Erciyes Üniversitesi
Bilgisayar Mühendisliği Bölümü

BZ 313 Yazılım Mühendisliği

3. Yazılım Geliştirme Süreçleri

İşlem Sırası

Ders 2’de bazı işlem adımları anlatıldı:

- Gereksinimler
- Kullanıcı arayüz tasarımı
- Sistem tasarımı
- Program geliştirme (Tasarım ve Kodlama)
- Kabul ve yayınlama

Her yazılım projesi bu temel adımları bir şekilde içermelidir, 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:**

  Bir sonrakine başlamadan önce her işlem adımını tamamlanır.

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

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

- **Spiral:**

  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.

Ağır (Heavyweight) ve Hafif (Lightweight) Yazılım Geliştirme

Heavyweight bir süreçte, amaç her adımı tamamen tamamlamak ve daha sonra minimum değişiklik ve revizyon yapmaktır. Her adım, bir sonraki adıma başlamadan dokümante edilir.

Örnek: Modifiye Edilmiş Şelale Modeli (Modified Waterfall Model)

Lightweight bir süreçte, geliştirme ekibi minimum seviyede ara doküman oluşturur. Daha önceki deneyimlere dayanarak değişikliklerin yapılacağı kabul edilir ve yalnızca nihai sistemin dokümantasyonun gerçekleştirilmesi beklentisi vardır.

Örnek: Çevik (Agile) Yazılım Geliştirme

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/

Geçmişte

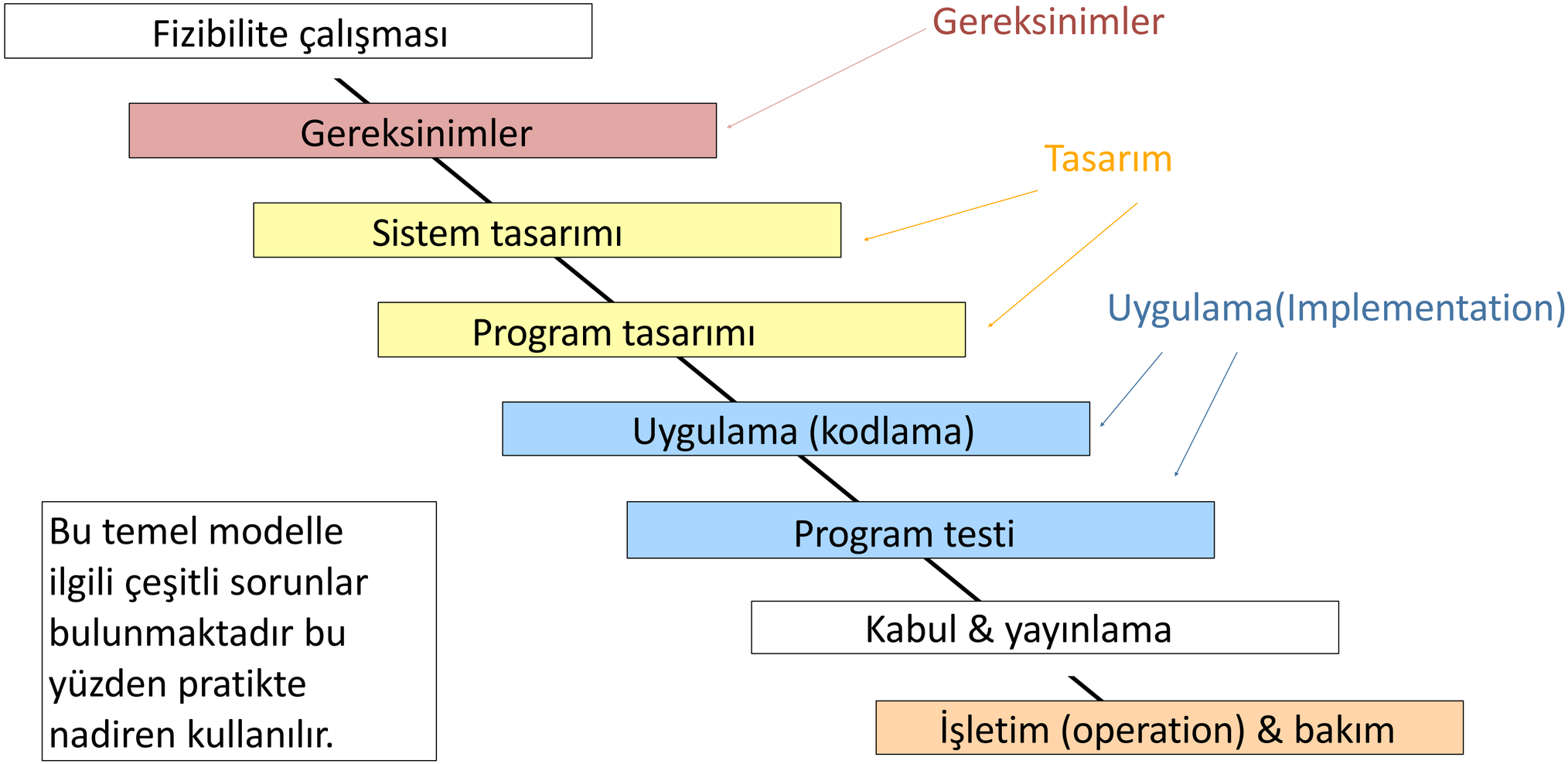
Yazılım mühendisliği, bir disiplin olarak, 1970'lerin başından beri bulunmaktadır.

