

Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

BZ 313 Yazılım Mühendisliği

3. Yazılım Geliştirme Süreçleri

Aşamalar

Ders 2’de bazı **işlem adımlarından** söz edilmiştir:

- **Gereksinimler:** Müşterinin ne istediğini belirleme
- **Kullanıcı arayüz tasarımı:** Ekranların nasıl görüneceğinin tasarımı
- **Sistem tasarımı:** Yazılımın iç yapısı
- **Program geliştirme (Tasarım ve Kodlama):** Kod yazma ve test etme
- **Kabul ve yayınlama:** Müşteriye teslim

Her yazılım projesi bu temel adımları bir şekilde içerir, ancak:

- Adımlar resmi veya gayri resmi olabilir.
- Adımlar çeşitli sıralarda gerçekleştirilebilir.

Bir **yazılım geliştirme süreci** veya **metodolojisi**, bir yazılım sistemi oluşturmak için bu adımları birleştirmenin sistematik bir yoludur.

Yazılım Geliştirme Süreçleri

Başlıca alternatifler

Bu derste, yazılım geliştirme süreçlerinin dört kategorisini inceleyeceğiz:

- **Şelale (Waterfall) modeli:**

Su gibi yukarıdan aşağıya akan sıralı bir süreçtir. Bir sonrakine başlamadan önce her işlem adımını tamamlanır.

- **Yinelemeli iyileştirme (Iterative refinement):**

Aynı işlemler tekrar tekrar yapılır. Kaba bir sistem oluşturmak için tüm adımları hızlı bir şekilde uygulanır, ardından sistemi geliştirmek için bu adımlar tekrarlanır.

- **Spiral:**

Döngüseldir ve her döngüde sistem büyür. Yeni ve güncelleştirilmiş bileşenlerin tamamlandıkça gelişmekte olan sisteme eklendiği yinelemeli iyileştirmenin farklı bir çeşididir.

- **Çevik (Agile) geliştirme:**

Küçük yazılım geliştirme artımları, her biri dağıtılabilir kod oluşturan bir dizi sprintte (koşu) geliştirilir.

Yaklaşım	Süre	Esneklik	Dokümantasyon	En Uygun Durum
Şelale	Uzun	Düşük	Çok	Gereksinimler net
Yinelemeli	Orta	Orta	Orta	Gereksinimler belirsiz
Spiral	Uzun	Orta	Çok	Büyük sistemler
Çevik	Kısa	Yüksek	Az	Hızlı değişim

Gerçek Dünya Örnekleri:

- Şelale: Bankacılık sistemi (uzun, karmaşık, değişmez)
- Yinelemeli: E-ticaret sitesi tasarımı (müşteri geri bildirimi önemli)
- Spiral: İşletim sistemi geliştirme (büyük, entegre sistem)
- Çevik: Mobil uygulama (hızla değişen pazar, sık güncelleme)

Ağır (Heavyweight) Yazılım Geliştirme

- **Heavyweight süreçlerde** amaç, her adımı eksiksiz tamamlamak ve sonradan minimum düzeyde değişiklik yapmaktır. Her aşama, bir sonrakine geçmeden önce ayrıntılı olarak dokümente edilir.

Örnek: Modifiye Edilmiş Şelale Modeli (Modified Waterfall Model)

- *Her adım için detaylı doküman hazırlanır.*
- *Örnek dokümanlar:*
 - *Gereksinim spesifikasyonu (100+ sayfa)*
 - *Teknik tasarım dokümanı (50+ sayfa)*
 - *Test planı (30+ sayfa)*
- *Her şey yazılı, takip edilebilir.*
- *Dezavantaj: Yavaş, bürokrasi*

Hafif (Lightweight) Yazılım Geliştirme

- **Lightweight süreçlerde** ise geliştirme ekibi en az seviyede ara doküman üretir. Sürecin ilerledikçe değişikliklere açık olacağı kabul edilir. Yalnızca nihai sistemin belgelenmesi beklenir.

Örnek: Çevik (Agile) Yazılım Geliştirme

- *Minimum doküman, maksimum kod*
- *Örnek dokümanlar:*
 - *User story'ler (2-3 cümle)*
 - *Kod yorumları*
 - *README dosyası*
- *Avantaj: Hızlı ve esnek*
- *Dezavantaj: Bilgi kaybı riski*

Senaryo: E-Ticaret Sitesi

Heavyweight Yaklaşım

- 6 ay: Gereksinim analizi ve dokümantasyon
- 3 ay: Tasarım ve mimari dokümanı
- 6 ay: Kodlama
- 2 ay: Test ve doküman güncellemesi

👉 **Toplam: 17 ay**

Lightweight Yaklaşım

- 1 hafta: Temel özellikler listesi
- 2 hafta: İlk versiyon kodlama
- Her hafta: Yeni özellik ekleme ve test
- 3 ayda: Market-ready ürün

👉 **Toplam: 3 ay**

Heavyweight ve Lightweight Metodolojiler

Heavyweight	↔	Lightweight
Süreçler ve Araçlar	↔	Bireyler ve etkileşimler
Dokümantasyon	↔	Çalışan uygulama
Plana sadık kalmak	↔	Değişime karşılık verme
Sözleşme pazarlığı	↔	Müşteri işbirliği

Çevik Yazılım Geliştirme Manifestosu'na dayanmaktadır:

<http://agilemanifesto.org/>

Heavyweight ve Lightweight Metodolojiler

1. Süreçler ve Araçlar ↔ Bireyler ve Etkileşimler

- Heavyweight: JIRA, Monday.com gibi proje yönetim araçları olmadan ilerlenemez.
- Lightweight: WhatsApp mesajı ya da kısa bir toplantı bile yeterlidir.

2. Dokümantasyon ↔ Çalışan Uygulama

- Heavyweight: Önce yüzlerce sayfa doküman hazırlanır, sonra kodlama başlar.
- Lightweight: Önce çalışan bir demo geliştirilir, gerekiyorsa daha sonra doküman eklenir.

3. Plana Sadık Kalmak ↔ Değişime Karşılık Verme

- Heavyweight: Planda değişiklik yapmak için uzun bürokratik süreçler gerekir.
- Lightweight: Pazartesi yapılan plan, çarşamba gününe değiştirilebilir.

