasarıma sürekli ilerleyen bir şekilde geliştirilmesi, *doğrusal model* olarak adlandırılır.
   - Şelale (Waterfall) Modeli
   - V Model

2. Yazılım geliştirilirken ihtiyaç ve tasarım kararlarının değişmesi, birçok geri dönüşe yol açar. Bunları yönetebilmek için *yinelemeli-aşamalı modeller* önerilir.
   - Artımlı Model
   - Evrimsel Model
   - Sarmal Model

3. Sürekli değişen ihtiyaçları yönetebilmek için ekip ve ürünü öne çıkartan, resmiyeti azaltan *çevik modeller* de son yıllarda yaygınlaşmıştır.

Yazılım Süreç Modelleri

4. Mevcut yazılım bileşenlerini tekrar kullanmaya dayalı modeller de yazılım geliştirmede yer tutar.
   - Modüler geliştirme,
   - Hazır bileşen
   - Altyapı kullanımı

5. *Servis tabanlı geliştirme* de son yıllarda yaygınlaşan bileşen temelli, her yerden erişilebilen servislere dayanan ve uçtan uca bir yazılım süreç modelidir.

Yazılım Süreç Modelleri

Seçilen süreç modeli ve seçilen modele göre işlemlerin nasıl birbirine bağlanacağı, planlananın nasıl yapılacağını ve yürütüleceğini doğrudan etkiler. Proje yöneticisi;

- Ürün
- Yapılacak proje
- Kurum ve ekibin yapısına göre

en uygun yönetim modelini seçmelidir.

Bazı durumlarda projedeki ihtiyaçlar süreç model seçimini doğrudan belirleyebilir. Örneğin ihtiyaçlar çok fazla değişiyorsa çevik model kullanılabilir.

1-Kod Eksenli Yazılım Geliştirme

Kodla ve düzelt (code and fix) olarak da adlandırılan bu yaklaşım, kullanıcının istediğini hemen gerçekleştirmeye çalışmaktan ibarettir. Proje ihtiyaç analizi tam olarak bitmeden plansız şekilde hemen kodlamaya başlamak veya ihtiyaç analizi kısmen tamamlandıktan sonra tasarım yapmadan hemen kodlamaya başlamak plansız geliştirmeye örnektir. Sistem mimarisinin baştan düzgün ve bütünleşik olarak tasarlanmaması, sonraki aşamalarda sürekli değişim ve sorunlara sebep olur. Sorunlar içerisinde boğuşulurken kullanıcı ihtiyaçlarının anlaşılması da zorlaşır.

Kod eksenli geliştirme bir model değildir ancak çok kullanılır. Çünkü eğitim gerektirmez ve müşteri beklentisine uygun olarak sanki hemen üretime başlanmış izlenimi verir.

Geliştirme sürecinin görünürde analiz, tasarım ve kodlama şeklinde takip edilmesi kaliteli olduğu manasına gelmez. Bir çok projede farkında olmadan analiz ile tasarım birleştirilir. Bu da aslında kod eksenli bir geliştirmedir.

1-Kod Eksenli Yazılım Geliştirme

Avantajları

Yoktur ! Sadece birkaç kişilik küçük projelerde kullanılabilir.

Dezavantajları

Analiz tam yapılmadığı ve kapsam belirsiz olduğundan, müşterinin sürekli değişen isteklerine kodları değiştirerek cevap vermeye çalışan yazılım ekibi kısır döngü içine düşer. *Kodun sürekli değişmesi* kodlama kalitesini düşüren, bir çok hataya sebep olan ve destek maliyetini arttıran bir işlemdir. Bu model kullanıldığında projede hemen hiçbir belge ortaya çıkmaması da ayrı bir dezavantajdır.

1-Kod Eksenli Yazılım Geliştirme

Planlama ve Yönetim Açısından

Kod eksenli yazılım geliştirme genellikle plansız çalışılan ortamların bir özelliğidir. Yapılan kısa vadeli planlar da kullanıcı istek ve hata düzeltmeleriyle sürekli bölünerek bir süre sonra geçerliliğini yitirir.

Kodla ve düzelt yaklaşımında sağlıklı bir yönetimden bahsetmek mümkün değildir. Çünkü sağlıklı bir yönetim olsa zaten bu yöntem tercih edilmeyecektir!!

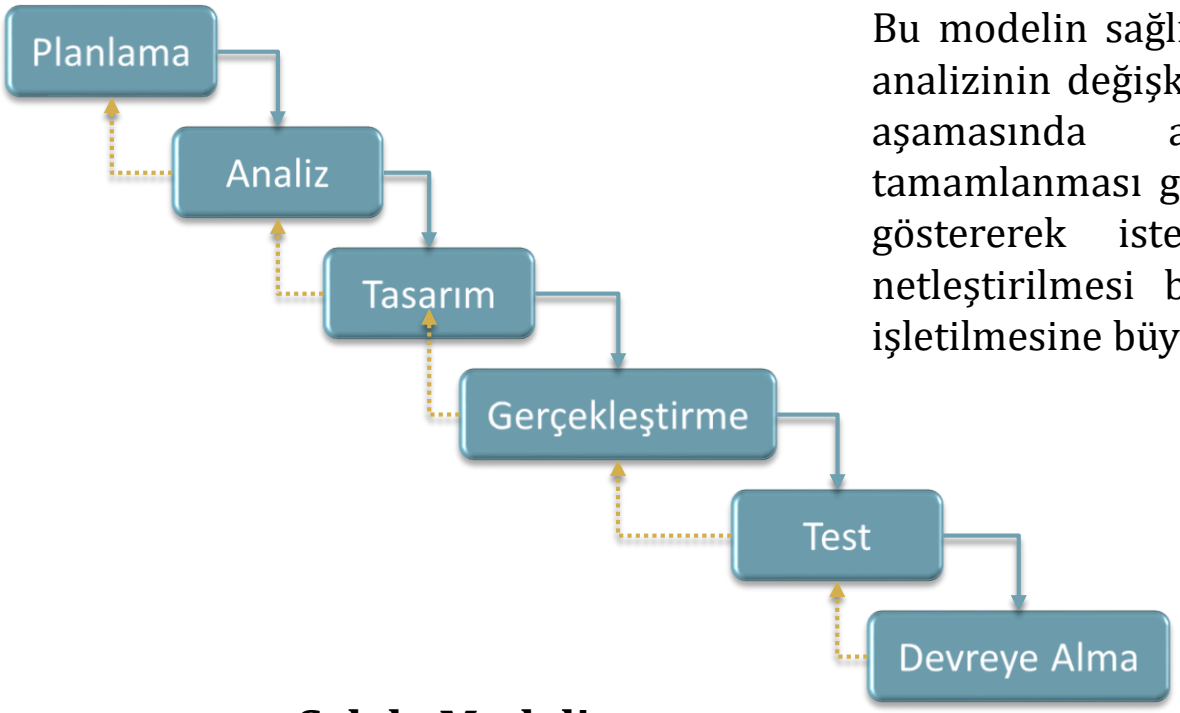
2-Doğrusal Modeller

Doğrusal Modeller, yazılım sürecinde analizden devreye alınıncaya kadarki aşamaların, birbirini ileriye doğru takip ettiği ve fazla bir geri dönüş yaşanmadığı varsayımından yola çıkarak önerilen modellerdir.

Yazılım geliştirmede en sık kullanılan model türü olarak kabul edilebilir. Alternatif modellerin dahi içerisinde kısmen kullanılmaktadır.

2.1.-Şelale Modeli

Şelale modelinde proje, analizden devreye alma aşamasına kadar doğrusal bir şekilde ilerliyor kabul edilir. Analizde kullanıcı ihtiyaçlarının tamamen anlaşıldığı kabul edilerek, tasarıma geçilir. Tasarım tamamen doğru ve bitmiş kabul edilerek gerçekleştirme aşamasına başlanır. Bunu diğer aşamalar takip eder.



Şelale Modeli

Bu modelin sağlıklı çalışabilmesi için ihtiyaç analizinin değişken olmaması ve projenin ilk aşamasında analizin büyük ölçüde tamamlanması gerekir. Müşteriye prototipler göstererek isteklerin erken aşamalarda netleştirilmesi bu yöntemin doğru şekilde işletilmesine büyük katkı sağlar.

2.1.-Şelale Modeli

Avantajları

- Yaygın kullanımı ve müşteri beklentilerine paralel olması önemli avantajlarıdır.
- Özellikle gerçek zamanlı, gömülü çalışan ve hayati risk içeren, dolayısıyla da ilk aşamalarda ihtiyacın belli olması gereken alanlarda bu teknikten faydalanılabilir.
- Aynı zamanda artımlı geliştirme gibi diğer teknikler içerisinde de kullanıldığından bu yöntemin bilinmesi gerekir.

