YAZILIM MÜHENDİSLİĞİ – SOFTWARE ENGINEERING

# YAZ16103
# Yazılım Mühendisliğine Giriş

Dr. Öğr. Üyesi Bora ASLAN

İÇİNDEKİLER

# 2.BÖLÜM
# YAZILIM SÜREÇLERİ

Yazılım süreci, bir yazılım sisteminin üretimiyle sonuçlanan ilgili etkinlikler dizisidir. Farklı şirketlerde kullanılan süreçler, geliştirilen yazılımın türüne, yazılım müşterisinin gereksinimlerine ve yazılımı yazan kişilerin yeteneklerine bağlı olarak değişmektedir.

Farklı yazılım süreçleri olmasına karşın, bu süreçlerin hepsi, Bölüm 1'de giriş yaptığım temel yazılım mühendisliği etkinliklerini bir şekilde içermelidir:

1. *Yazılım spesifikasyonu* Yazılımın fonksiyonelliği ve uygulanmasındaki kısıtlamalar tanımlanmalıdır.

2. *Yazılım geliştirme* Spesifikasyonu karşılayan yazılım üretilmelidir.

3. *Yazılım geçerleme* Müşterinin istediğini gerçekleştirdiğinden emin olmak için yazılım geçerlenmelidir.

4. *Yazılım evrimi* Yazılım, değişen müşteri ihtiyaçlarını karşılamak için evrim geçirmelidir.

# YAZILIM SÜREÇ MODELLERİ

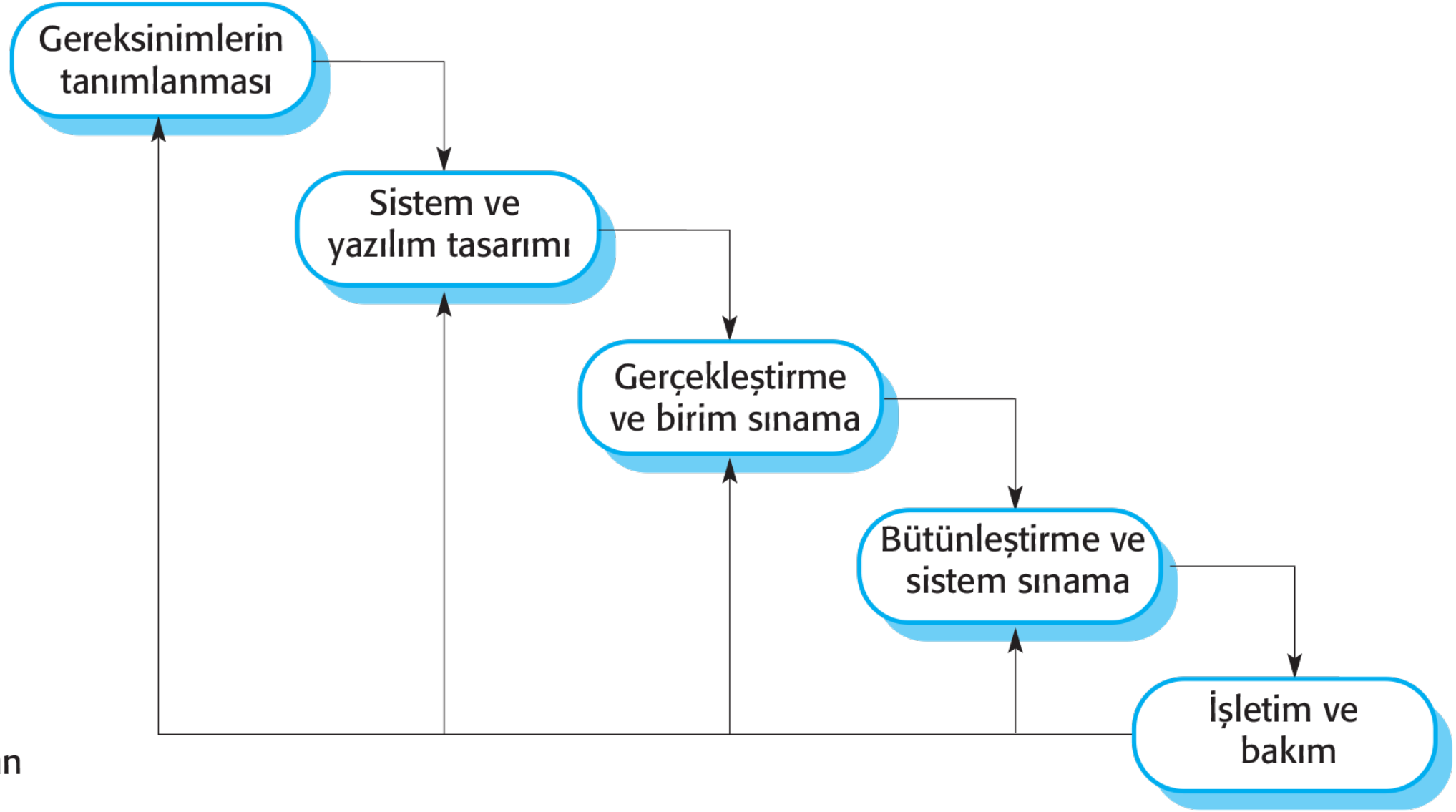
Her süreç modeli, süreci belli bir açıdan ele aldığından söz konusu süreç ile ilgili yalnızca kısmi bilgi sağlar. Örneğin, bir süreç etkinlik modeli, etkinlikler ve bunların sıralamasını göz önüne serebilir; ancak bu etkinliklerde yer alan kişilerin rollerini ortaya koymayabilir.

Bu genel modeller, farklı yazılım geliştirme yaklaşımlarını açıklamak için kullanılan yazılım süreçlerinin üst-düzey, soyut tanımlamalarıdır. Bu modeller, daha belirli yazılım mühendislik süreçleri yaratmak amacıyla genişletilebilen ya da uyarlanabilen süreç çerçeveleri olarak da düşünülebilir. Bu bölümde ele alacağım genel süreç modelleri aşağıda yer almaktadır:

1. Çağlayan modeli
2. Artırımlı geliştirme
3. Bütünleştirme ve konfigürasyon.

## Rational Birleşik Süreci (RBS) (İng. The Rational Unified Process-RUP)

Rational Birleşik Süreci (RBS), burada sözü edilen genel süreç modellerinin hepsini bir araya getirerek yazılımın prototiplemesini ve artırımlı teslimini destekler (Krutchen 2003). RBS, genel olarak üç perspektifle açıklanmaktadır: Modelin zaman içerisindeki evrelerini gösteren dinamik perspektif, süreç etkinliklerini ortaya koyan statik perspektif ve süreçte kullanılabilecek iyi uygulamaları öneren uygulama perspektifi. RBS, şu evrelerden oluşmaktadır: sistemin iş durumunun ortaya koyulduğu başlangıç evresi; gereksinim ve mimarinin geliştirildiği detaylandırma evresi; yazılımın gerçekleştirildiği inşa etme evresi; ve sistemin uygulamaya konulduğu kullanıma geçiş evresi.