4. Sözleşme Pazarlığı ↔ Müşteri İşbirliği

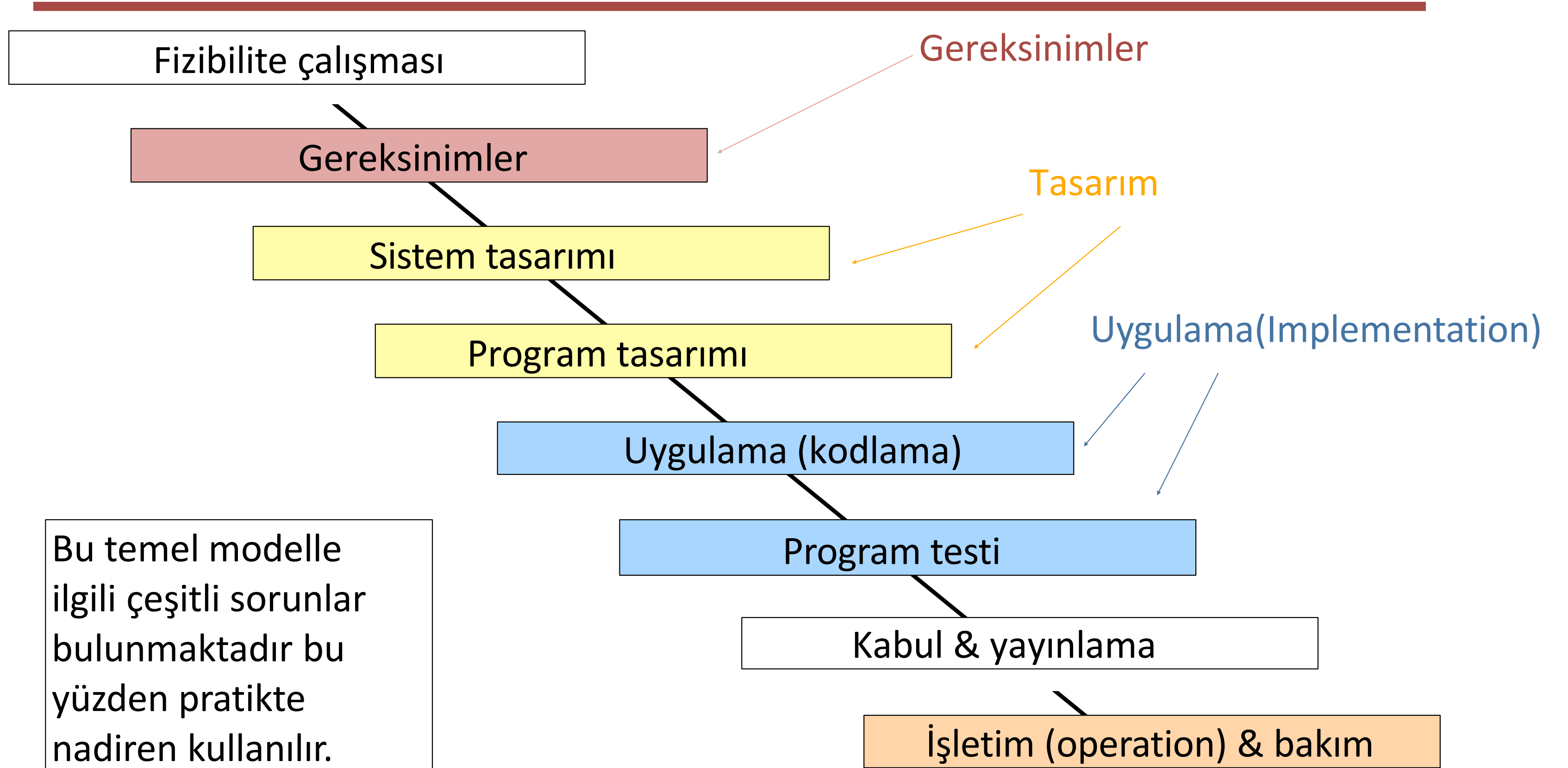
- Heavyweight: “Bu sözleşmede yok, ek ücret talep ediyoruz.” yaklaşımı hakimdir.
- Lightweight: “Müşteri memnuniyeti için bu özelliği ekleyelim.” anlayışı öne çıkar.

Yazılım Mühendisliğinin Doğuşu (1970'ler)

Yazılım mühendisliği, bir disiplin olarak **1970'lerin başında ortaya çıktı.**

- O dönemde bilgisayar sistemleri, daha önce manuel yapılan işlemlerin dijital dönüşümüne dayanıyordu.
(Örn: bordro, faturalandırma, havayolu rezervasyonları)
- Gereksinimler genellikle **önceden belirli ve nettir.**
- Sistemler aynı mimariyi izliyordu, sadece ana dosya güncelleniyordu (sistemin mimarisi aynı kalırken, sadece kullanılan veritabanı/dosya yapısının değiştirilmesi) → **tasarım daha kolay anlaşıldı.**
- Kodlama ise modern diller ve araçlar olmadığından oldukça **zorlu ve sıkıcıydı.** Bu nedenle **kodlamadan önce kapsamlı bir tasarım** yapmak büyük önem taşıyordu.
- Tüm bu faktörler, yazılım geliştirmede **Şelale Modeli'nin doğmasına** yol açtı.

Şelale Modeli



Şelale Modeli Üzerine Tartışma

Şelale modeli, her işlem adımının tam dokümantasyonuna sahip **heavyweight** bir süreçtir.

Avantajlar:

- Görevlerin ayrışması
- Süreç görünürlüğü
- Her adımda kalite kontrol
- Her adımda maliyet izleme

Dezavantajlar:

Uygulamada bu süreç, her aşamada, önceki aşamaların tamamen gözden geçirilmesini gerektirir. Böylece önceki aşamaların yeni bir bakış açısıyla ortaya koyulmasına neden olmaktadır.

Şelale Modeli yeterince esnek değildir.