2.1.-Şelale Modeli

Dezavantajları

- En önemli sorunu, kullanıcı ihtiyaçlarının ilk anda tamamen analiz edilememesidir.
- İstekler tam olarak anlaşılsa dahi sonradan değişebilir. Bu da bir çok projede şelale modelinin doğrudan kullanımını zorlaştırır.
- Bu modelde, çalışan bir sürümün ortaya çıkması son aşamalarda, proje başladıktan uzun bir süre sonra gerçekleşir. Bu yüzden müşteri çok fazla beklemek zorunda kalır.
- Yazılım geliştirme ekibindeki bir çok çalışan kendi işlerine sıra gelene kadar atıl bekler.
- Yapılanların doğruluğu ve müşteri taleplerine uygunluğu projenin son aşamalarında kontrol edilir. Bu da problemlerin fark edilmesini geciktiren önemli bir dezavantajdır.

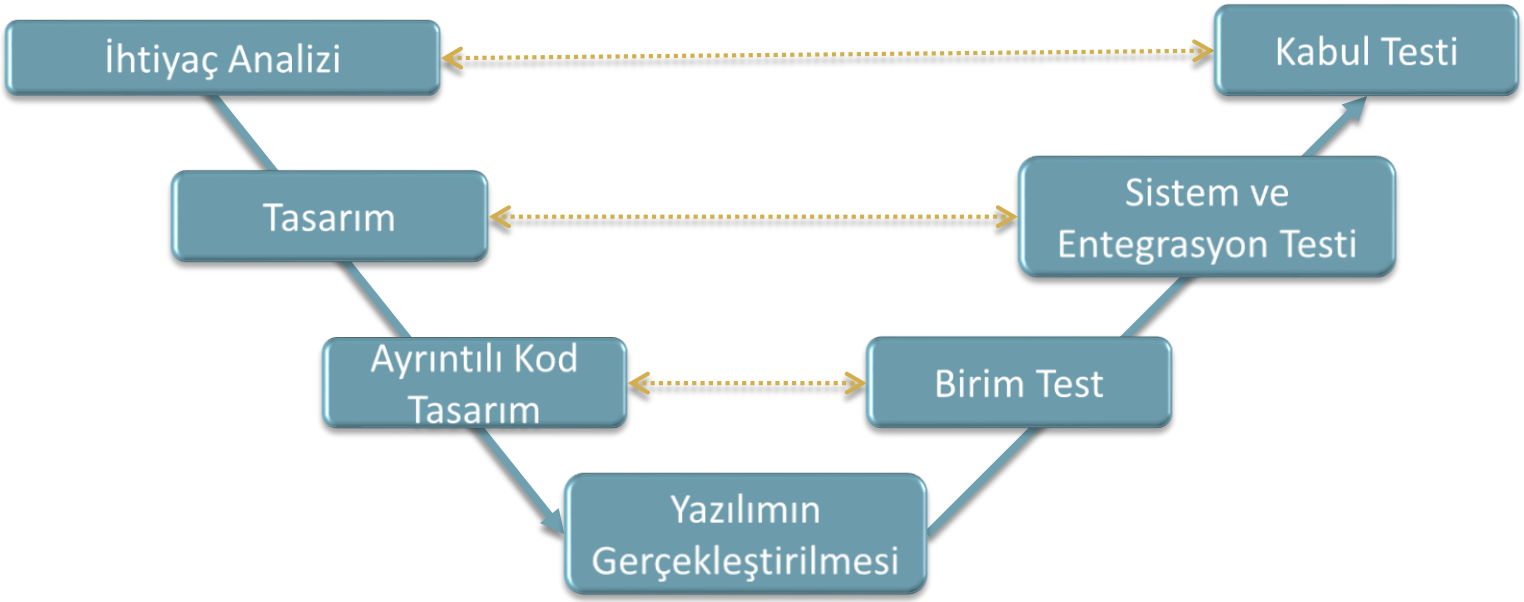
2.1.-Şelale Modeli

Planlama ve Yönetim Açısından

- Şelale modeli ilk anda yapılacakların neredeyse tümünün planlanmasını gerektirir. Ancak bu mümkün olmadığı için planda ciddi sapmalar görülür ve bu sapmalar planın takibini zorlaştırır. Bu sorunları gidermek için analiz ve tasarım gibi her aşamanın başında plan tekrar gözden geçirilip ayrıntılı hale getirilmelidir.
- Proje yönetimi açısından gözden geçirmelerin önemi büyüktür. Şelale modeli kullanılan projelerde, ana gözden geçirme noktaları plan içerisinde analiz, tasarım gibi temel aşamaların sonlarına veya başlarına eklenmelidir.
- Ekip sayısının sabit olması, planlamada kişilerin bir kısmının görevlerinin ilerleyen aşamalara yerleştirmesine sebep olur. Plandaki boş zamanlar teknik hazırlık, güvenlik ve akış gibi her projede karşılaşılan altyapılan hazırlanması için kullanılabilir. Ayrıca bu boş zamanlar önceki projelerin incelenmesi ve eğer yeni teknoloji kullanılacaksa bunun eğitimi gibi faaliyetlerle faydaya dönüştürülmelidir.

2.2.-V Model

V-model, şelale modelinin kontrol safhaları daha organize edilmiş hali olarak görülebilir. Her aşama kendi kontrol aşamasıyla eşleştirilerek "V" harfine benzer şekilde gösterildiği için bu ismi alır. Bu modelde analiz ile kabul testi, tasarım ile sistem bütünleştirme testi, kodlama ile birim testi eşleştirilmiştir.



V-modelin yazılımın geliştirme aşamalarının tanımlanma şekline göre bir çok gösterimi vardır. Örneğin nesneye yönelik programlamada nesne tasarımı ile bütünleştirme testi eşleştirilir. Asıl önemli olan her aşamanın kendi testiyle eşleştirilmesidir.

2.2.-V Model

Avantajları

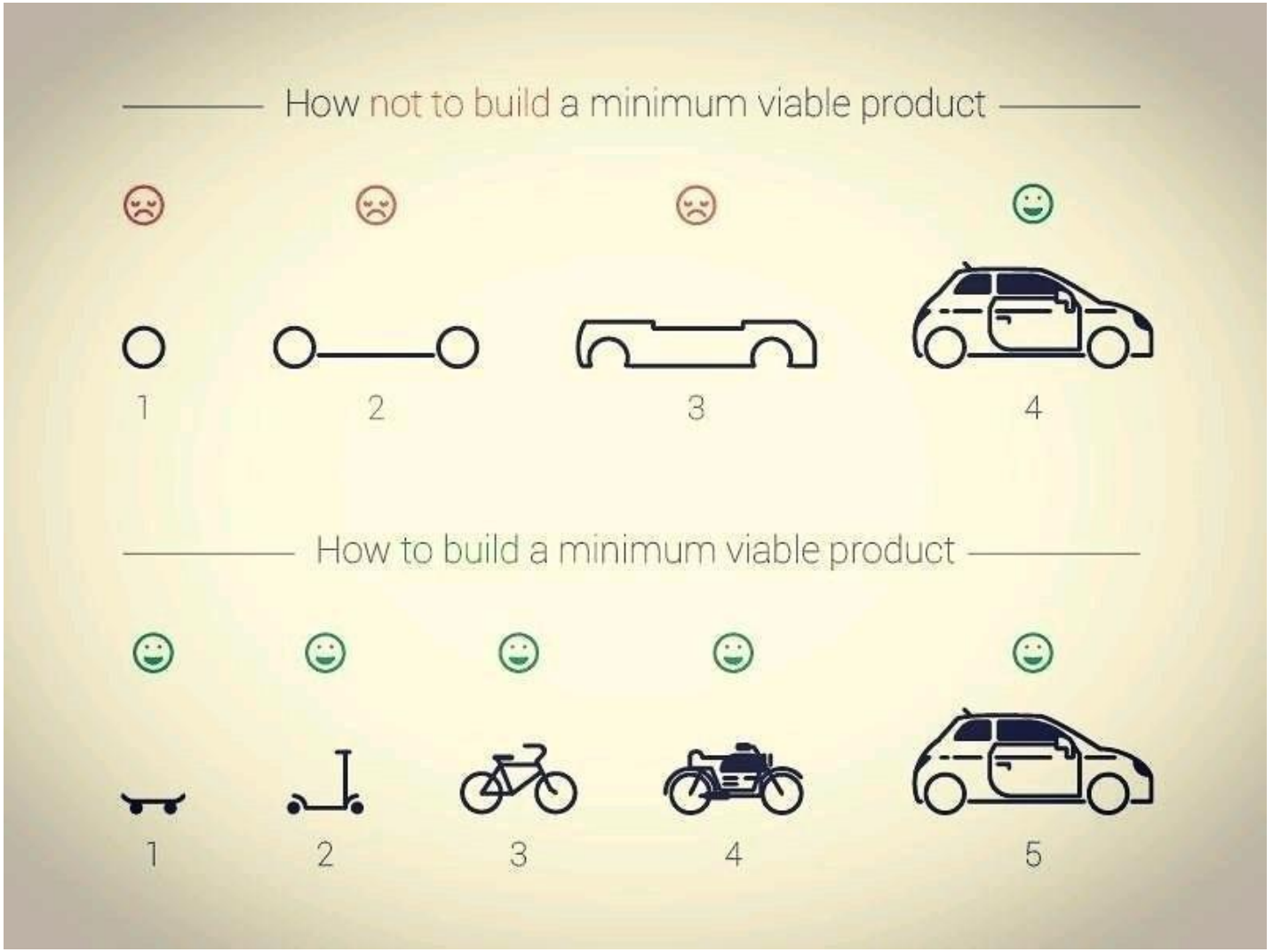
- V-modelinde hangi aşamanın ne şekilde test edileceği belirgin bir şekilde gösterilmiştir. Bu sayede yapılanların testi ve doğrulanması planlı bir sistematik disipline taşınır. Böylece hataların daha kolay fark edilmesi ve düzeltilmesi sağlanabilir.