O vakitlerde:

- Çoğu bilgisayar sistemi, daha önce manuel olarak yapılan sistemlerin dijital dönüşümleriydi (örneğin; bordro, faturalandırma, havayolu rezervasyonları vs.). Gereksinimler belliydi.
- Birçok sistem aynı mimariyi takip etti, yalnızca ana dosya güncellendi. Sistem tasarımı iyi anlaşılmıştı.
- Kodlama, modern dillerin ve araçların hiçbiri olmadan sıkıcıydı. Bu nedenle kodlamaya başlamadan önce iyi bir program tasarımına sahip olmak önemliydi.

Bu faktörler yazılım geliştirmede **Şelale Modeli**'nin doğuşuna yol açtı.

Şelale Modeli



Bu temel modelle ilgili çeşitli sorunlar bulunmaktadır bu yüzden pratikte nadiren kullanılır.

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

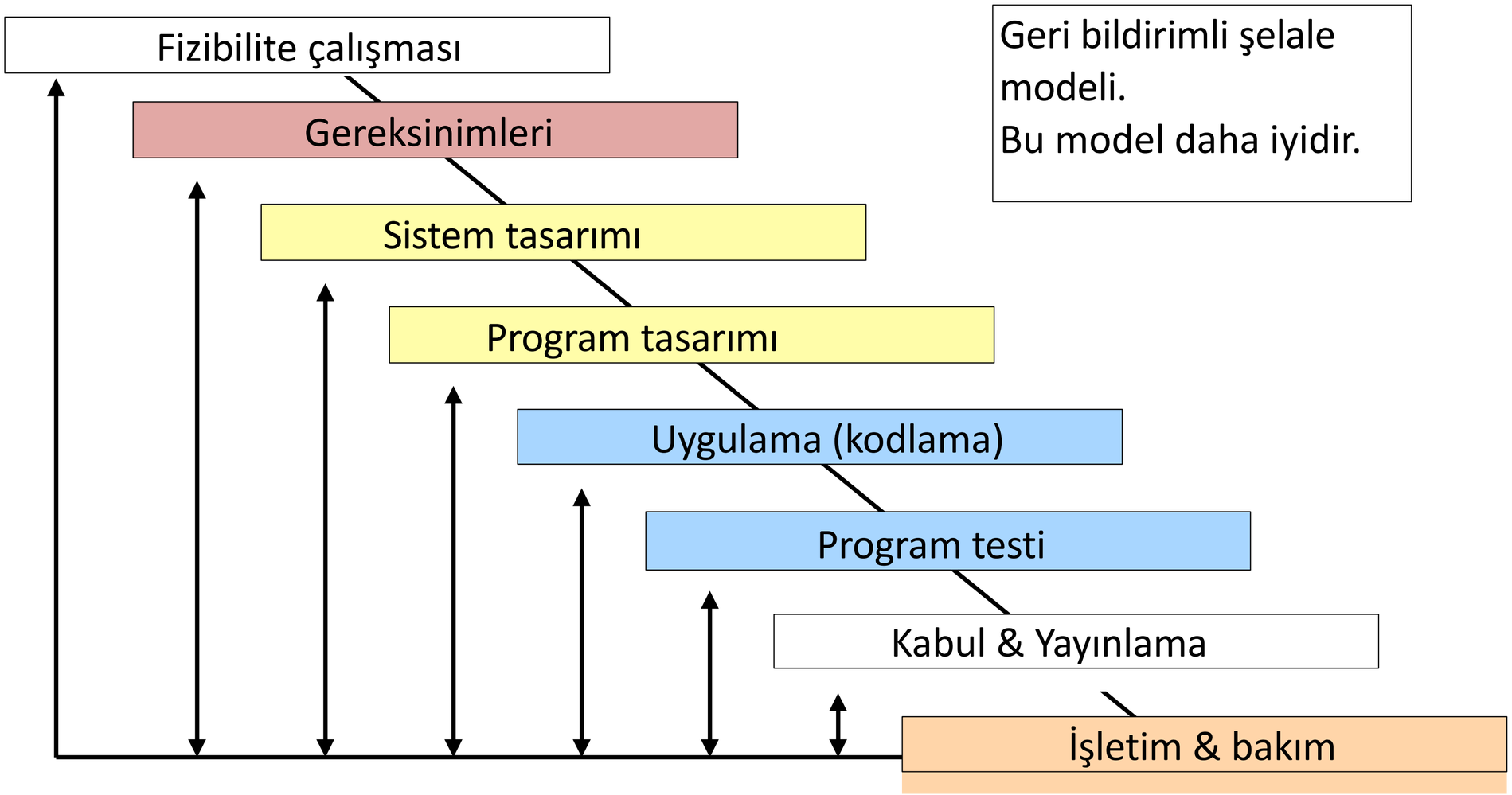
Saf bir sıralı modelin uygulanması mümkün değildir.