Şelale Modeli Üzerine Tartışma

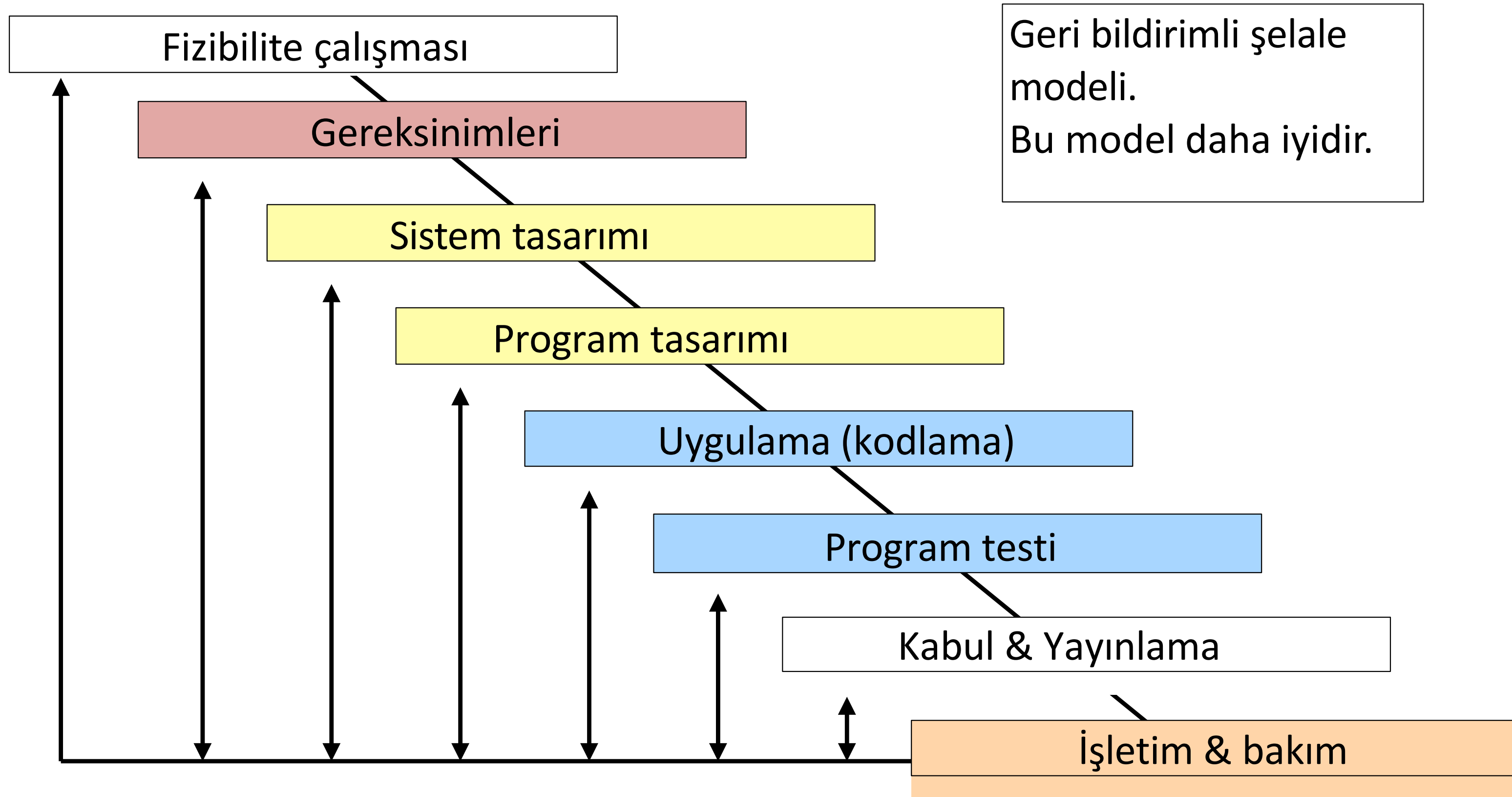
Saf bir sıralı modelin uygulanması mümkün değildir.

Plan, bir çeşit yinelemeye izin vermelidir.

Örnekler:

- Bir fizibilite çalışması ile gereksinimlerin ön çalışması ve geçici bir tasarımı gerçekleştirilmeden **bütçe ve program** oluşturulamaz.
- Ayrıntılı tasarım ve uygulama, gereksinim spesifikasyonundaki boşlukları/eksiklikleri ortaya çıkarır.
- Geliştirme sırasında gereksinimler ve/veya teknolojiler değişebilir.

Modifiye Şelale Modeli



Modifiye Şelale Modeli Ne Zaman Kullanılır?

Modifiye Şelale Modeli, gereksinimler iyi anlaşıldığında ve tasarım basit/açık olduğunda oldukça faydalıdır, örneğin,

- Gereksinimlerin iyi anlaşıldığı manuel bir veri işleme sisteminin dönüştürülmesi (örneğin, elektrik faturaları).
- İşlevselliği önceki bir ürüne oldukça benzeyen bir sistemin yeni sürümü (örneğin, Bir araba için otomatik fren sistemi).
- Bazı bileşenlerin açıkça tanımlanmış gereksinimlere sahip olduğu ve sistemin geri kalanından açıkça ayrıldığı büyük bir sistemin bölümleri.

Yinelemeli İyileştirme (Iterative Refinement)

Kavram

- Gereksinimler genellikle ilk başta net değildir.
- Prototipler sayesinde kullanıcı geri bildirimi alınarak sistem adım adım geliştirilir.

Süreç (Process)

- Prototip oluşturma
 - Hızlı ve temel bir sistem taslağı hazırlanır.
- Gözden geçirme ve test
 - Kullanıcılarla test edilir, geri bildirim toplanır.
- İyileştirme
 - Geri bildirimlere göre sistem geliştirilir.
- Yineleme
 - Bu döngü sistem istenen kaliteye ulaşana kadar tekrarlanır.