Dezavantajları

- *V-modelinin, şelale modelinin* test kısmı standartlaştırılmış bir türü olduğu ve yeni bir şey getirmediği ileri sürülmektedir.
- Şelale modelindeki ihtiyaçların ilerleyen aşamalarda anlaşılması, maliyetli geri dönüşler ve ekibin projeye dahil olmasının gecikmesi gibi dezavantajlar V-modelinde de görülmektedir.

Planlama Açısından

- *V-modeli,* planlama açısında da şelale modeliyle benzerlik gösterir. Ancak kontrol aşama ve testlerin standartlaşması bunların unutulmasını önler.
- Test öncelikli geliştirme kullanılarak testin ilk anda planlanması önemli yararlar sağlar. Yapılacak çalışma ile testinin eşleştirilmesi, bu konularda sorumlu ekiplerin erken aşamalarda birlikte çalışmasını sağlar.


How not to build a minimum viable product
1
2
3
4
How to build a minimum viable product
1
2
3
4
5

3-Yinelemeli Geliştirme

Yinelemeli (iterative) veya *aşamalı geliştirme* yazılımı tek bir defada baştan sona değil, çeşitli sürümlere bölerek aşama aşama geliştirmektir. Birçok alt modeli olmakla birlikte *temel yöntem,* önce seçilmiş bir özellikler kümesini içeren temel bir sürüm geliştirmek, sonra bu sürüme adım adım yeni özellikler ekleyerek kullanıcının ihtiyacını karşılayacak nihai sürüme ulaşmaktır. *Yinelemeli modelde tekrarlama* ve *artım* bir arada olmalıdır. *Tekrar* olgunlaşmayı, *artım* gelişmeyi sağlar.

Yinelemeli bir model kullanıldığında kullanıcı yazılımı daha erken kullanmaya başlayabilir. Sürümler adım adım test edilerek olgunlaşacağından hatanın yayılımı engellenir. Belirsizlikleri çok fazla olduğu ilk aşamada ihtiyaç analizi ve tasarımın bütünüyle tek seferde oluşturulmasının ortaya çıkardığı risk de azalır. Yük, aşamalar arasında paylaştırıldığından yazılım ekibi üzerindeki planlama baskısı azalır. Yüksek beklentiler, yazılımın proje ekibinden çok kısa sürede istenmesine neden olabilir. İstenen sürede yazılımın tümünün bitirilemeyeceği açıksa, proje aşamalara bölünür. Adım adım yapılan geliştirme ile en önemli özellikler kısa sürede kullanıcıya sunulur.

3-Yinelemeli Geliştirme

Avantajlar

Yinelemeli geliştirmenin doğrusal modele göre birçok avantajı vardır.

- Kullanıcılara daha kısa sürede bir yazılım sunabilmek
- Yazılım ekibinin daha verimli çalışması; İlk sürümün analizi tüm proje analizinden daha kısa süreceğinden tasarım ekibi projeye kısa sürede dahil olabilir. Aynı durum kod ve test ekibi için de geçerlidir.
- Yazılımdaki değişimlere sistematik olarak gösterme
- Yazılımın özellikleri ve özellikler arası bütünleşmenin sürekli test edilmesi ve doğrulanması
- Her tekrarda yazılımın son durumunun belirlenmesi sayesinde yazılım geliştirme sürecinin izlenebilirliğinin artması
- Adım adım olgunlaşma, hatanın erken tespiti ve yayılımının engellenmesi
- Mimari ve kodlama hatalarının erken tespiti
- Yazılım planında sürelerini kısalması ve sapmaların erken tespiti
- Yazılım tahminlerinin doğruluğunun ilk sürümlerde sınanarak artması
- Planın her sürümde sürekli geliştirilmesi