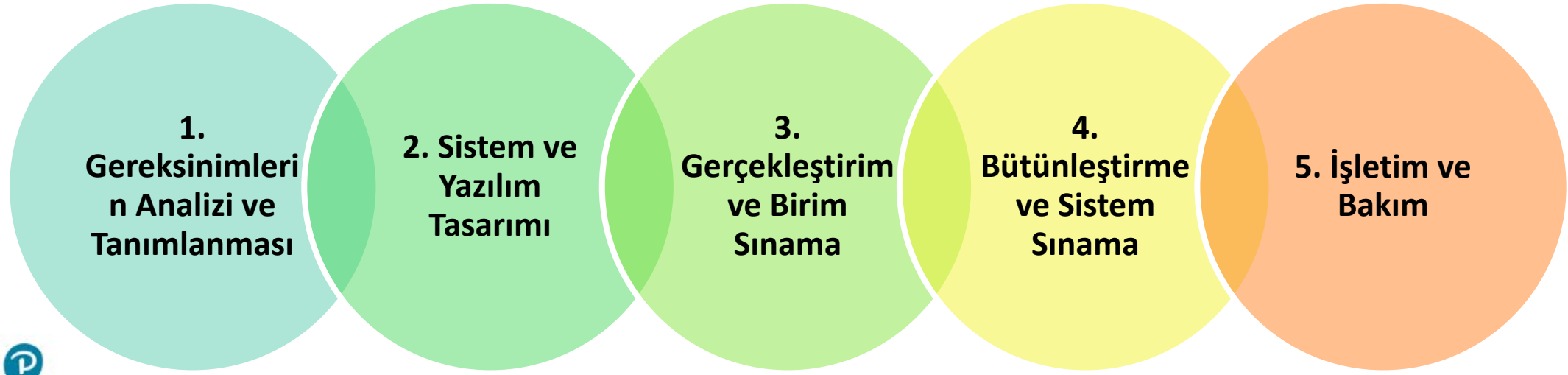



**Şekil 2.1** Çağlayan modeli

# Çağlayan Modeli

Yazılım geliştirme sürecinin yayınlanmış ilk modeli, büyük askeri sistem mühendisliklerinde kullanılan süreç modellerinden yola çıkarak geliştirilmiştir. Bu model, yazılım geliştirme sürecini Şekil 2.1'de gösterildiği şekilde aşamalar halinde ortaya koyar. Bir evreden diğerine art arda akış söz konusu olduğundan bu model çağlayan modeli ya da yazılım yaşam döngüsü olarak bilinmektedir.

Çağlayan modelinin aşamaları, temel yazılım geliştirme etkinliklerini yansıtmaktadır:



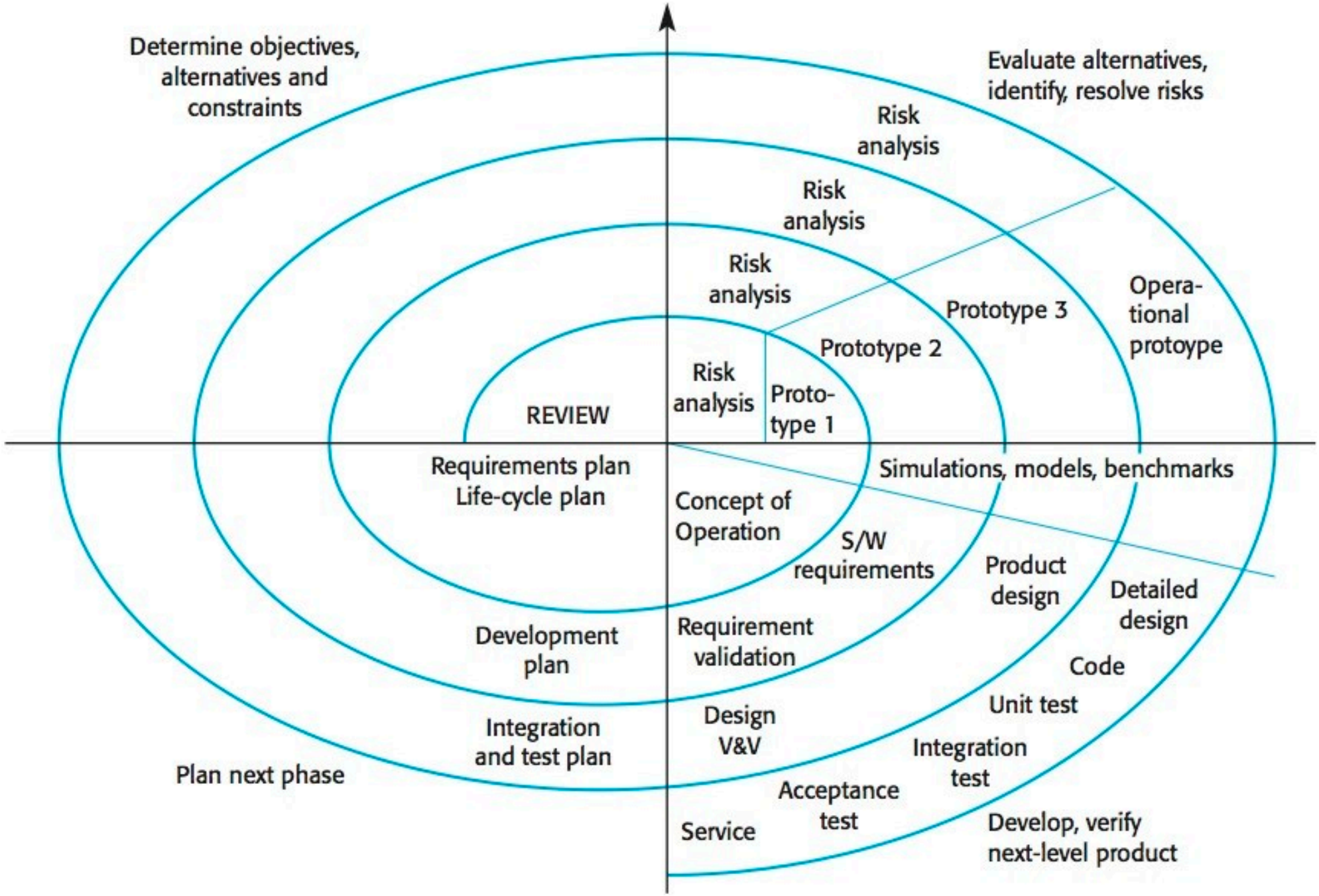
Barry Boehm
Born in1935
In America
Software engineer,
Distinguished
Professor at the
University of
Southern California