Yinelemeli İyileştirme: Örnek Senaryo

Problem: Grafik paket eklenmesi gereken bir programlama ortamı

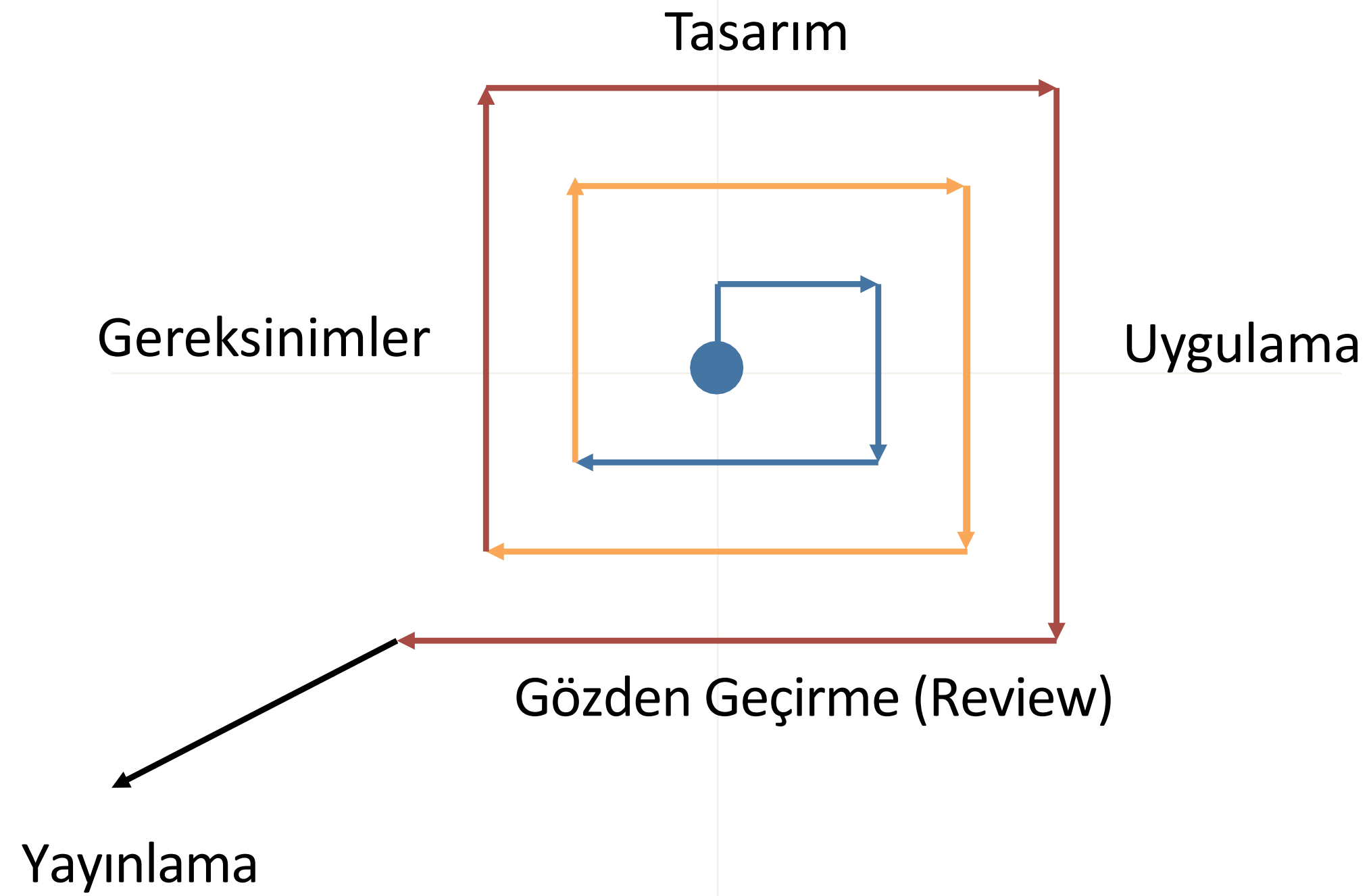
Gereksinimler

Client, farklı nesneler arasındaki koordinatların nasıl yönetileceğine ilişkin bazı gereksinimlerden emin değil.

Süreç

- Prototip geliştirme
 - Bir ön işlemci ve ön çalışma zamanı paketi ile prototip sürümü oluşturulur.
- Kullanıcı testi
 - Prototip kullanıcılarla test edilir, geri bildirim toplanır.
- İyileştirme
 - Geri bildirimlere göre sistemde değişiklik yapılır.
- Son aşama
 - Ön işlemci güncellenir, geçici yamalar kaldırılır, nihai ürün oluşturulur.

Yinelemeli İyileştirme



Yinelemeli İyileştirme Üzerine Tartışma

- **Medium weight** bir süreçtir (belgelere kısmen önem verir).
- Amaç: **Müşterinin ve kullanıcıların sistemi erken aşamada görmesini** sağlamaktır.
- Kullanılan teknikler:
 - Kullanıcı arayüzü **mock-up'ları**
 - **Atılabilir prototipler**
 - **Dummy modüller** (gerçek işlevi olmayan sahte modüller)
 - **Hızlı prototipleme**
 - **Ardışık iyileştirme**

Önemli Not:

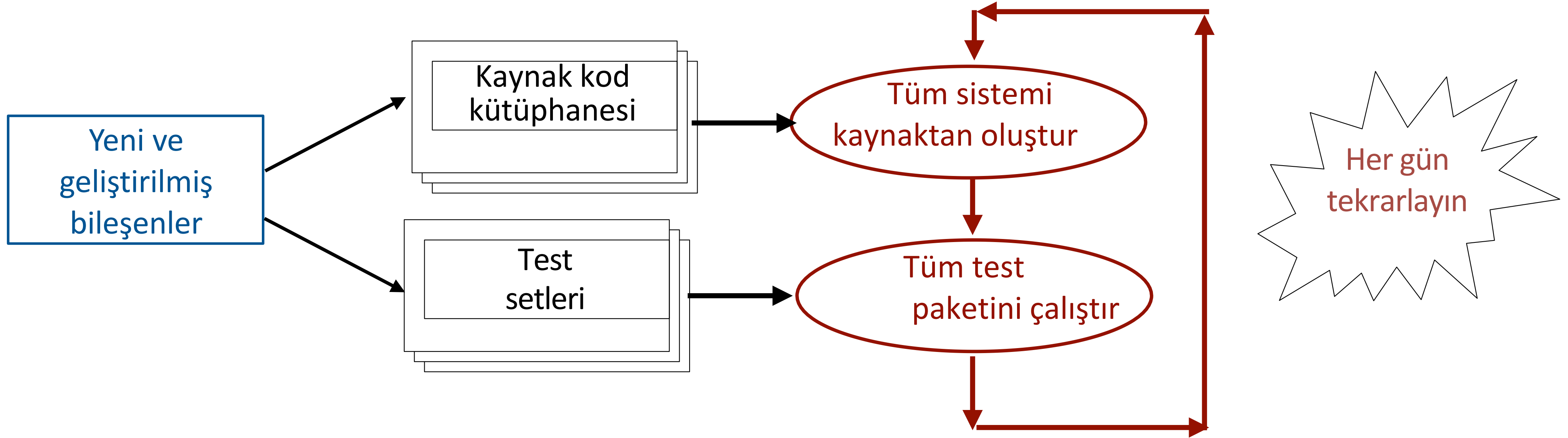
Müşterinin değerlendirmesi için **hızlı bir uygulama** geliştirin, fakat **bu sürümü yayınlamayın**.