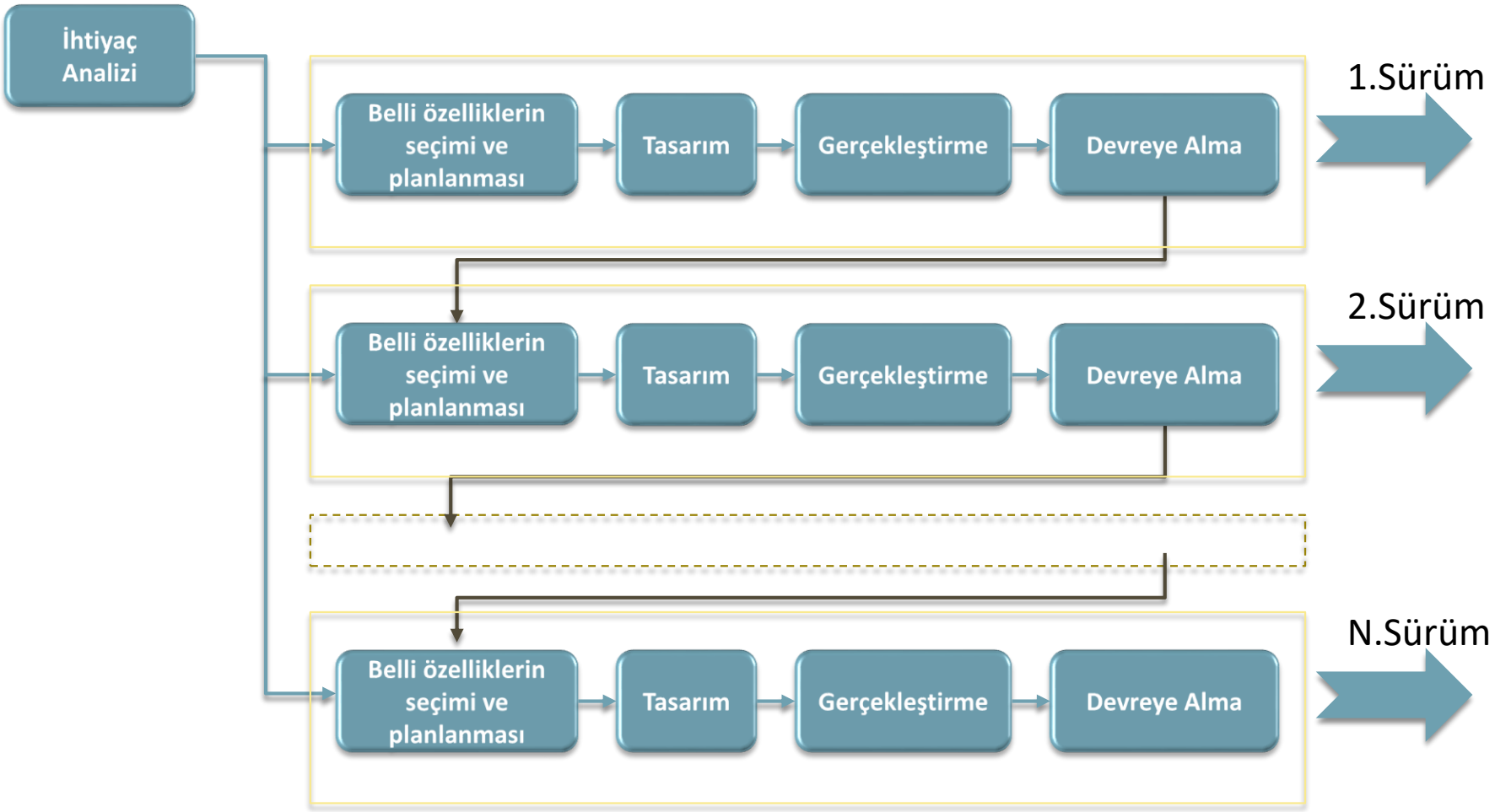
3-Yinelemeli Geliştirme

Dezavantajlar

- Gerçekleştirilmesi düşünülen yineleme sayısı ve her yinelemede yapılacaklar projenin en başında planlanmalıdır. Projenin ilk anındaki belirsizlikler doğru yineleme sayı ve büyüklüğünü seçmeyi engelleyebilir.
- Yapılan her şeyin kalıcı bir tarafı vardır. Herhangi bir tekrarda çıkan ürün müşteri tarafından nihai standart olarak görülebilir. Müşteri, bir sürümde edindiği alışkanlıklar nedeniyle, proje ekibinin ürünü geliştirmek için yaptığı değişiklikleri olumsuz karşılayabilir.
- *Yinelemeli geliştirme* kullanıldığında sürümleri arasındaki bağlantıların doğru kurulması çok önemlidir. Eğer bunlar tamamen bağımsız hale gelirse, proje yöneticisi aynı anda birden fazla projeyi yönetmek zorunda kalır. Aynı anda birden fazla sürümün bulunması geliştirme sürecini daha karmaşık hale getirebilir. Yinelemeli geliştirme tercih edildiğinde konfigürasyon ve sürüm yönetimi çok titiz yürütülmelidir.

3.1.-Artımlı Geliştirme Modeli

Artımlı geliştirme; ihtiyaç analizi büyük ölçüde belli olduktan sonra, yazılım geliştirme sürecindeki işlemin sürümlere bölünerek ve her sürümde ihtiyacın bir kısmı karşılanarak gerçekleştirilmesidir. Yazılım ürünü her sürümde yeni eklenen özelliklerle, daha gelişmiş hale gelir.



3.1.-Artımlı Geliştirme Modeli

Genellikle birkaç haftada bir, sistemin çalışan yeni sürümü ortaya çıkartılır. Sürümler geliştirilir., test edilir ve kullanıcılara veya proje ekibine gösterilir. Bu sayede hataların kısa sürede ortaya çıkması ve sürecin sürekli islenmesi sağlanır.

Sürümlerin özellikleri hedef kitle göz önüne alınarak belirlenmelidir. Bası sürümler proje ekibini hedef alırken, bazı sürümler kurum üst yöneticisinin kısa süreli beklentilerine cevap verir, bazıları da son kullanıcının ayrıntılı ihtiyaçlarına yönelik geliştirilir.

3.1.-Artımlı Geliştirme Modeli

Avantajları

Artımlı geliştirme, yinelemeli geliştirmenin genel avantajlarına sahiptir. Ancak bu avantajlar ilk andaki analizin geçerliliğiyle orantılıdır.

Dezavantajları

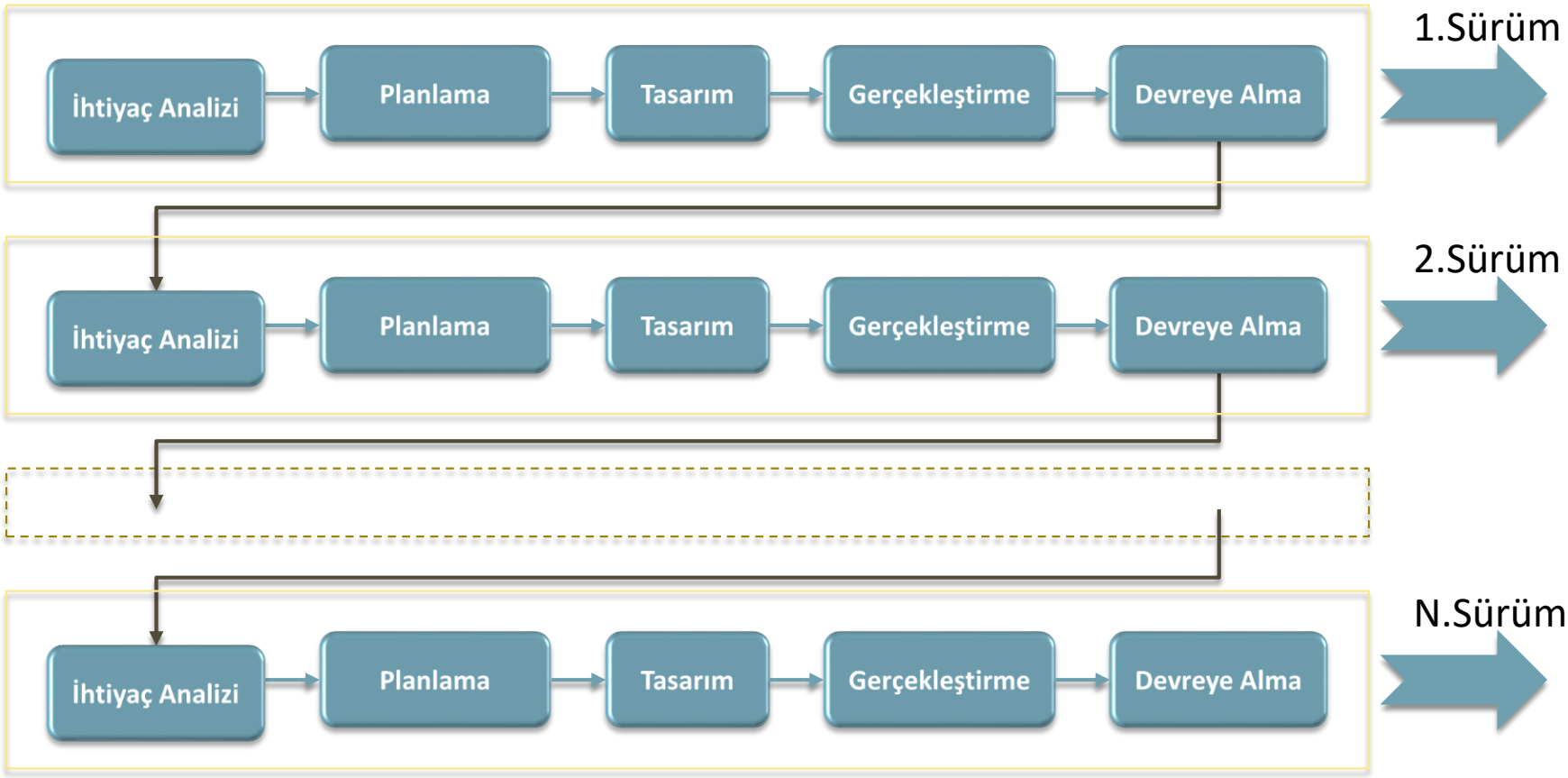
Artımlı geliştirmenin başarısı büyük ölçüde ihtiyaçların ilk aşamada belli olmasına bağlıdır. İhtiyaç analizi ve sürüm planları büyük ölçüde ilk aşamada belirlendiğinden, ilerleyen aşamalarda geniş kapsamlı değişiklikler soruna yol açar. Analizi sıklıkla değişen projeler için uygun bir yöntem olmayacaktır.

Planlama ve Yönetim Açısından

Artımlı geliştirmede, proje planı sürümlerin özellikleri, sürüm oluşturma sıklığı, sürüm tamamlama tarihleri ve bunların birbirleriyle bağlantıları düşünülerek yapılmalıdır. Her sürümde önceli sürümlerden kazanılan tecrübelerle plan güncellenir.

3.2.-Evrimsel Geliştirme Modeli

Evrimsel geliştirme; projenin ihtiyaç analizinden başlayarak geliştirme ve teste kadar tüm aşamaları içine alan ve giderek büyüyen çevrimler şeklinde geliştirilmesidir. Her çevrim önceki çevrimlere bağlantılı küçük bir şelale modeli gibidir.



3.2.-Evrimsel Geliştirme Modeli

*Evrimsel geliştirme*de her sürümde tekrar ihtiyaç analizi yapılır. Projenin her aşamasında kapsamlı değişiklikler istenmesi durumuna karşı üretilmiş bir çözümdür.

Evrimsel geliştirmede her çevrimin en fazla bir ay gibi kısa sürelerde yapılması önerilir. Her çevrim sonucunda kapsamlı bir doğrulama ve onaylama çalışması gerekir.