MyShared

## Boehm'in sarmal süreç modeli

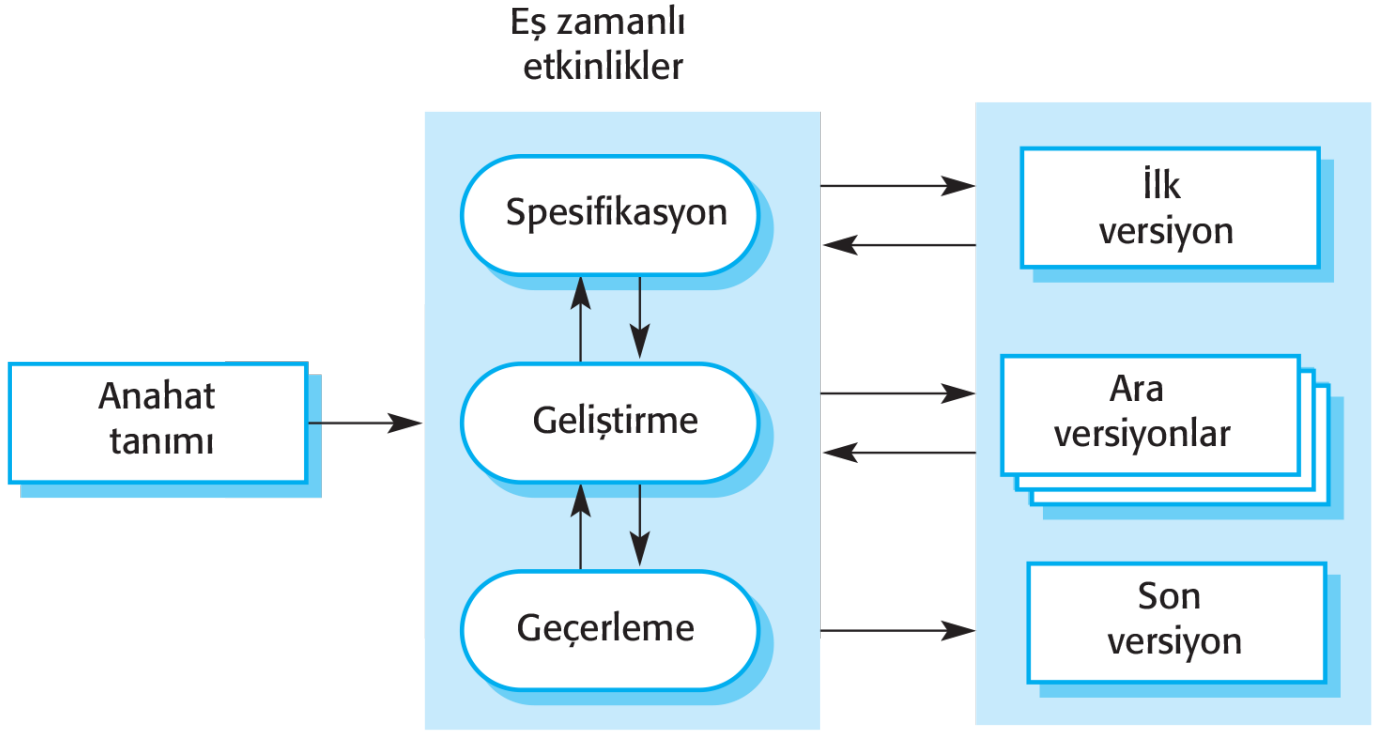
Yazılım mühendisliğinin öncülerinden olan Barry Boehm, risk-güdümlü bir artırımlı süreç modeli önermiştir. Bu süreç, bir etkinlikler dizisinden ziyade sarmal olarak gösterilmektedir (Boehm 1988).

Sarmaldaki her döngü, yazılım sürecindeki bir evreyi temsil eder. Örneğin, en içteki döngü sistem fizibilitesiyle, bir sonraki gereksinimlerin tanımıyla, ondan sonraki sistem tasarımıyla, vb. ilgili olabilir. Sarmal model, değişimden kaçınmayı değişime tolerans ile birleştirir. Değişikliklerin proje risklerinin bir sonucu olduğunu varsayarak bu riskleri azaltmak için belirgin risk yönetim etkinlikleri içerir.


Determine objectives, alternatives and constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Prototype 3
Prototype 2
Operational protoype
Proto-type 1
REVIEW
Requirements plan
Life-cycle plan
Simulations, models, benchmarks
Concept of Operation
S/W requirements
Product design
Detailed design
Development plan
Requirement validation
Code
Unit test
Integration and test plan
Design V&V
Integration test
Acceptance test
Plan next phase
Service
Develop, verify next-level product

# Artırımlı Geliştirme

Artırımlı geliştirme; başlangıç gerçekleştirimi, kullanıcı ve diğer kişilerden geri bildirim alma ve gereken sistem geliştirilene kadar pek çok versiyon yoluyla yazılımın evrim geçirmesi fikrine dayanır (Şekil 2.2). Spesifikasyon, geliştirme ve geçerleme etkinlikleri ayrı değil, birbirine geçmiş durumdadır ve etkinlikler arasında hızlı geri bildirimler söz konusudur.




**Şekil 2.2** Artırımlı geliştirme

## Artırımlı geliştirmeyle ilgili sorunlar

Artırımlı geliştirmenin pek çok avantajı olmasına karşın sorunsuz olduğu söylenemez. Karşılaşılan zorlukların başlıca nedeni, büyük kuruluşların zaman içerisinde evrim gösteren bürokratik prosedürlere sahip olması ve daha gayriresmî olan yinelemeli veya çevik süreçler ile bu prosedürler arasında bir uyumsuzluğun söz konusu olmasıdır.

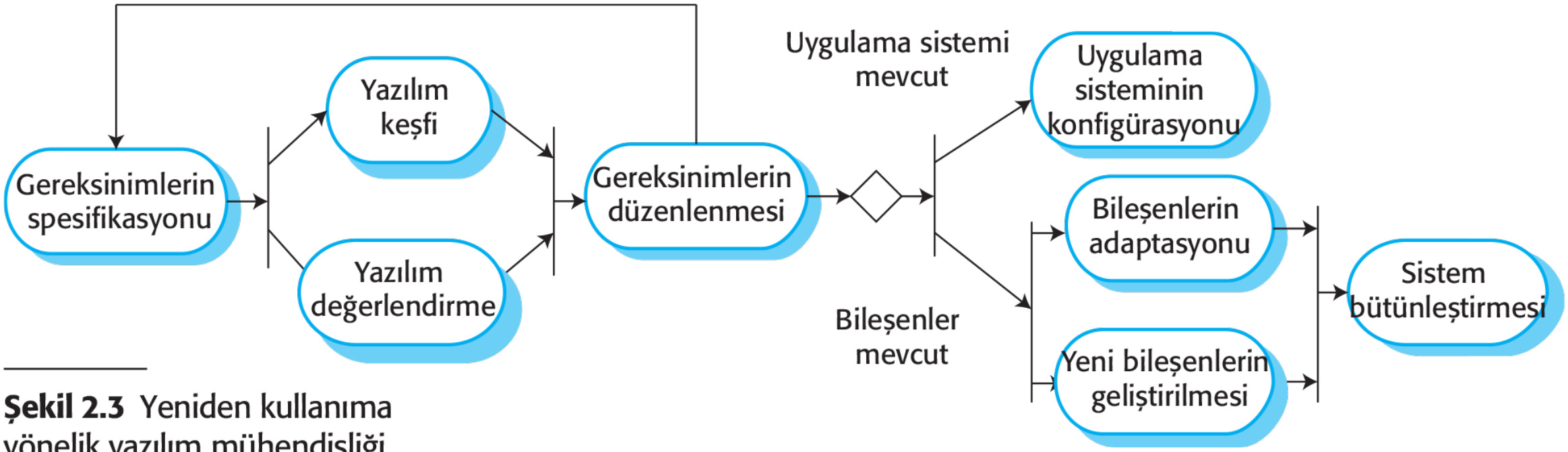
Bazen bu prosedürlerin iyi gerekçeleri bulunmaktadır. Örneğin, yazılımın dış yasal düzenlemelere (örn., Amerika Birleşik Devletleri'nde Sarbanes Oxley muhasebe kanunu) uygunluğunu sağlayan prosedürler mevcuttur. Bu prosedürleri değiştirmek mümkün olmadığından süreç ile uyuşmazlıklar kaçınılmaz olabilmektedir.