Spiral Geliştirme

Örnek: Microsoft'un işletim sisteminin yeni sürümünü geliştirmesi

- Spiral geliştirmede her zaman **tamamen test edilmiş bir sistem** vardır, ancak başlangıçta işlevsellik eksiktir.
- Eksik bileşenler için **dummy stub'lar** kullanılarak, nihai ürüne benzeyen temel bir sistem kurulur.
- Her tamamlanan bileşen için **kapsamlı bir test paketi** hazırlanır.
- Geliştirme ekipleri, yeni veya geliştirilmiş bileşenler üretir → bu bileşenler **kaynak kod kütüphanesine eklenir**.
- Günlük döngü:
 - Test ekipleri tüm sistemi kaynak kütüphanesinden çeker
 - Sistemi ayağa kaldırır
 - **Tüm test setini** çalıştırır

Spiral Geliştirme



Spiral Geliştirme Üzerine Tartışma

Neden Kullanılır?

- Genel mimari iyi anlaşılmıştır.
- Büyük bileşenler bağımsız şekilde geliştirilebilir ve test edilebilir.
- Kapsamlı otomatik test setleri oluşturmak, maliyetli olmasına rağmen büyük sistemlerde katma değer sağlar.

Zorluklar:

- Mimari üzerinde büyük değişiklikler yapmak zordur.
- Birçok bileşeni etkileyen değişiklikleri uygulamak karmaşık ve risklidir.

Çevik Yöntemler

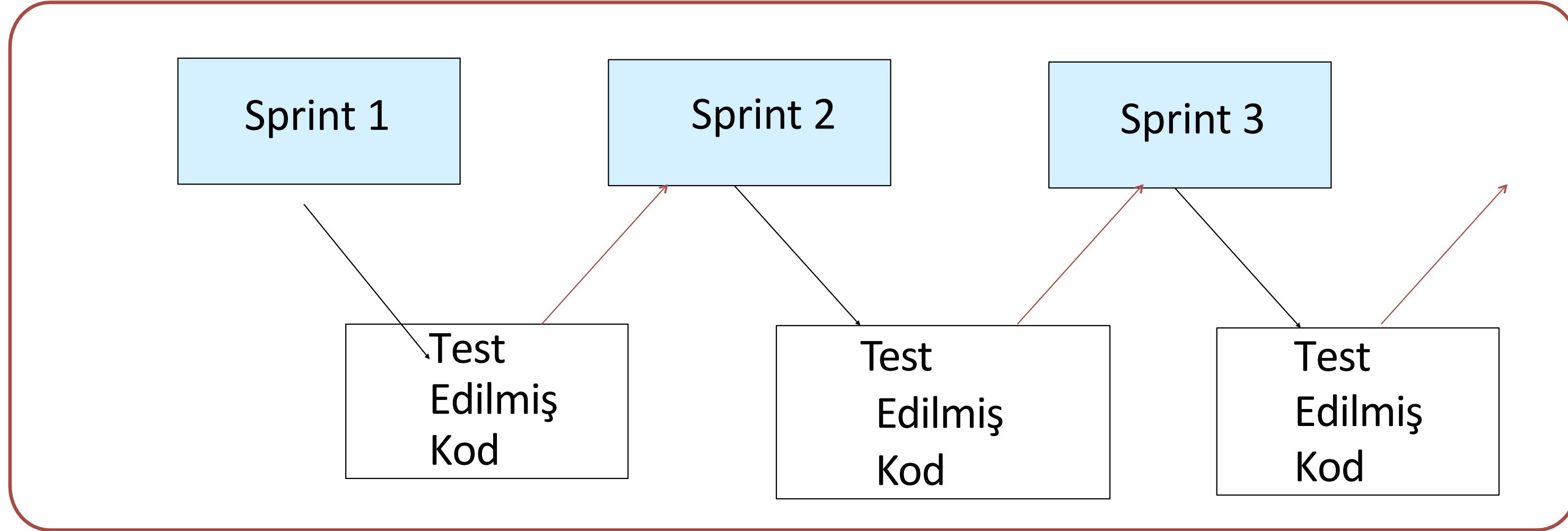
Genel İlkeler:

- Büyük projeler, **sprint** adı verilen küçük parçalara bölünür.
- Geliştirme, **4–9 kişilik küçük ekipler** tarafından yapılır.
- Sprintler genellikle **2–4 haftalık sabit zaman dilimleri** halinde planlanır.
- Her sprint:
 - Gereksinimler
 - Tasarım
 - Kodlama
 - Testadımlarını kapsar.
- Sprint sonunda: **üretime hazır, test edilmiş bir kod** elde edilir.




Not:

- Çevik geliştirme, minimum dokümantasyona sahip **lightweight** bir süreçtir.
- Ancak nihai sürümde, gelecekteki bakım için **tam dokümantasyon** hazırlanmalıdır.

Çevik Geliştirme



Her sprintten sonra kod şöyle olabilir:

-  **Yayımlanabilir** (Orijinal Çevik Yaklaşım)
-  **Sonraki sürüme** aktarılabilir ve diğer sprint kodlarıyla birleştirilebilir
-  **Daha büyük bir kod tabanına** entegre edilebilir (Spiral Geliştirme yaklaşımı)

Çevik: Kod Yayınlama

Çevik sürümler

- Çevik süreçlerin **orijinal yaklaşımında**, her sprint **yayımlanan bir kod** ile sona erer.
Ancak bu pratikte **nadiren mümkündür**.