Avantajları

- Evrimsel geliştirme özellikle ihtiyaçların ilk aşamada belirlenemediği veya sürekli değiştiği durumlar için uygun bir geliştirme şeklidir.
- Projedeki ihtiyaçların olgunlaşması sistematik bir süreçle takip edilir.

3.2.-Evrimsel Geliştirme Modeli

Dezavantajları

- *Evrimsel geliştirme,* projenin başındaki bir tercihten ziyade projedeki değişkenliğin kaçınılmaz bir neticesi gibi düşünülebilir. Hedefin sürekli değişmesi riski vardır. Bu duruma ***hareketli hedef*** ismi verilir.
- Proje ekibi, bir projenin yeni sürümlerinde tekrar baştan analiz yapmak gibi külfetli bir iş istemez.
- Yapılacak işlerin bütçesi sabitse ilk anda tüm özelliklerin analizi gerekir. Bu durumda her yeni sürümde tekrar analiz etmek zaten mümkün olmaz.

Planlama ve Yönetim Açısından

- Planlama açısında oldukça zor bir geliştirme şeklidir.
- Her çevrimde tekrar önemli bir planlama çalışması gerekir.
- Çevrimlerde yapılacak çalışmanın kapsam ve planı büyük ölçüde bir önceki çevrimdeki çalışmaların başarısına bağlıdır.

3.3.-Sarmal Model

Boehm tarafından 1988 yılında önerilmiş olan *sarmal (spiral) model, yinelemeli geliştirme ve şelale modelinin* üstünlüklerini risk temelli bir yaklaşımla birleştirmeyi hedeflemektedir. Bu modelin iki temel avantajı olduğu kabul edilir.

- Sistem netleşir, anlaşılır ve gerçekleştirilen kısmı büyürken riskin azalması
- Tanımlanan kilometre taşları vasıtasıyla geliştirilen sistemin ihtiyaçlarıyla uygunluğu konusunda müşterilerle mutabakat sağlanması ve böylece müşteri memnuniyetinin artması.

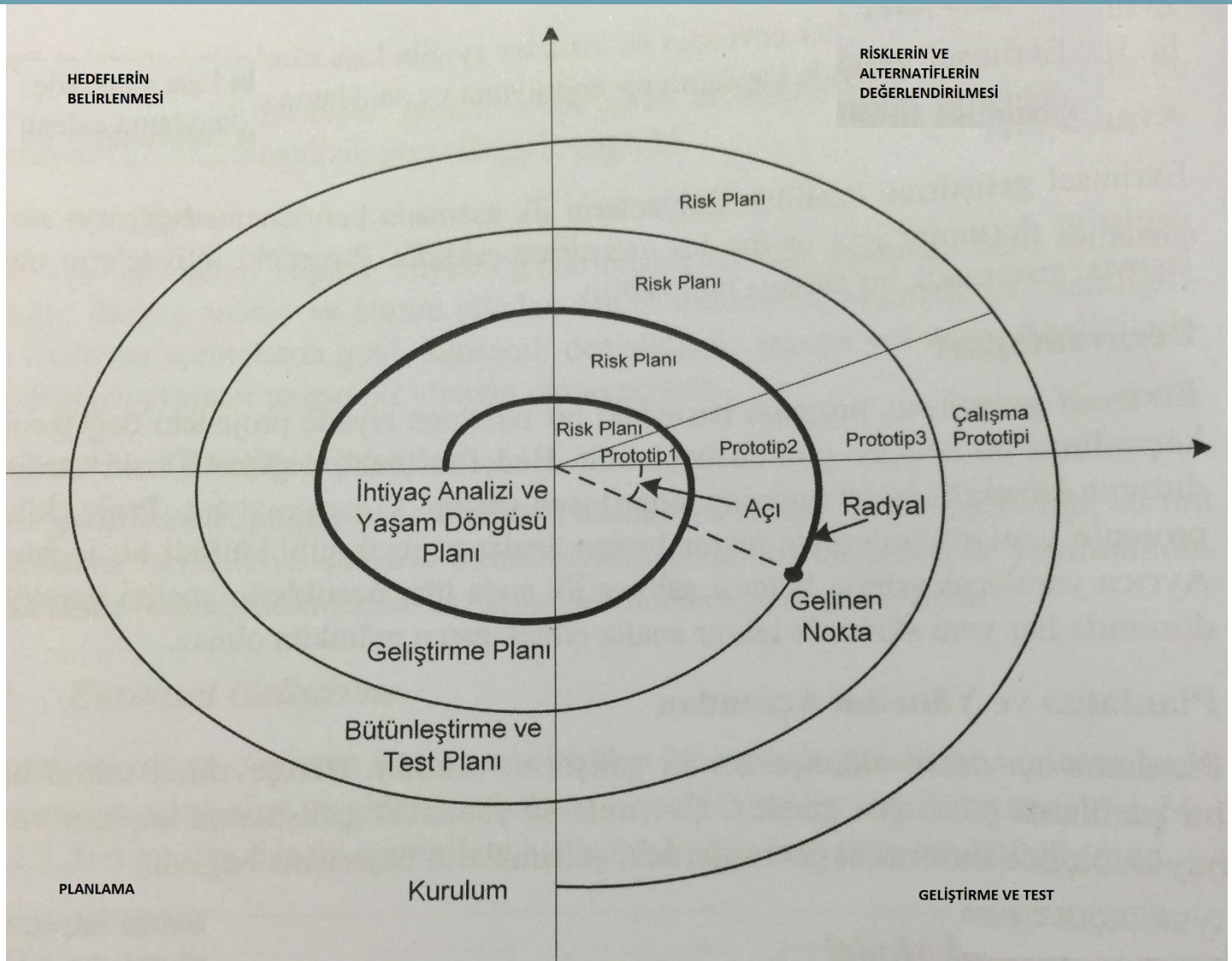
3.3.-Sarmal Model

Her sürüm bir dairesel çevrimdir (sarmalın bir dönüşü) ve geliştirilmesi şelale modeli şeklindedir. İlk sürümler daha çok prototip ve kullanıcı ihtiyaçlarını anlamayı hedeflerken sonraki sürümler geliştirilmesi düşünülen uygulamaya yakın özel fonksiyonlara sahip olmaya başlar. Modelde her sarmal, yaklaşık eşit gösterilmekle birlikte, projede yapılması planlananlara göre gerçek hayatta bazı sarmallar daha dar, daha geniş veya düzgün olmayan şekillerde olabilir.

- Sarmal modelde sistem bir seri evrimsel sürüm şeklinde geliştirilir.
- Daire yayının uzunluğuna dayalı (*radyal*) uzaklık o tarihe kadarki bitirilen işler için harcanan toplam maliyeti,
- açısal uzaklık sarmalın her dönüşünü tamamlamayla sağlanan ilerlemeyi gösterir.

3.3.-Sarmal Model

Şekildeki proje, geldiği noktada *geliştirme ve test* aşamasındadır (açısal). Bu noktaya kadar harcanan toplam maliyet sarmal üzerindeki kalın çizgiyle gösterilmiştir (radyal).



3.3.-Sarmal Model

Boehm, sarmalın her döngüsü için risk analizini de içeren aşağıdaki gibi bir tablo önermiştir. Ayrıca projedeki ön hazırlığı ifade etmek için *sıfırıncı sarmal* tanımı yapmıştır. Döngü *yazılımla iyileştirilebilecek özel kurumsal işlerin* varlığına dayanan bir varsayım ve risk analiziyle başlar. Riskleri azaltacak bir planlama yapılır. Aşağıdaki tablo yeni devreye alınacak örnek bir avans projedeki ilk sarmalda yapılacakları ifade edecek şekilde doldurulmuştur. *Alternatifler* kısmı, her aşamada seçilebilecek farklı yöntem ve araçları göstermektedir. Örneğin istihdam kalemi kendi yapmak veya satın alma arasındaki seçimi gösterir. Her aşama ayrıntılı bir gözden geçirme aşamasıyla sonlanır.