# Bütünleştirme ve Konfigürasyon

Yazılım projelerinin büyük çoğunluğunda, yazılımın bir kısmı yeniden kullanılmaktadır. Bu genellikle proje üzerinde çalışan kişilerin ihtiyaç duyulana benzer kodları bilmesi ya da araması sonucu gayriresmî şekilde meydana gelir. Bu kodlar bulunup gerekli kısımlarında değişiklik yapıldıktan sonra geliştirilen yeni kodlara entegre edilir.

Yazılım bileşenlerinin aşağıda verilen üç türü sık olarak yeniden kullanılmaktadır:

1. Belli bir ortamda kullanma amaçlı konfigüre edilen, tek başına kullanılabilen uygulama sistemleri. Bu genel-amaçlı sistemler pek çok özelliğe sahip olmakla birlikte bunların belirli bir uygulamada kullanılması için adaptasyonu gerekir.
2. Java Spring benzeri bir bileşen çerçevesiyle entegre edilmek üzere, bileşen ya da paket olarak geliştirilen nesne derlemeleri.
3. Servis standartlarına göre geliştirilen ve internet üzerinden uzaktan çağırma için kullanılan web servisleri.




**Şekil 2.3** Yeniden kullanıma yönelik yazılım mühendisliği

## Yazılım geliştirme araçları

Yazılım geliştirme araçları, yazılım mühendisliği süreç etkinliklerini desteklemek amacıyla kullanılan programlardır. Bunlar arasında gereksinim yönetim araçları, tasarım editörleri, yeniden üretim destek araçları, derleyiciler, hata ayıklayıcılar, hata izleyiciler ve sistem inşa araçları yer almaktadır.

Yazılım araçları, bazı etkinlikleri otomatikleştirerek ve geliştirilen yazılım hakkında bilgi sağlayarak sürece destek vermektedir. Örneğin:

- Gereksinim spesifikasyonunun ya da yazılım tasarımının parçası olan grafik sistem modellerinin geliştirilmesi
- Bu grafik modellerden kod üretimi
- Kullanıcı tarafından etkileşim halinde oluşturulan grafik arayüz tanımları kullanılarak kullanıcı arayüzünün oluşturulması
- Çalıştıran program hakkında bilgi sağlayarak hataların ayıklanması
- Programlama dilinin eski bir versiyonunda yazılmış programların otomatik olarak daha güncel bir versiyona çevrilmesi

Yazılım araçları, Etkileşimli Geliştirme Ortamları ya da EGO olarak adlandırılan çerçevede bir araya getirilebilir. Bu çerçeve, araçların birbiriyle iletişim kurmasını ve entegre biçimde çalışmasını kolaylaştıracak ortak bir olanaklar dizisi sunmaktadır.

# SÜREÇ ETKİNLİKLERİ

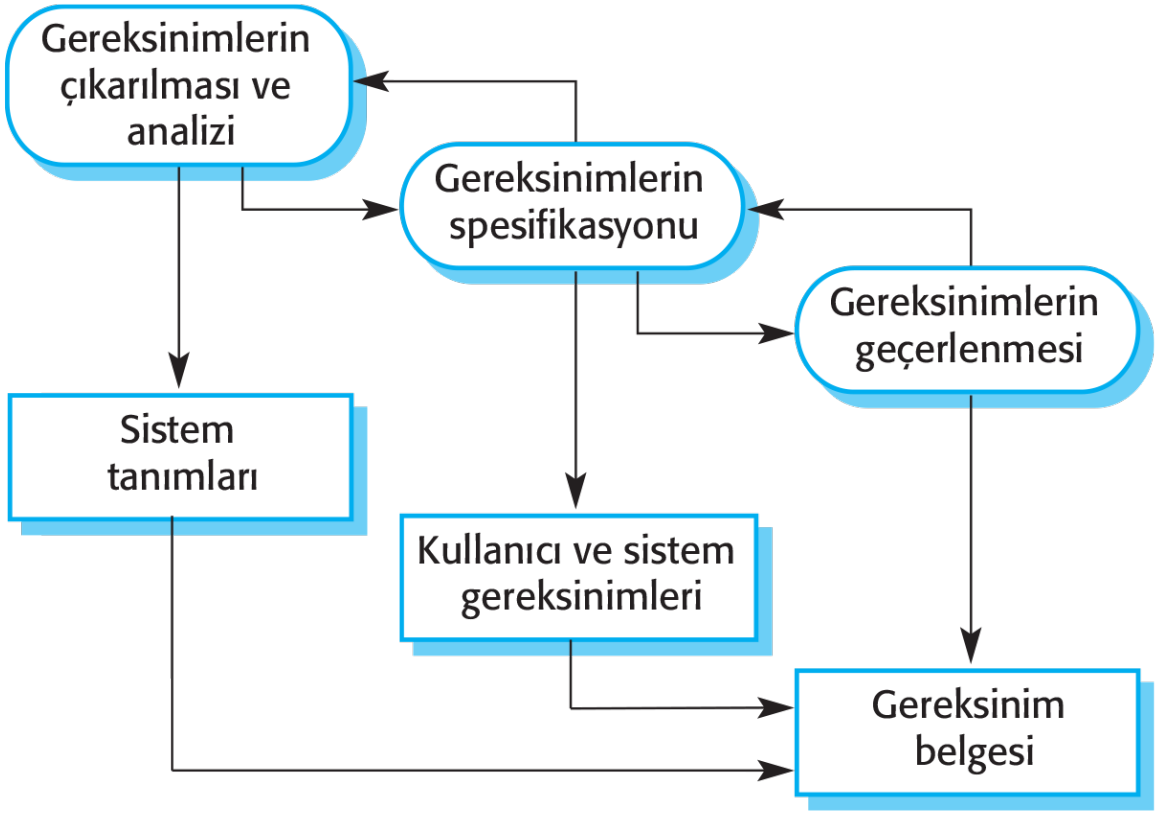
Gerçek yazılım süreçleri, genel hedefi bir yazılım sisteminin spesifikasyonu, tasarımı, gerçekleştirimi ve sınanması olan iç içe geçmiş teknik, işbirliğine dayalı ve yönetimsel etkinlikler dizisidir. Günümüzde süreçler genellikle araçlar tarafından desteklenmektedir.

Bir başka deyişle yazılım geliştiriciler, gereksinim yönetim sistemleri, tasarım model editörleri, program editörleri, otomatikleştirilmiş test araçları ve hata ayıklayıcıları gibi çeşitli yazılım araçlarından faydalanabilmektedir.