- Bu derste:

Sprint = Üretime (production) hazır kalitede kod anlamına gelecektir.

Not:


Bazı kişiler, **her kısa aktiviteyi “sprint”** olarak adlandırır. Bu, **çevik yaklaşımın ruhuna aykırıdır**.

Çevik Geliştirme: Rework (Yeniden Ele Alma)

Çevik Geliştirmenin Zorluğu

- Çevik yaklaşım, özellikle **yerleşik bir mimari** üzerinde sistem geliştirme veya sürekli iyileştirme için uygundur.
- Üst düzey bir ekip, **genel mimariyi belirler** ve sprintleri koordine eder.

Rework (Yeniden Ele Alma)

- Çevik geliştirmede, gereksinimler ve tasarım **kademeli olarak ortaya çıkar.**
- Bu nedenle bazı sprintlerin çıktılarının **yeniden ele alınması kaçınılmazdır.**
- Hatta bazen bu durum, **test edilmiş ve yayımlanmış kodda değişiklik** yapılmasını gerektirir.
-  Yazılım geliştirmede oldukça “**alışılmadık**” bir durumdur.
- Eğer yeniden ele alma çok fazla oluyorsa:
 - Her bileşeni baştan düzeltmek yerine
 - **Yinelemeli iyileştirme** kullanarak rework miktarını en aza indirmek daha verimlidir.

Karma Süreçler

Uygulamada, büyük projeler genellikle birden fazla sürecin birleşimini kullanır.

Örnekler

- Gereksinimleri **iyi anlaşılmış** bir projede:
 - Başlangıç → **Değiştirilmiş Şelale** ile gereksinim ve sistem tasarımı
 - Sonraki aşamalar → **Çevik sprintler** ile geliştirme
- Gereksinimleri **belirsiz** bir projede:
 - Başlangıç → **Yinelemeli İyileştirme** ile gereksinimleri netleştirme
 - Sonraki aşama → **Modifiye Şelale** ile son sürüm oluşturma
- **Spiral Geliştirme** içinde, yeni bileşenler **sprintler** halinde geliştirilebilir.

Not:

- **Kullanıcı arayüzleri**, mutlaka kullanıcılarla test edilmelidir.
- Bu, hangi süreç kullanılırsa kullanılsın, **yinelemeli geliştirmeyi zorunlu kılar.**

Karma Süreçler: Aşamalı (phased) Geliştirme

Büyük projeler, **iki veya daha fazla aşamaya** bölünebilir.

Aşama 1: Sistemin temel işlevselliklerini içeren ilk sürüm hızla üretime alınır.

Sonraki aşamalar: Kullanıcıların deneyimlerinden elde edilen geri bildirimlere göre şekillenir.

Avantajları

- Yatırımın geri dönüşü **erken başlar**.
- Gereksinimler, sonraki aşamalarda **daha net anlaşılır**.
- Maliyetler, **daha uzun vadeye yayılabilir**.

Yazılım Süreçleri Hakkında Gözlemler

- Tamamlanan projeler, mutlaka **tüm temel süreç adımlarını** içermelidir.
- Ancak geliştirme süreci, çoğu zaman sadece **kısmen evrimsel** ilerler.

Riskleri Azaltma Yöntemleri:

- Temel bileşenlerin **prototipini** oluşturmak
- **Sık sık yayınlama** yapmak veya büyük projeleri **aşamalara bölmek**
- Kullanıcı ve müşterilerle **erken ve tekrarlı testler** yapmak
- **Görünür ve takip edilebilir** bir yazılım süreci kullanmak

Uygun Yazılım Sürecini Seçme

Yazılım geliştirme sürecindeki değişiklikler oldukça maliyetlidir.

- **Gereksinimler belirsiz veya değişken**
 - Süreç: **Yinelemeli İyileştirme, Çevik Sprintler, Aşamalı Uygulama (Phased)**
 - Örnek Proje: Yeni kurulan bir **sosyal medya uygulaması** (özellikler kullanıcı geri bildirimiyle şekillenir).
- **Büyük ve birbirine bağlı bileşenler**
 - Süreç: **Modifiye Şelale Modeli**
 - Örnek Proje: **Banka çekirdek sistemi** veya **hastane bilgi yönetim sistemi** (karmaşık ve güvenilir olması gereken modüller içerir).
- **Pazar koşulları belirsiz**
 - Süreç: **Çevik Sprintler**
 - Örnek Proje: **E-ticaret start-up platformu** (önce temel satış modülü açılır, sonra hızlıca ödeme, kampanya, teslimat gibi özellikler eklenir).

Kurumsal Süreçler

Büyük yazılım organizasyonları, kendi ihtiyaçlarına uygun özel süreçler geliştirir:

Amazon (E-ticaret)

- Çevik yöntemlerin öncülerindendir.
- Çoğu proje **4 haftalık sprintler** halinde yürütülür.

Lockheed Martin (Havacılık & Savunma)

- ABD hükümetinin sözleşme yapısına uyumlu **Modifiye Şelale Modeli** uygular.

Microsoft (PC Yazılımları)

- Geniş ekipman yelpazesinde test ve **geriye dönük uyumluluğa** önem verir.
- Genellikle **Spiral Geliştirme** süreci kullanır.

Getir & Trendyol (Hızlı teslimat / E-ticaret)

- **Hızlı üretim süreci** ihtiyacı vardır.
- Çoğunlukla **Çevik yöntemleri** benimserler.

Sözleşme

Yazılım geliştirme sözleşmeleri hakkında not

Bazı kuruluşlar, **Şelale Modeli**'nin her aşaması için ayrı sözleşme yapar.

Her aşama tamamlandığında → **müşteri onayı + ödeme** alınır.