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



Geri bildirimli şelale modeli.
Bu model daha iyidir.

Modifiye Şelale Modeli Ne Zaman Kullanılır?

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

- Gereksinimlerin iyi anlaşıldığı manuel bir veri işleme sisteminin dönüştürülmesi (ör., elektrik faturaları).
- İşlevselliği önceki bir ürüne oldukça benzeyen bir sistemin yeni sürümü (ör. 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

Özellikle kullanıcı arayüzlerinde operasyonel bir sistem oluşturulana kadar gereksinimlerin anlaşılması zordur.

Sistem ve program tasarımında prototiplerden faydalanılabilir.

Süreç (Process)

- Geliştirme sürecinin başlarında bir **prototip** sistem oluşturun.
- Prototipi client ile gözden geçirin ve kullanıcılarla test edin, böylelikle gereksinimler daha anlaşılır olur ve tasarım netleşir.
- Prototipi bir dizi iterasyon ile iyileştirin.

Yinelemeli İyileştirme: Örnek

Problem: Programlama ortamına grafik paketi ekleme

Gereksinimler

Client, farklı nesneler arasındaki koordinatların nasıl yönetileceğine ilişkin syntax gibi birkaç önemli gereksinimden emin değil.

Süreç

- Bir ön işlemci (preprocessor) ve ön çalışma zamanı (preliminary run-time) paketi ile prototip sürümü oluşturun.
- Birden fazla iterasyon uygula. Her bir iterasyonda:

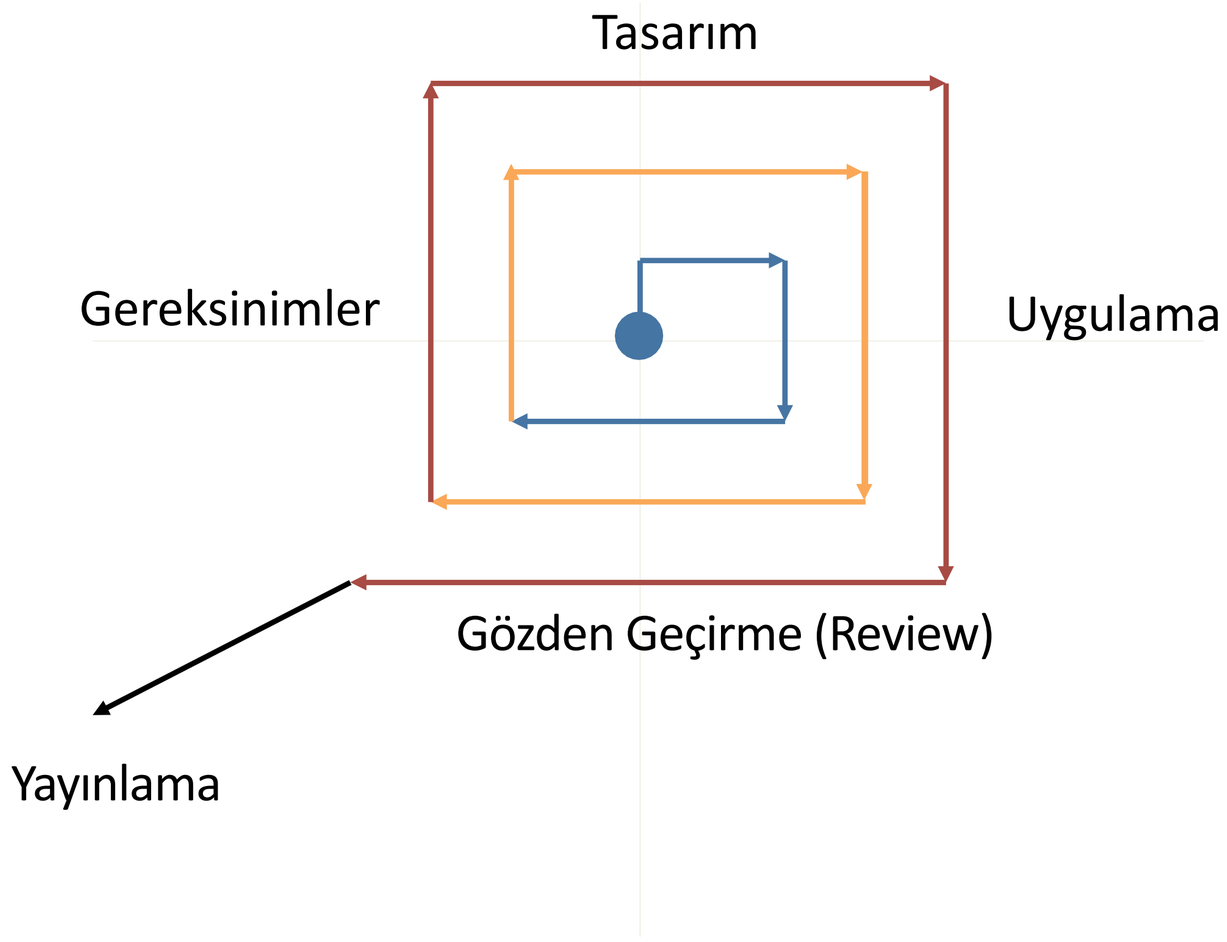
  -> Sistemi kullanıcılarla test edin

  -> Değişiklik yap

  -> Kullanıcılar işlevden memnun kalana kadar tekrarla
- Son yineleme olarak, ön işlemciyi değiştirin ve çalışma zamanı paketindeki yamaları ve kısayolları ortadan kaldırın.

Bu, yinelemeli iyileştirmeye bir örnektir.

Yinelemeli İyileştirme



Yinelemeli İyileştirme Üzerine Tartışma

Yinelemeli iyileştirme, işlem sırasında oluşturulan belgelere sahip medium weight bir işlemdir.

Yinelemeli iyileştirmede, client'ın planlanan sistemi geliştirme sırasında erken gözden geçirmesini sağlayan çeşitli teknikler kullanılır:

- Kullanıcı arayüzü mock-up'ları
- Atılabilir yazılım bileşenleri
- Dummy modüller
- Hızlı prototipleme
- Ardışık iyileştirme

Client ve kullanıcının değerlendirebilmesi için olabildiği kadar hızlı bir uygulama geliştirin, ancak bu uygulamayı yayınlamayın.