Dört temel süreç etkinliği olan spesifikasyon, geliştirme, geçerleme ve evrim, farklı geliştirme süreçlerinde farklı olarak düzenlenir. Bu etkinlikler çağlayan modelinde art ardayken, artırımlı geliştirmede iç içe geçmiş durumdadır.

# Yazılım Spesifikasyonu

Yazılım spesifikasyonu ya da gereksinim mühendisliği, sistemin sunması gereken servislerin anlaşılması ve tanımlanması, sistemin işletim ve geliştirimi üzerindeki kısıtlamaların belirlenmesi sürecidir. Bu aşamada yapılan hatalar sistem tasarımı ve gerçekleştiriminde daha sonra kaçınılmaz sorunlara yol açtığından gereksinim mühendisliği, yazılım sürecinde kritik öneme sahiptir.

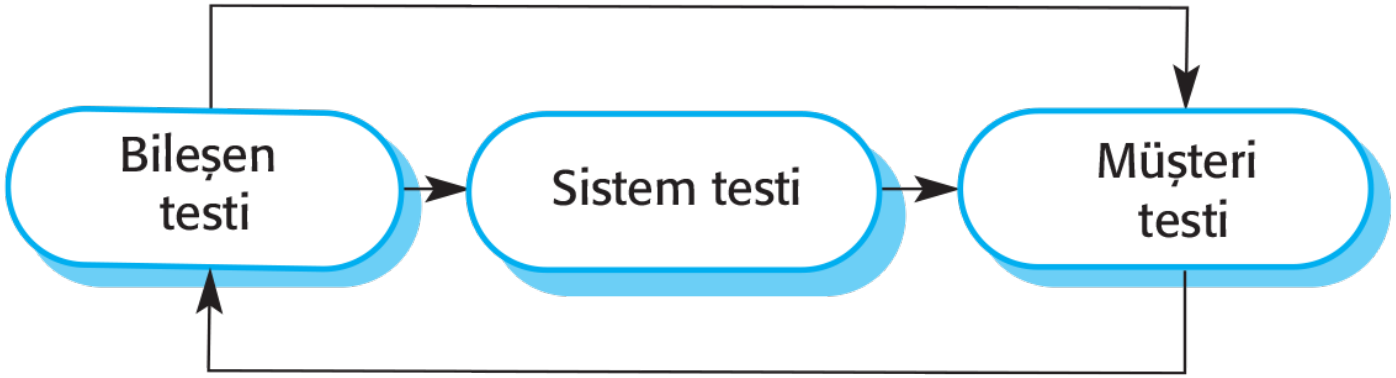



**Şekil 2.4** Gereksinim mühendisliği süreci

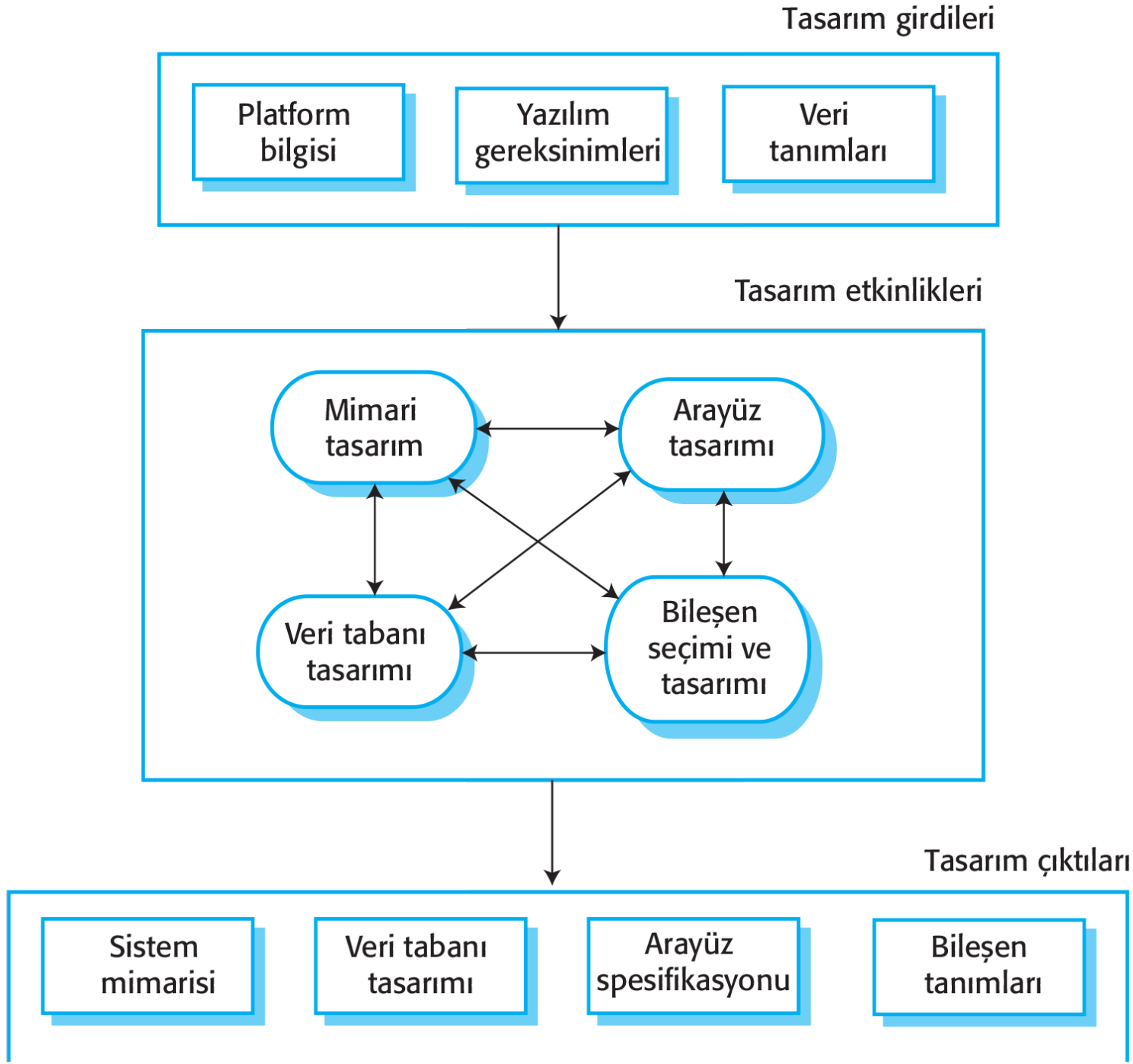
# Yazılım Tasarımı ve Gerçekleştirimi

Yazılım geliştirmenin gerçekleştirim aşaması, çalıştırılabilir bir sistemin müşteriye teslim amaçlı geliştirilmesi sürecidir. Bu süreç kimi zaman, yazılım tasarımı ve programlamayla ilgili ayrı etkinlikler içerir.

Ancak, çevik geliştirme yaklaşımı kullanıldığı durumlarda, tasarım ve gerçekleştirim iç içe geçmiş durumdadır ve bu süreçte, biçimsel tasarım belgeleri üretilmez. Doğal olarak, yazılım tasarımı hâlâ yapılmaktaysa da bu tasarım gayriresmî olarak beyaz tahta ve programcı defterlerine kaydedilir.