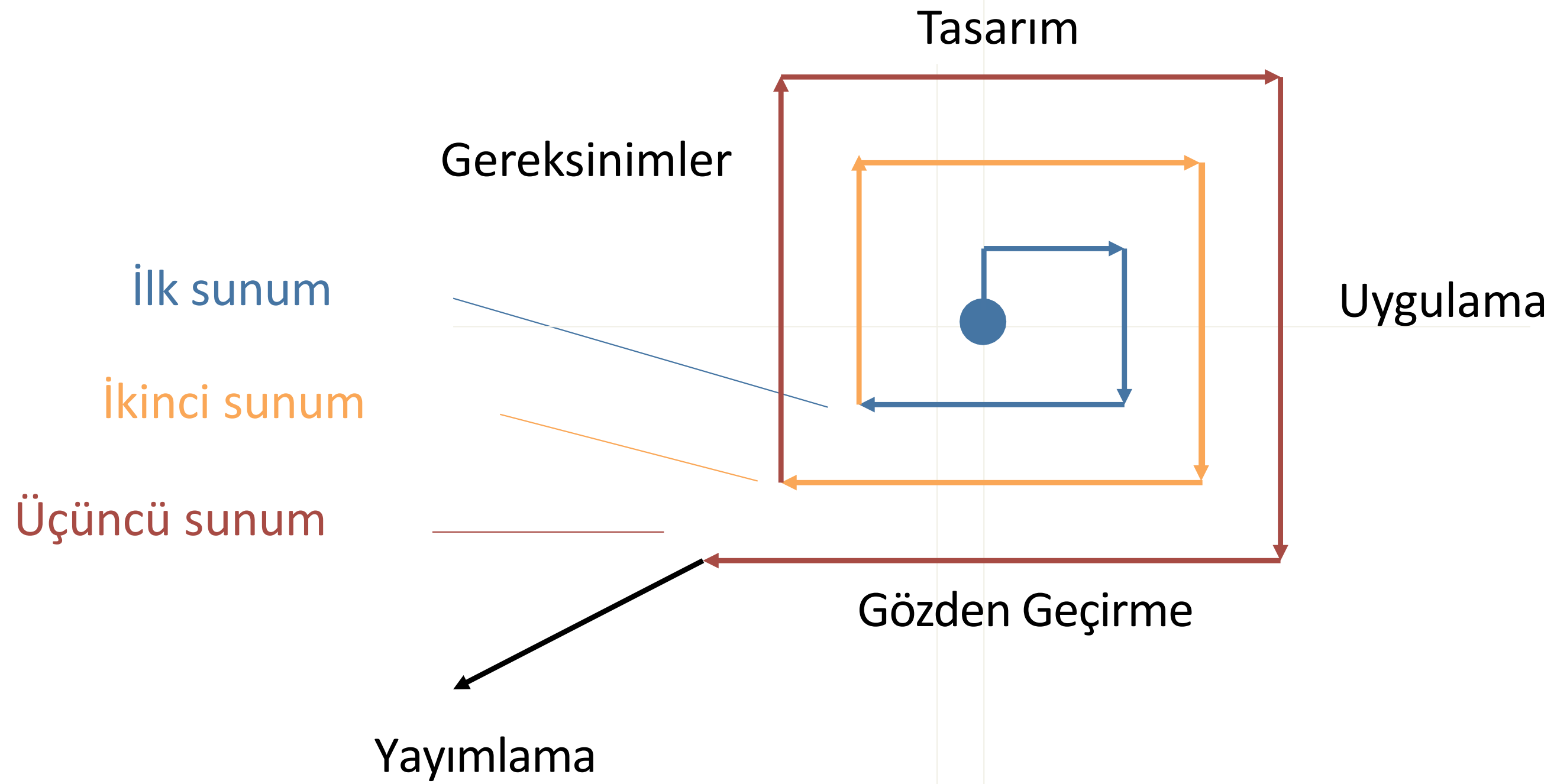
Yani her adımın tamamlanması, **finansal bir müzakere** ile bağlanır.

 **Sorun:**

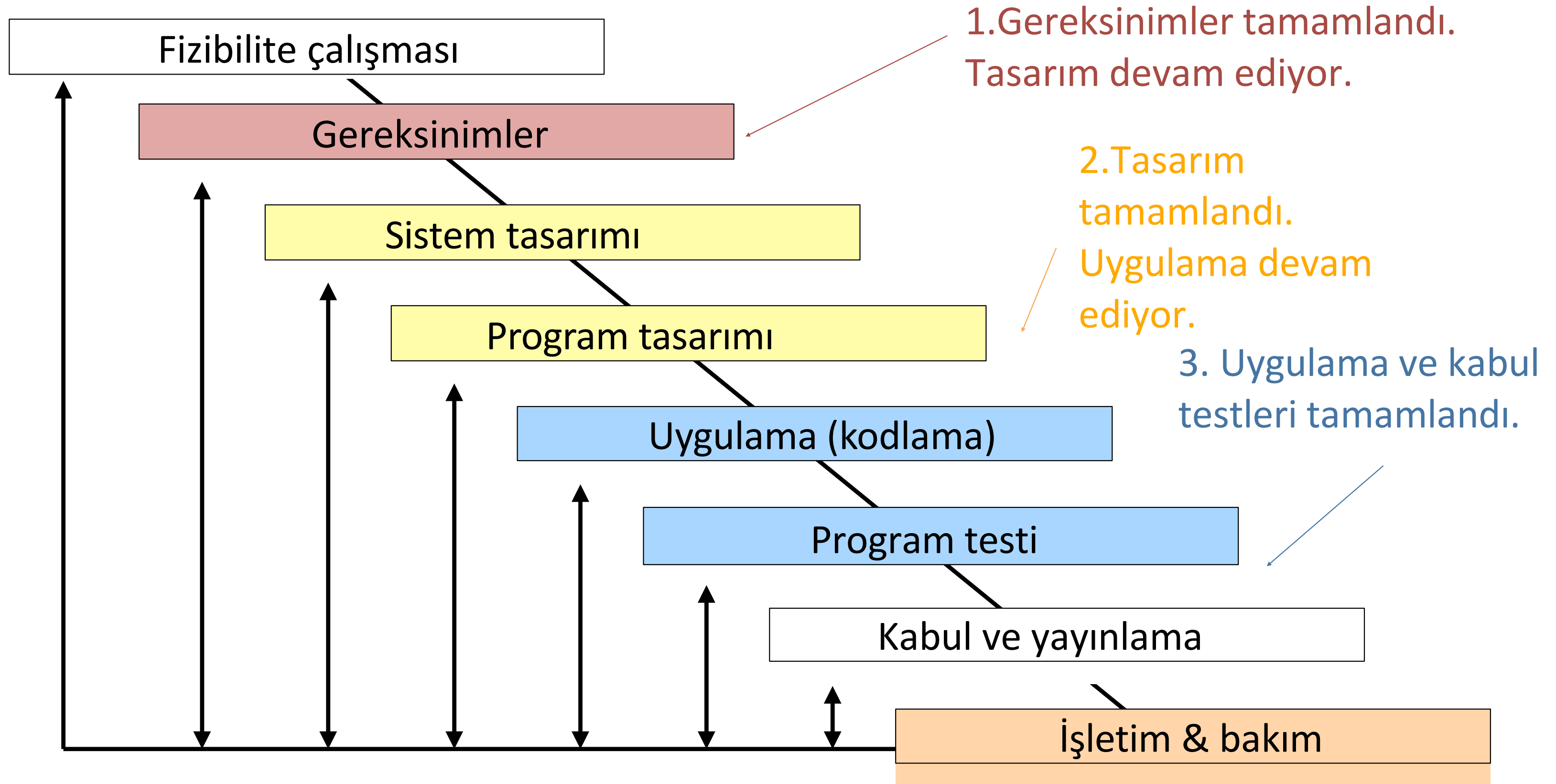
Bu yaklaşım pratikte **kötü bir uygulamadır**.