Spiral Geliştirme

Örnek

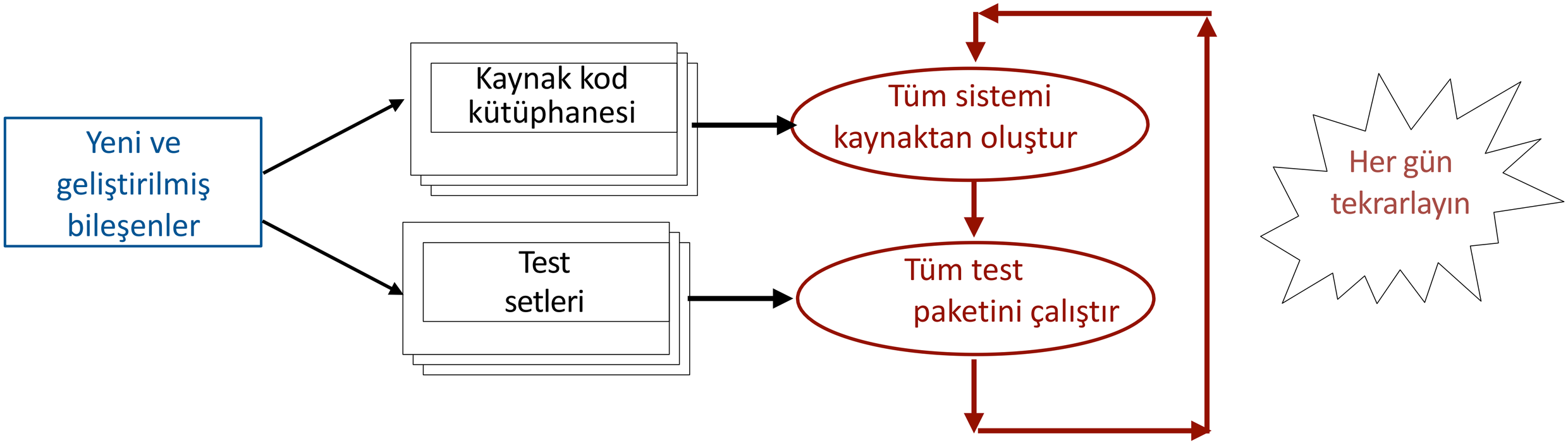
İşletim sisteminin yeni bir sürümünü geliştirme (Microsoft).

Spiral geliştirme

Spiral geliştirmede her zaman tamamen test edilmiş bir sistem vardır, ancak işlevsellik eksiktir.

- Eksik bileşenler için dummy stub'larla nihai ürünün genel yapısına sahip bir temel sistem oluşturun.
- Tamamlanan her bileşen için, kapsamlı bir test paketi oluşturun.
- Geliştirme ekipleri, her biri bir dizi test paketi içeren yeni veya geliştirilmiş bileşenler oluşturur. Bu bileşenler kaynak kod kütüphanesine eklenir.
- Günlük bir döngüde, test ediciler tüm sistemi kaynak kodu kütüphanesinden çekerek ayağa kaldırır ve bütün test kümesini çalıştırır.

Spiral Geliştirme



Spiral Geliştirme Üzerine Tartışma

Spiral geliştirme, büyük sistemlerin yeni versiyonlarını geliştirmek için yaygın olarak kullanılmaktadır:

- Genel sistem mimarisi iyi anlaşılmıştır.
- Büyük bileşenler ayrı ayrı geliştirilebilir ve test edilebilir.
- Sistem kapsamlı bir otomatik test seti üretmenin getireceği yükün hakkını verecek seviyede değerlidir (testler oldukça maliyetlidir).

Zorluklar

- Mimaride büyük değişiklikler yapmak zor.
- Birçok bileşeni etkileyen değişiklikler yapmak zor.