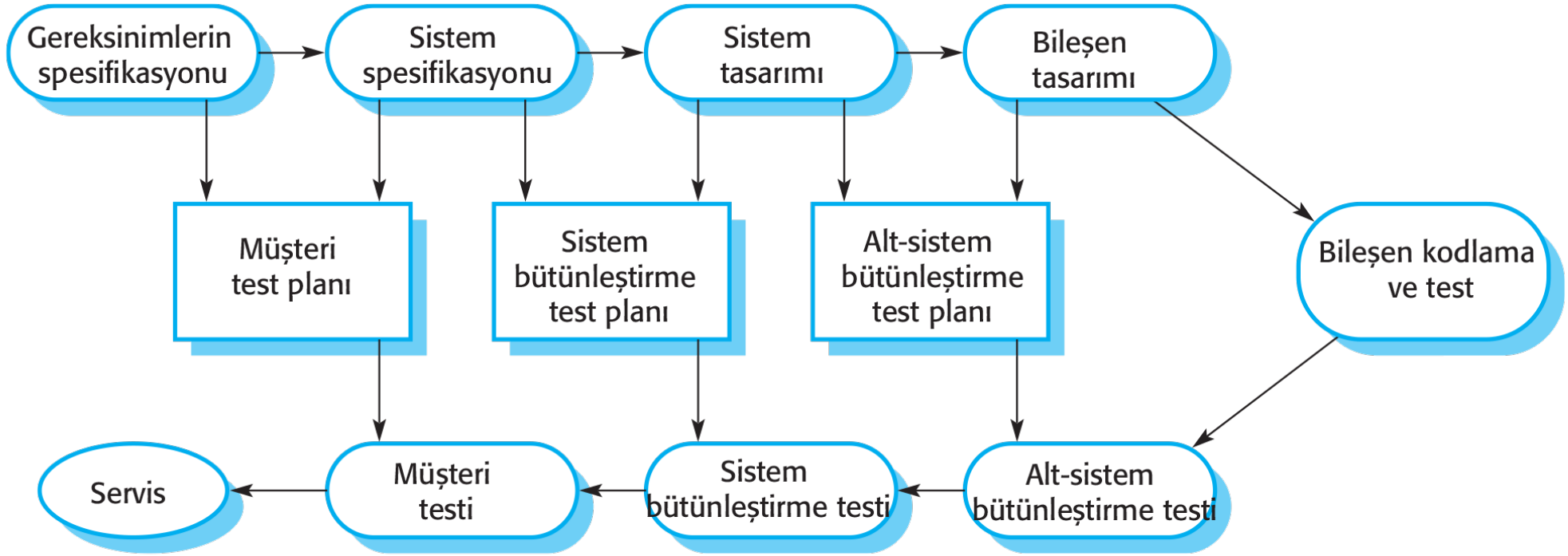

**Şekil 2.6** Test aşamaları




**Şekil 2.5** Tasarım sürecinin genel bir modeli

# Yazılım Geçerleme

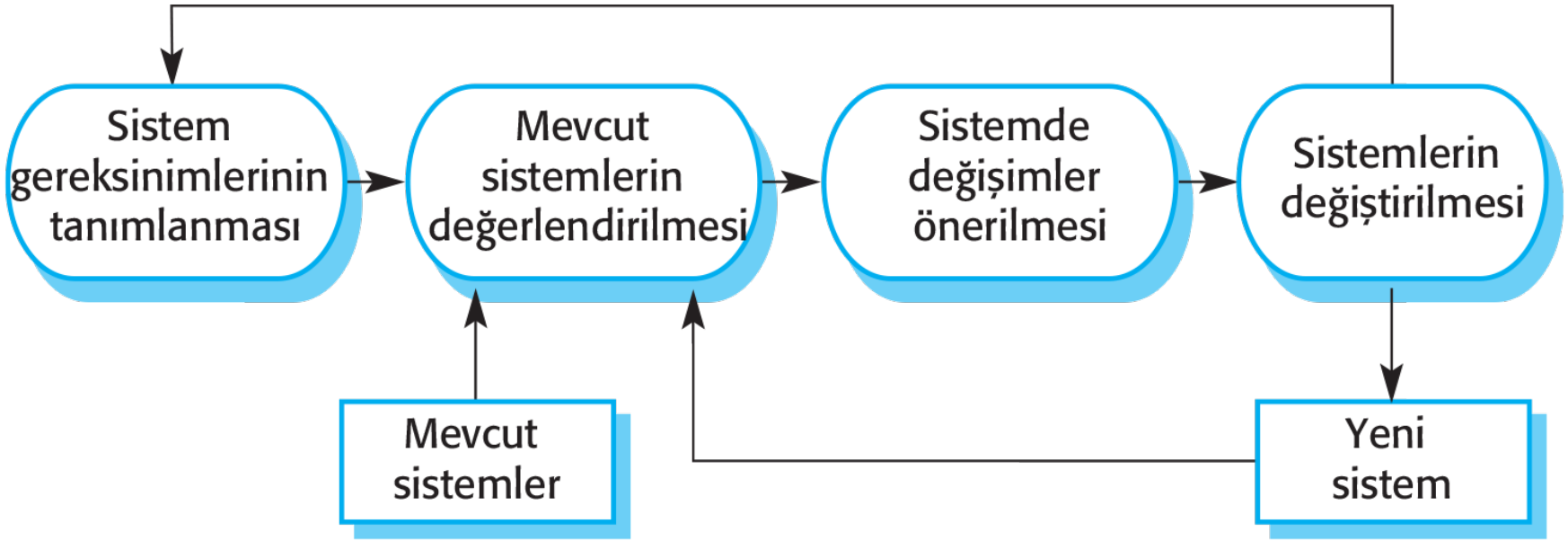
Yazılım geçerleme ya da genellikle kullanılan ifadesiyle doğrulama ve geçerleme (D & G), bir sistemin hem spesifikasyonuna uygun olduğunun hem de müşterisinin beklentilerini karşıladığının gösterilmesini amaçlar. Başlıca geçerleme tekniği, sistemin temsili test verileri kullanılarak çalıştırıldığı program testidir.




**Şekil 2.7** Plan-güdümlü yazılım sürecinde test aşamaları

# Yazılım Evrimi

Yazılımın esnekliği sayesinde her geçen gün daha fazla yazılım, büyük ve karmaşık sistemlere dahil edilmektedir. Donanım üretimiyle ilgili bir karar alındıktan sonra donanım tasarımında değişiklik yapmak oldukça masraflıdır. Ancak, sistem geliştirme esnasında ya da sonrasında herhangi bir zamanda, yazılımda değişiklik yapmak mümkündür. Kapsamlı değişiklikler bile, sistem donanımına yapılan benzer değişikliklerden önemli ölçüde az maliyete sahiptir.




**Şekil 2.8** Yazılım sisteminin evrimi

# DEĞİŞİMLE BAŞ ETME

Büyük yazılım projelerinde değişim kaçınılmazdır. İşletmelerdeki dış baskı, rekabet ve değişen yönetim öncelikleri karşısında sistem gereksinimleri değişmektedir. Yeni teknolojiler ortaya çıktıkça tasarım ve gerçekleştirmeye yönelik yeni yaklaşımlar mümkün hale gelmektedir. Bu nedenle, hangi yazılım süreç modeli kullanılırsa kullanılsın, geliştirilen yazılımı değişime uydurabilmesi büyük önem taşımaktadır.