Amaçlar	Personelin avans alma işlemini elektronik ortama taşıyarak hızlandırmak için personel avans giriş yazılımı
Kısıtlar	Belirlenmiş uygun bir maliyet ve sürede, Şirket kültürüne uygun bir şekilde, Üst yönetimin kullanabileceği pratiklikte
Alternatifler	Yönetim : Proje, kurum, politika, planlama Personel : İstihdam, Eğitim Teknoloji : Entegre yazılım geliştirme ortamı, veritabanı
Riskler	Analizin hatalı yapılması
Risk Çözümlemeleri	Prototiplerle analizin gözden geçirilmesi
Sonraki aşamanın planı	Avans talep işleminin iş akış ortamına taşınması
Yapılması onaylanan işler	Personel giriş ekran ve raporu, avans giriş ekranı

3.3.-Sarmal Model

Olumlu Yanları

- Risk yönetimini, yazılım geliştirme sürecine entegre etmek sarmal modelin önemli ve yeni özelliklerinden birisidir.
- Her çevrimde öncelikle projede ortaya çıkabilecek riskler değerlendirilerek bunların çözüm yolları araştırılır.

Olumsuz Yanları

- Sarmal modelde analiz, risk tanımlama ve kullanıcı iletişimi gibi konularda prototip kullanılır.
- Müşteri beklentilerinin erken yükselmesi sorun olabilir.

3.3.-Sarmal Model

Planlama Açısından

Risk planı büyük önem kazanır ve proje riskleri azaltacak şekilde planlanır. Bu konuda özellikle prototip uygulama geliştirilmesi plana dahil edilmelidir. Proje planı her dönüşte müşteriden gelen geri beslemelerle tekrar gözden geçirilmeli ve düzenlenmelidir. Her tekrarda yapılacaklar açıkça belirlenmelidir. Bir tekrarda çok fazla özellik planlanması *şelale modelindeki* sorunları bu modele taşır. Model içerisinde bu tür sorunlar risk cevaplama yöntemleriyle çözümlenmeye çalışılır.

4-Çevik Geliştirme

Çevik geliştirme teknikleri, süreç ve belgeleme yerine doğrudan yazılımın kendisine yoğunlaşan ve yinelemeli geliştirmeye dayanan tekniklerdir. Teknoloji ve kullanıcı beklentilerindeki sürekli değişime cevap olarak doğmuştur. Beck 2001de yayınladığı bir bildiriyle çevik geliştirmenin temel prensiplerini ortaya koymuştur.

- Kişiler ve kişilerarası etkileşime, süreç ve araçtan daha çok önem verme
- Çalışan yazılıma ayrıntılı belgelemeden daha çok önem verme
- Müşteriyle doğrudan iletişimi dolaylı yollara tercih etme
- Planı katı şekilde takip yerine değişimlere cevap verme

Çevik geliştirmede her tekrarın süresi çok kısa, hatta bir gün bile olabilir. *Çevik geliştirmenin* bir çok alt modeli vardır.

- *Aşırı programlamada* sadece bir sonraki artımda geliştirilecek gereksinimler tanımlanır ve her artım genellikle bir aydan kısa sürer.
- *Scrum* isimli çevik modelde, geliştirme süreci bir aydan kısa süren koşulara *(sprint)* bölünür. *Scrum* kullanıldığında şelale modeline benzer şekilde müşteri istekleri bir koşul içinde değiştirilemez.

4-Çevik Geliştirme

Olumlu Yanları

- Değişimden korkmamak ve kolayca uyum sağlama esasına dayanan *çevik geliştirme modeli* küçük ve orta çaplı uygulamalar için uygundur.
- Özellikle kullanıcı ve geliştiricilerin birlikte çalışabileceği kurum içi yazılım birimlerinde uygulanabilir.
- Kullanıcıların geliştirilecek projeyle ilgili tecrübeli, bilgi sahibi ve ne istediğini bilen kişiler olması gerekir.
- Yazılım geliştiricileri bazı kurumlarda görülen gerçekten faydasız ağır bürokratik belge yükünden kurtarabilir.

4-Çevik Geliştirme

Olumsuz Yanları

- *Çevik geliştirmede* analizle ilgili belgeleme yerine, kullanıcı isteklerinin ve isteklerin önceliklerinin kaydedildiği hikaye kartları vardır.
- Tasarım belgesi de kullanılmaz.
- UML gibi modeller bir belge olarak kabul edilir. Ancak tasarımdaki karar ve yöntem tercihlerinin dayandığı sebepleri, bu tür şemalarla anlatmak her zaman mümkün değildir.
- Belgelemenin olmadığı ortamlarda konular büyük ölçüde kişiye bağlı kalmaktadır.
- Sadece kullanıcı isteklerini dikkate almak, yazılım teknolojisinin gelişimi açısından olumsuz sonuçlar doğurur.
- Müşteri geliştirilecek projenin iş ayağıyla ilgili ayrıntılar hakkında bilgi sahibidir. Yazılım ekibi ise bu işleri gerçekleştirmek için bir çok farklı yönteme sahiptir. Örneğin müşteri, kişiler arasında akan işlerin takibini ister ama bir iş akış yazılımı istemez. Sadece müşteri isteklerini düşünmek bu tür yenilikleri önler.
- Bir çok durumda kullanıcıya yeni imkanlar sunmak yazılım geliştiricilerin görevidir. Kullanıcı ihtiyaçlarını karşılamak için kısa vadeli çözüm ve planlama yeterli olabilir. Bununla birlikte orta ve uzun vadede her yazılım ekibi, *kendi işlerini kolaylaştıracak* standart araç ve tekrar kullanılabilecek bileşenler geliştirmelidir.

4-Çevik Geliştirme

Planlama ve Yönetim Açısından

- *Çevik geliştirmede* proje, kısa süreli plan ve anlık kararlarla ilerlediğinden, yönetim proje yöneticisinin kişisel saygınlık ve etkinliğine bağlıdır.
- Ancak son yıllarda teknolojideki hızlı değişime cevap verebilmek için aşırı esneklik gerekmesi, çevik süreçlerin CMMI (Capability Maturity Model Integrated) (Entegre Yeteneklilik Olgunluğu Modeli) gibi resmi belgeye dayalı süreç yönetim modelleri içerisine dahi eklenmesine sebep olmuştur.
- Böylece *çevik geliştirme* daha kurallı bir yapıya kavuşmuş ve yönetimi kolaylaşmıştır denebilir. Bu durum *çevik geliştirme* yönteminin önem ve kullanımının arttığının da bir göstergesidir.

5-Modüler Geliştirme

Modüler geliştirme, yazılımın tek bir parça olarak değil, bir çok bütünleşik parçadan oluşacak şekilde geliştirilmesidir. *Modül* kendi içinde bütünlüğü olan ve gerekirse tek başına çalışabilecek büyük ölçekli yazılım parçası olarak tanımlanabilir.

Büyük bir sistemi en başından küçük sistemlere bölmek, bu sistemi hiyerarşik olarak bölüp yönetmekten farklıdır. Modüler yöntemde analizin ilk hedefi yazılımın bütünleşik parçalarını yani modüllerini tespit etmektir.

Bu aşamadan sonra her modül ayrı bir proje olarak ele alınır. Her biri için ayrı bir ekip oluşturulabilir. Sonrasında her modül *doğrusal* veya *yinelemeli* gibi istenen bir yöntemle planlanır ver gerçekleştirilir.

5-Modüler Geliştirme

Olumlu Yanları

- Modülerlik, karmaşıklığı azaltarak tasarım kolaylığı sağlar. Sistem basit alt sistemler şeklinde tasarlanarak, bir defada çözülecek karmaşa azalır.
- Tekrar kullanılabilirlik sadece koda has bir özellik değildir. Modülerlik, birçok projede ortak olarak kullanılabilecek parçaların ayrı geliştirilmesini sağlayarak tekrar kullanımı arttırır. Güvenlik, akış, kullanıcı arayüzü, iş kuralları ve raporlama gibi her proje mevcut ihtiyaçlar, dolayısıyla ortak kullanılabilecek yazılım parçaları vardır.
- Modülerlik, yazılım geliştirmeyi kurumun çalışma yapısına paralel hale getirir. Modül tespiti için kurumdaki birimler kullanılabilir. Örneğin insan kaynakları, mali işler birimleri için ayrı yazılım modülleri geliştirilebilir. Böylece farklı birimlerin ihtiyaçları farklı ekipler tarafından analiz edilerek geliştirilebilir.

5-Modüler Geliştirme

Olumsuz Yanları

- Modül büyüklük ve amaçları doğru belirlenmelidir. Aksi halde birçok entegrasyon problemi oluşur. Bunu engellemek için modüllerin hem bağımsız hem de bütünleşik çalıştırılabilecek şekilde geliştirilmeleri gerekir. Bu özelliği sağlamak oldukça zordur. Bağımsızlık ve bütünleşme arasındaki denge için, modül içi bağlantıların sayısını arttırmak ve modüller arası bağlantı sayısını azaltmak önerilmektedir.
- Modüllerin farklı ekipler tarafından geliştirilmesi eğer doğru yönetilmezse mükerrer çalışma ve tutarsızlıklara yol açar. Ortak kullanılması gereken bileşenler her modül için ayrı ayrı geliştirilebilir. Örneğin aynı kurumun farklı birimlerde çalışan uygulamalarda hiç gerek yokken farklı güvenlik altyapıları kullanılabilmektedir.