Çevik yöntem savunucuları tarafından sıkça eleştirilir.

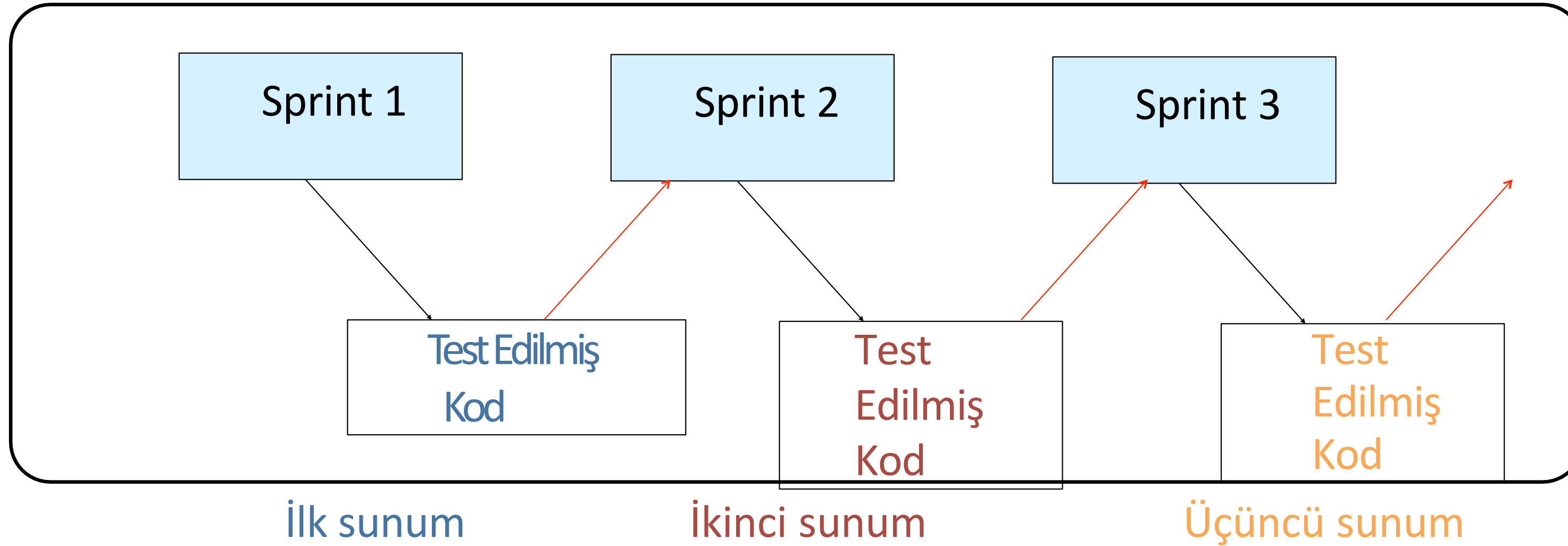
BZ 313 Projeleri: Yinelemeli İyileştirme



BZ 313 Projeleri: Modifiye Şelale Modeli



BZ 313 Projeleri: Çevik Geliştirme



Her sprint hedefi kodun bir bölümünü tamamlamalıdır.

Moda ve Vızıltılı Kelimeler

•Her birkaç yılda bir, yeni bir metodoloji büyük iddialarla ortaya çıkar:
Heroic Programming, Agile, Scrum, DevOps, Jenkins Pipeline ...

•Hepsi, öncekilere göre büyük gelişme olduğunu iddia eder.

•Gerçekte:

- Bazıları iyi fikirler içerir.
- Bazıları ise sadece eski kavramların yeniden paketlenmiş halidir.

•Her yeni metodoloji, kendi “**vızıltı kelimelerini**” üretir.

⚠ Dikkat:

- Pazarlama söylemlerine aldanmayın.
- Tüm yazılımlar için tek bir doğru süreç yoktur.
- En iyi süreç bile → iyi, güvenli, güvenilir sistemler için yetenekli geliştiricilere ihtiyaç duyar.

İddia Edilen

Yeni metodoloji, yazılım geliştirmede devrim yaratıyor

Bu süreç her tür yazılım için çalışır

Süreç sayesinde yüksek kalite garanti

Yeni süreç = Yeni kavramlar ve kelimeler

Gerçekte

Bazı yenilikler var ama çoğu eski fikirlerin yeniden paketlenmiş hali

Tek bir süreç tüm yazılım türleri için uygun değildir

Kaliteli, güvenli ve güvenilir yazılım için **yetenekli geliştiriciler** şarttır

Çoğu zaman sadece **vızıltı kelimeler**

Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

BZ 313 Yazılım Mühendisliği

3. Yazılım Geliştirme Süreçleri

Dersin Sonu