Yeniden çalışmanın maliyetini düşürmede aşağıdaki iki yaklaşım kullanılabilir:

1. *Değişimi öngörme*

2. *Değişime tolerans*

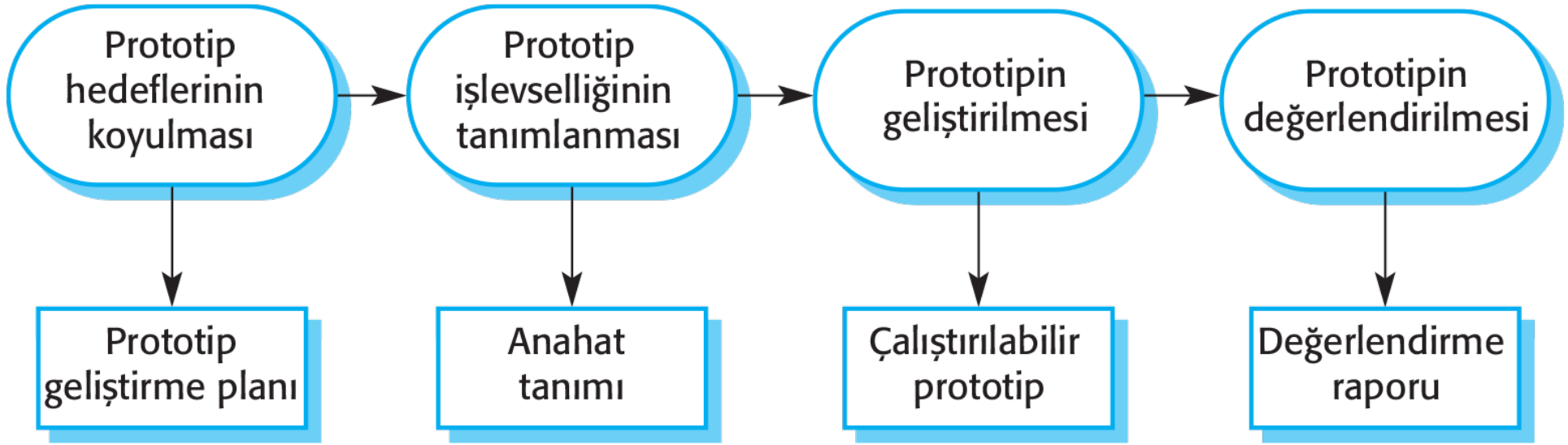
# Prototip Geliştirme

Prototip, kavramların gösterilmesi, tasarım seçeneklerinin değerlendirilmesi ve sorunlar ile olası çözümleri hakkında daha fazla bilgi edinilmesi amacıyla kullanılan yazılım sisteminin erken bir versiyonudur.

Yazılımın prototipi, yazılım geliştirme sürecinde gerek duyulabilecek değişimleri öngörmeye yardımcı olmak amacıyla kullanılabilir:

1. Gereksinim mühendisliği sürecinde prototip, sistem gereksinimlerinin çıkarılması ve geçerlenmesinde yardımcı olur.

2. Sistem tasarım sürecinde prototip, yazılım çözümlerinin araştırılması ve sistem için bir kullanıcı arayüzü geliştirilmesi için kullanılabilir.




**Şekil 2.9** Prototip geliştirme

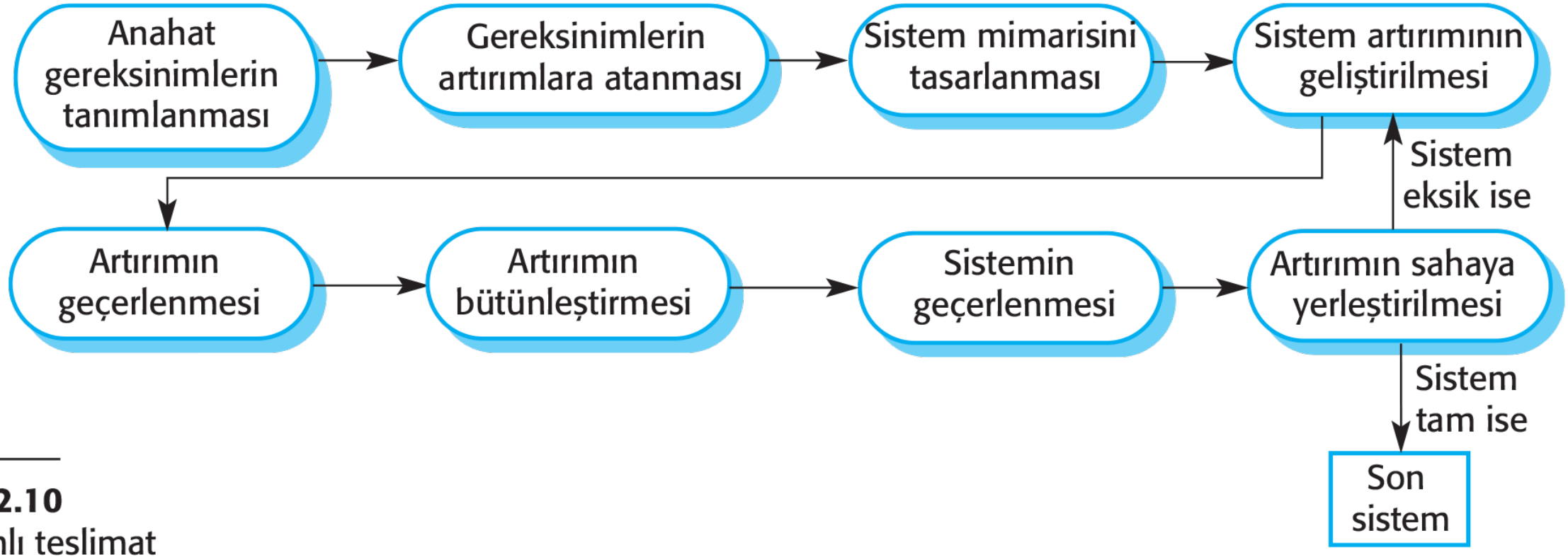
# Artırımlı Teslimat

Artırımlı teslimatın avantajları şu şekildedir:

1. Müşteriler, erken artırımları prototip olarak kullanarak sonraki sistem artırımlarında gereksinimlerin daha bilgili olarak belirlenmesi için deneyim kazanırlar. Prototiplerin aksine, artırımlar gerçek sistemin parçası olduğundan bütün sistem hazır olduğunda, yeniden öğrenme ihtiyacı söz konusu değildir.

2. Müşterilerin sistemden değer elde edebilmek için, tüm sistemin teslimatını beklemeleri gerekmez. İlk artırım en kritik gereksinimleri karşıladığından müşteriler, yazılımı hemen kullanmaya başlayabilir.