5-Modüler Geliştirme

Planlama ve Yönetim Açısından

- Planın basitleşmesi, izlenebilmesi için anlamlı parçalara bölünmesi gerekir. Hiyerarşik bölümleme dikey bir parçalama sağlar ve plan yine bir bütündür. Buna karşın modülerlik sistemin bir kısmını tamamen ayrı düşünmeyi sağlar. Plan da proje yapısına paralel olarak birbiriyle entegre bir çok yarı bağımsız parçadan oluşur ve basitleşir. Parçaların birlikte çalışabilmesi için, modüller arası bütünleştirme de planlanmalıdır. Bütünleşme ayrı bir proje olarak da yürütülebilir.
- Modülerlik ekibin yönetimini kolaylaştırır. Büyük projelerde yüzlerce hatta binlerce kişiden oluşan ekipler vardır. Böylesi bir ekibi yönetebilmek için daha küçük ekiplere bölmek gerekir.
- Ekip modül yapısına göre bölünebilir. Her modülü ayrı bir ekip geliştirir. Her ekibe ayrı bir yönetici atanabilir. Aynı mekanda çalışma imkanı olmayan ekipler de farklı modüllerde görevlendirilerek iletişim sorunları azaltılabilir.

6-Servis Tabanlı Yazılım Geliştirme

Yazılımlar platformları arası iletişim ve entegrasyon, yazılım dünyasının en önemli sorunlarından birisidir. Bu ihtiyaca çözüm olarak firmalar *Corba (Common Object Request Broker Architecture - Ortak Nesne İstem Aracısı Mimarisi) ve Microsoft COM (Component Object Model)* gibi bir çok öneri sunmuştur. Bu öneriler çeşitli uygulama ihtiyaçlarına cevap vermekle birlikte, firmaya bağımlı ve zor kullanıldıkları için herkesin kabul ettiği standartlar haline gelememiştir. Son yıllarda internetin gelişmesiyle birlikte, *web servisi* yazılımlar arası iletişim ve entegrasyonda en fazla tercih edilen çözüm olmuştur.

Servis, verilen bir web adres üzerinden standart bir mesajlaşma protokolüyle çağırılabilen platform bağımsız bir yazılım bileşenidir. Tüm yapı XML dili üzerine standartlaştırılmıştır. Bu standart yapı sayesinde servis, kod değişikliğine gerek kalmaksızın farklı platformlarda geliştirilen uygulamalardan kolayca çağırılabilmektedir. Servis çağrısında, servisin geliştirildiği platformla ilgili bilgi sahibi olmaya gerek yoktur. Sadece servisin adresi, sağladığı işlev ve çağırılırken kullanılacak giriş-çıkış değişkenlerin yapısının bilinmesi yeterli olmaktadır.

6-Servis Tabanlı Yazılım Geliştirme

Günümüzde servisler ve servisler arası akıştan yola çıkarak *Servis Tabanlı Mimari* olarak isimlendirilen yeni bir yazılım geliştirme mimarisi önerilmektedir. Servisler arası akışı tanımlamak ve işletmek için BPEL (Business Process Execution Language) adında XML tabanlı standart bir dil geliştirilmiştir. *Servis Tabanlı Mimari,* analizden tasarıma ve koda dönüşümü sağlayan araçlar da içermektedir. Bu araçlar temelde üç seviyededir.

- **İş Süreç Analiz Araçları (BPA, Business Process Analysis):** İş analizlerinin BPEL şematik gösterimiyle yapıldığı görsel araçlardır. Hedefleri sadece analizin yapılmasıyla sınırlıdır. Analizin yazılıma dönüşmesi elle (manuel) veya harici araçlarla yapılmaktadır.
- **İş Süreç Yönetim Araçları (BPM, Business Process Management):** İş analizlerinin BPEL yazım şeklinde yapıldığı ve iş akışlarını çalıştırma ortamı sunan araçlar. Çizilen BPEL akışları sadece analiz amaçlı değildir. Bunlar çalıştırılabilir iş akışlarına dönüşmektedir. Bu araçlar akışta kullanılacak kullanıcı ara yüzlerini geliştirmek için de basit çözümler sunar.
- **Bütünleşik İş Süreç Yönetim Araçları (BPMS, Business Process Management Suite):** Analizden kodlamaya kadar tüm işlemleri bütünleşik olarak servis altyapısında sunan yazılımlardır. Yapılan akış temelli analiz otomatik olarak tasarıma, tasarım da kodlamaya dönüşmektedir.

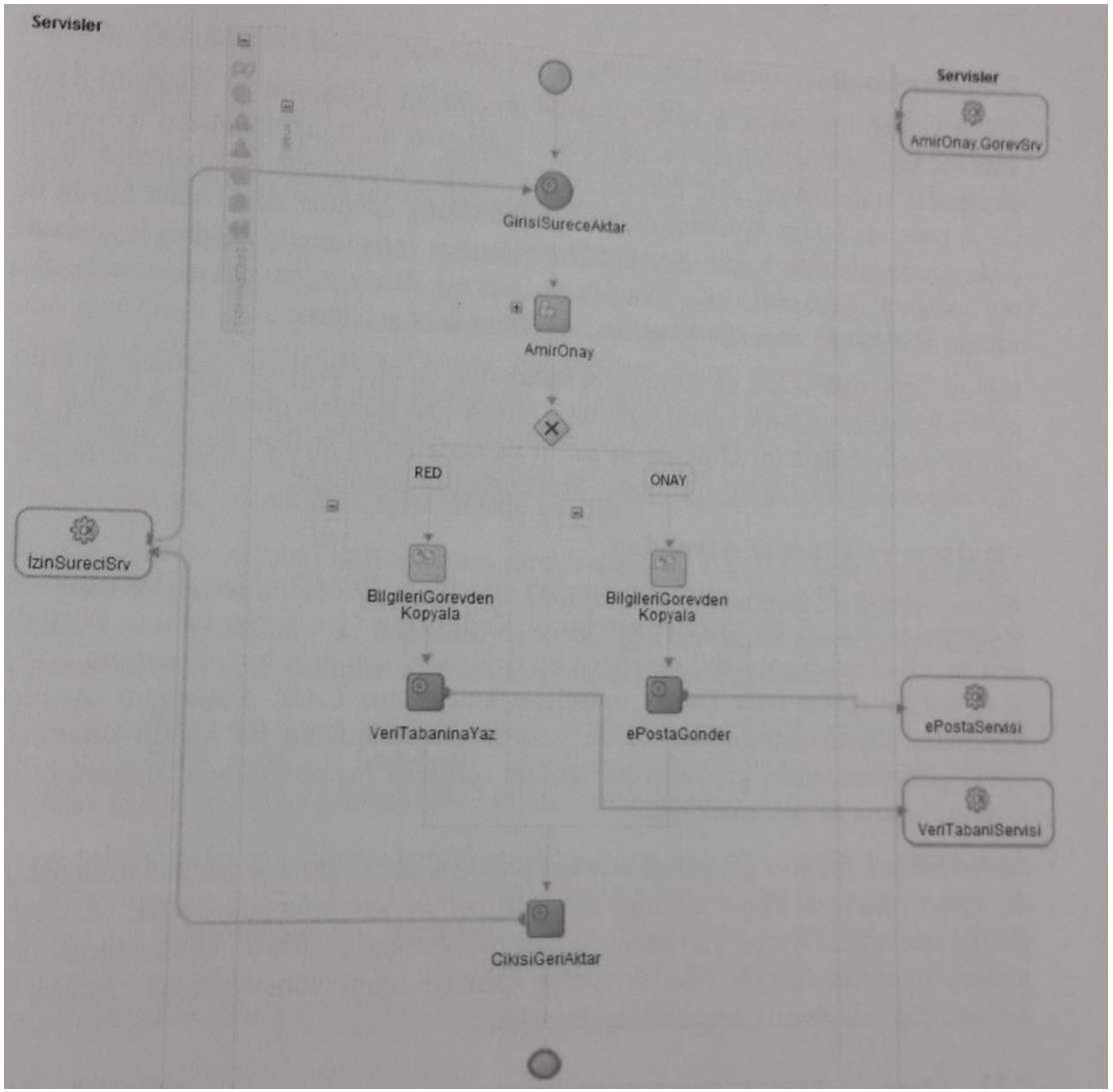
6-Servis Tabanlı Yazılım Geliştirme

Servis tabanlı mimari bazı kullanım zorluklarına rağmen yazılım geliştirme ve süreç tekniğinde, önemli bir gelişme ve değişim olarak kabul edilmektedir. Daha önce kullanılan *bilgisayar destekli yazılım geliştirme ortamlarının (CASE, Computer Aided Software Engineering)* fonksiyonları bu teknolojide standartlaştırılmış ve geliştirilmiştir. *Servis tabanlı mimari* web teknolojilerindeki gelişmeye paralel olarak gelecekte de önemini artarak sürdürecek bir yöntem olarak görülebilir.

Servis tabanlı mimarinin tasarım ortamına ait bir örnek verilmiştir. Analiz ortamı da buna benzer nesneler içermektedir. Servis tabanlı mimaride servisler arasındaki akışın modellendiği grafik araçlar, yazılım geliştirmede kullanılan fonksiyon (servis) çağırımı, döngü ve şart gibi bir çok yapıyı içerir. Bu sayede analizden tasarıma geçiş tam olmasa da otomatik yapılabilmektedir.

6-Servis Tabanlı Yazılım Geliştirme

Şekilde yandaki servis bantlarında çağırılacak veritabanı, mail gönderme ve diğer servisler gösterilmektedir. Ortadaki kısımda modellenen işlemler bu servisleri çağırır ve geri dönen bilgilere göre akış devam eder. İlk adımda dışarıdan bir çağrıyla bilgi alan son adımda bu bilgileri geri aktaran ana süreç de bir servistir.



6-Servis Tabanlı Yazılım Geliştirme

Olumlu Yanları

Servis tabanlı mimari, yazılımlar arası entegrasyon problemine getirilmiş en çok kabul gören çözüm olarak görülmektedir. Uygulama bir yerde geliştirilir ve platform bağımsız olarak birçok farklı yerden çağırılır.

Özellikle *bütünleşik iş süreç yönetim araçları* yeni bir yazılım geliştirme mimarisine işaret etmektedir. Bu ürünler içerisinde sunulan nesne ve akışın tanımlandığı arayüzlerle, analiz aşamasının kullanıcının anlayabildiği şemalarla gerçekleştirilmesi hedeflenir. Analiz, yazılım geliştirmenin gerektirdiği teknik kurallara bağlı olarak otomatik araçlarla tasarıma dönüştürülür. Sonrasında da tasarım, yine dönüştürücü yazılım araçlarıyla kod ve ekranlara çevrilir. Bunlar yapılırken süreç yönetiminin sağladığı izleme, raporlama ve benzetim gibi bir çok avantaj da sunulmaktadır.

6-Servis Tabanlı Yazılım Geliştirme

Olumsuz Yanları

Servis tabanlı bütünleşik iş süreç yönetim araçlarının kullanımı ve yönetimi henüz oldukça karmaşıktır. Ayrıca yazılım süreç aşamaları arasında, dönüşüm için çeşitli yöntemler olmasına rağmen, henüz tam bir standart oluşmamıştır. Analizdeki değişimin tasarıma tekrar uyarlanması gibi geriye dönük işlemler de şu anda büyük ölçüde elle yapılabilmektedir. Servis tabanlı yazılım geliştirmede ilişkisel veri tabanının kullanım şekli de henüz standartlaşmamıştır. Bu yüzden servisler arası akış verilerine ulaşmak ve sorgulamak karmaşık teknikler gerektirir.

Her ne kadar *web servis* ve *BPEL* gibi standartlar da olsa bunların tasarlandığı ürünler içerisinde firmaların kendilerine özel bir çok yan standart olması uyumluluk sorunları oluşturmaktadır. Örneğin *BPEL*'in ilk sürümünde kişilere görev atama standart olarak sunulmadığından firmalar özel çözümler üretmiştir.

6-Servis Tabanlı Yazılım Geliştirme

Planlama ve Yönetim Açısından

Servis Tabanlı Mimari geliştirme araçları (özellikle BPMS) oldukça kurallıdır ve kullanımları önemli bir disiplin gerektirir. Kod, ekran ve raporlar tanıma dayandığından, bir ekranda yapılabilecek basit bir değişiklik için dahi bir çok tanımın değişmesi gerekir. Daha önce benzer amaçlarla kullanılmış *CASE* araçlarının istenilen yaygınlığa ulaşmamasının sebebi de yazılım ekiplerinin bu tür kurallı ortamlara uyum göstermesindeki güçlüklerdir. Benzer zorluklar *Servis Tabanlı Mimari* araçlarını da beklemektedir.

Servis Tabanlı Mimari geliştirme araçları oldukça uzun öğrenme zamanlarına sahiptir. Proje yöneticisi ekibe bu zamanı sağlamalı ve her aşamada destek olmalıdır. *BPMS* araçlarında henüz geri dönüşler zor olduğundan planlarda analiz süresi normalden uzun tutulmalıdır. Bu ürünlerin karmaşıklığını yönetebilecek tecrübe ve uzmanlıktaki ekiplerin istihdamı sağlanmalıdır.

ÖZET

Yazılım geliştirme planlamadan devreye almaya uzanan bir süreçtir. Kullanıcı ihtiyaçlarının anlaşılması, yapılacakların tasarlanması, kodlama, test, devreye alma ve bakım temel aşamalardır.

İhtiyaçların değişkenliği yazılım geliştirme süreciyle ilgili farklı modeller ortaya çıkmasına sebep olmuştur. Yazılım sürecini yönetmek için

- kod eksenli,
- doğrusal,
- yinelemeli,
- çevik,
- modüler
- servis tabanlı mimariler geliştirilmiştir.

Projedeki ihtiyaçlara göre bu mimariler tek başına ya da birlikte kullanılabilir.

Planlama ve analiz aşamalarının öneminin tam olarak anlaşılmaması, *kodla* ve *düzelt* veya *kuralsız* geliştirmeye yol açmaktadır. Günümüzün karmaşık iş ortamında bu yöntem kullanılarak orta ve büyük ölçekli yazılımlar geliştirilmesi mümkün görülmemektedir.

ÖZET

Şelale modeli, analizin başlangıçta tam yapıldığını kabul ederek yazılımı sürekli ileri doğru geliştirmeye dayanır. *V-model*, *şelale modelinin* test aşamaları ayrıntılı planlanmış şekli olarak kabul edilebilir. Her iki modelin de süreçte ortaya çıkan değişiklik taleplerine uyum gösterme özelliği zayıftır. Bu modeller projenin ilk aşamasında yapılacakların çok ayrıntılı bir şekilde planlanmasını gerektirir.

Yinelemeli geliştirme, yazılımın sürümler şeklinde adım adım olgunlaşarak geliştirilmesine dayanır. *Artımlı, evrimsel* ve *sarmal* model gibi alt modelleri vardır. *Artımlı geliştirme,* analizi tamamlanan yazılımın sürümlere bölünerek geliştirilmesidir. *Evrimsel geliştirmede* yazılımın analizi her sürümde tekrar düzenlenir. *Sarmal model* artımlı geliştirme ve şelale modelinin üstünlüklerini birleştiren risk yönetimine dayalı bir modeldir. Bir *yinelemeli geliştirme* modeli kullanılacaksa proje planı sürümleri ve sürümlerdeki özellikleri içerecek şekilde yapılmalıdır. Her yeni sürümde plan gözden geçirilerek daha ayrıntılı hale getirilir.

ÖZET

Çevik geliştirme, müşteri ihtiyaçlarının sürekli değiştiği küçük ve orta ölçekli projelere daha uygun, resmiyeti ve belgeleme ihtiyacı az bir yöntemdir. Kısa bir planlamadan sonra müşterinin de doğrudan dahil olduğu bir ekiple proje geliştirilmeye başlanır. Sürekli yeni sürümler çıkartılarak proje nihai hale getirilir.

Modüler geliştirmede, proje temel özellikler etrafından şekillenen modüllere bölünerek karmaşıklığı azaltmak hedeflenir. Modül bazında planlama yapılarak izlenir. *Modüler geliştirme,* diğer yöntemlerle birlikte kullanılabilir.

Servis tabanlı mimari platform bağımsız çağrılabilen web servislerine dayanan, *web* servisleri arası koordinasyon için standart geliştirme araçları öneren, yeni ve gelecek vaat eden bir teknolojidir. Bu teknoloji, analizden tasarıma kadar tüm aşamaların bütünleşik ve standart görsel bir ortamda modellenmesine imkan sağlar. Ancak henüz geliştirme araçlarının kullanımı oldukça karmaşıktır.