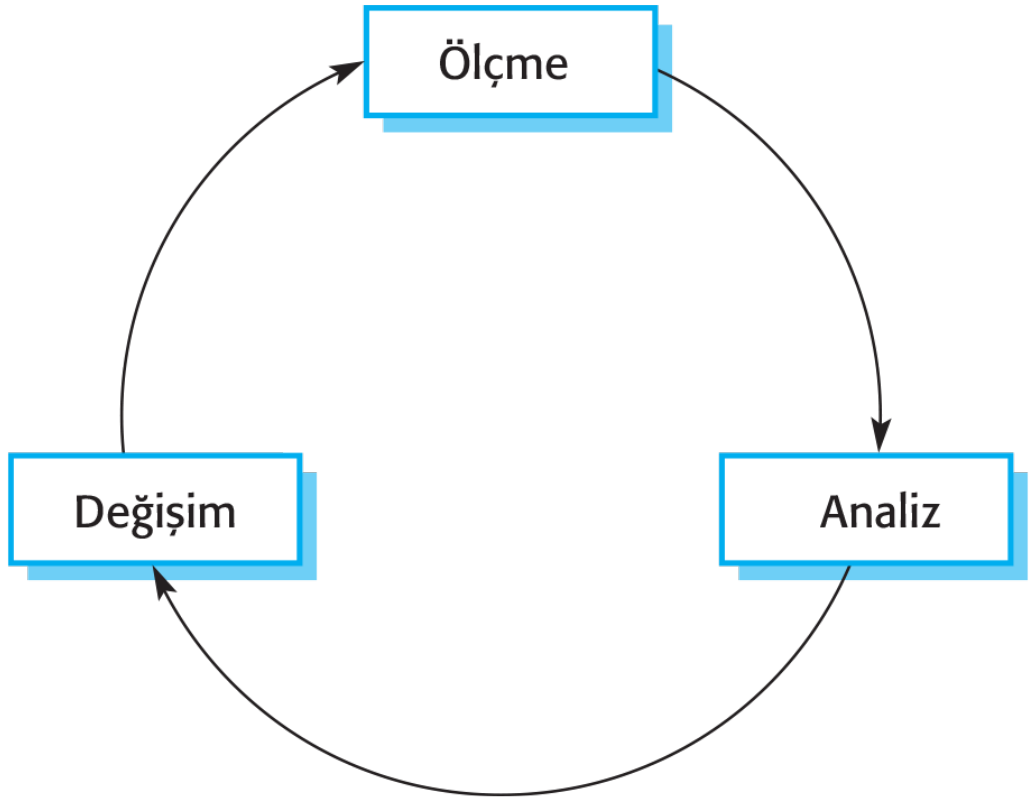
3. Bu süreç, artırımlı geliştirmenin sisteme dahil edilecek değişiklikleri nispeten daha kolay hale getirmesi faydasına sahiptir.



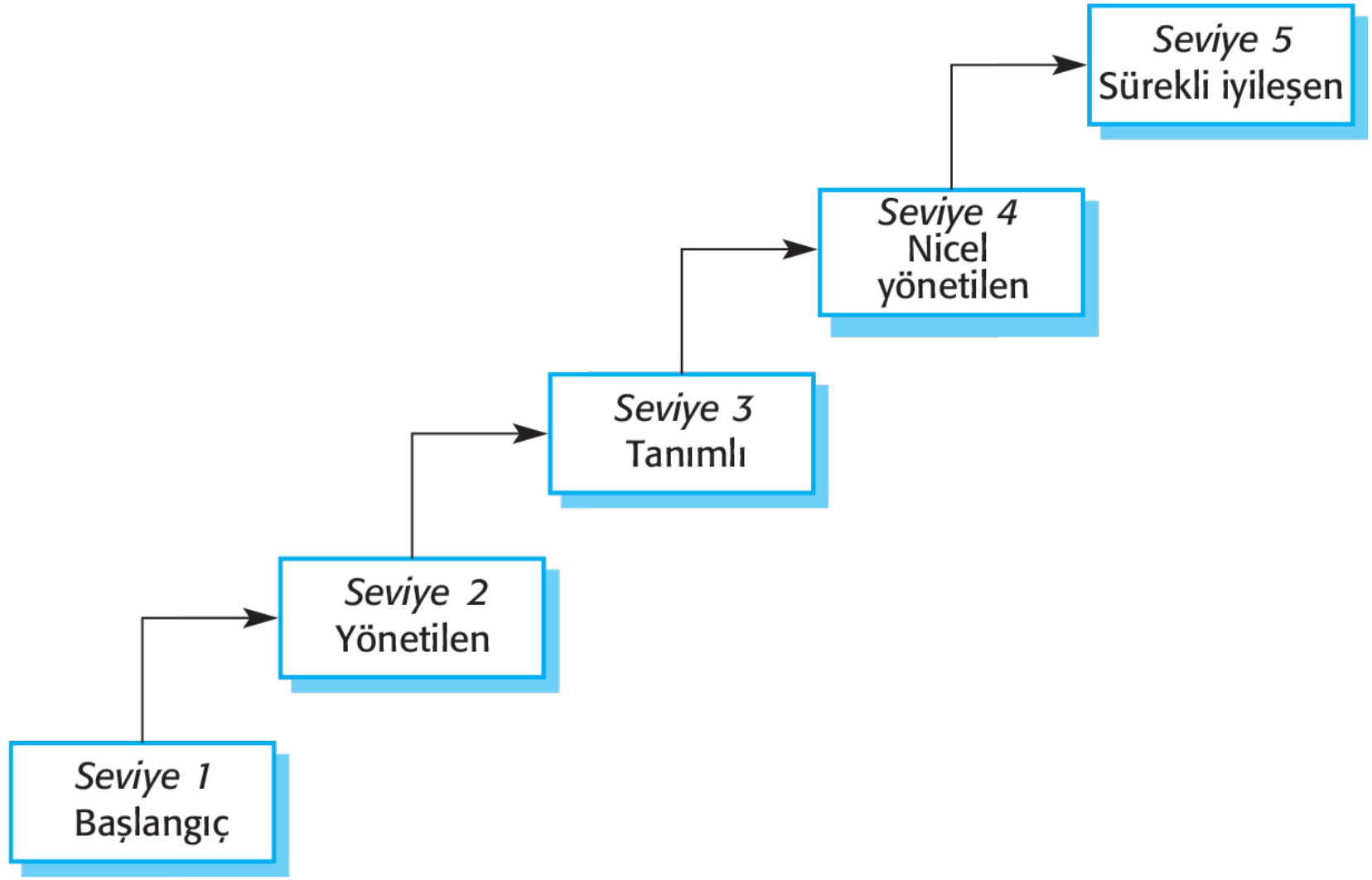

**Şekil 2.10**
Artırımlı teslimat

# SÜREÇ İYİLEŞTİRME

Günümüzde, giderek daralan sürelerde teslim edilmesi beklenen, daha ucuz ve daha iyi yazılıma yönelik sürekli bir talep söz konusudur. Bu nedenle yazılım şirketleri, yazılım süreç iyileştirmeyi yazılımlarının kalitesini arttırma, maliyetleri düşürme ya da geliştirme süreçlerini hızlandırmanın bir yolu olarak görmeye başlamıştır.




**Şekil 2.11** Süreç iyileştirme döngüsü




**Şekil 2.12** Yetenek olgunluk seviyeleri