Çevik Yöntemler

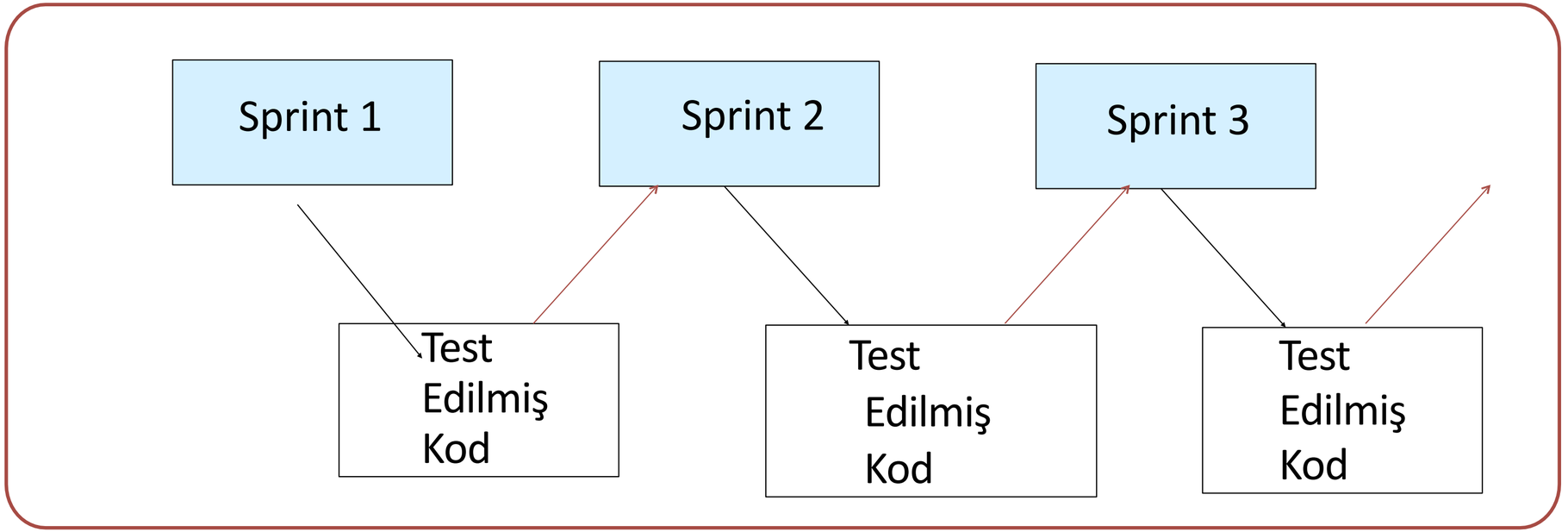
Çevik terimi çeşitli yöntemler için kullanılır.

Genel ilkeler:

- Büyük bir proje, sprint adı verilen küçük artımlara bölünür.
- Geliştirme 4 ila 9 kişilik küçük ekipler tarafından gerçekleştirilir.
- Program 2 ila 4 hafta gibi sabit zaman çerçevelerine bölünür.
- Her sprint, takımın bir yazılım projesinin yalnızca bir bölümünü tamamladığı bir zaman çerçevesidir. Tek bir sprint, gereksinimler, tasarım, kodlama ve test gibi çeşitli işlem adımlarını içerir.
- Her sprint, üretime alınmaya hazır, tamamen test edilmiş kodla sona erer.

Bu, geliştirme sırasında oluşturulan minimum dokümantasyona sahip lightweight bir süreçtir, ancak son sürümde gelecekteki bakım için tam dokümantasyon oluşturulmalıdır.

Çevik Geliştirme



Her sprintten sonra kod şöyle olabilir:

- yayımlanmış (Orijinal Çevik Yöntem)
- sonraki sürüm için diğer sprint'lerden gelen kodla birleştirilir
- daha büyük bir kod tabanına dahil edilir (spiral geliştirme)

Çevik: Kod Yayımlama

Çevik sürümler

- Çevik süreçlerin orijinal sürümünde, her sprint yayımlanan bir kod ile sona erer
- Bu pratikte nadiren mümkündür.
- Bu derste, Üretime (production) hazır kalitede kod oluşturmak için bir sprint tanımlanacaktır.
- Bazı insanlar herhangi bir kısa aktiviteyi kapsayacak şekilde "sprint" terimini kullanır, ancak bu çevik ruhun ötesinde bir davranıştır.

Çevik Geliştirme: Rework

Çevik geliştirmenin zorluğu

Çevik yaklaşım, yerleşik bir mimari içinde bir sistemin geliştirilmesi veya sürekli iyileştirilmesi için oldukça uygundur.

Üst düzey bir takım genel mimariyi oluşturmalı ve sprintleri koordine etmelidir.

Rework

Çevik geliştirme ile genel sistemin gereksinimleri ve tasarımı kademeli olarak ortaya çıkar.

- Kaçınılmaz olarak bazı ilk sprintlerin bazı bölümlerinin yeniden ele alınması gerekecektir.
- Bu yüzden de zaten tamamen test edilmiş ve yayınlanmış olabilecek kodda değişikliklerin meydana gelmesi gayet olasıdır. Bu da yazılım geliştirmede oldukça tuhaf bir durumdur.

Yeniden ele alma çok fazla gerçekleşiyorsa, her bir bileşeni tamamen iyileştirmeyle uğraşmak yerine tekrardan ele alma miktarını en aza indirmek için yinelemeli iyileştirme kullanmak daha verimli olacaktır

Karma Süreçler

Uygulamada birçok büyük proje, dört tür yazılım sürecinin çeşitli yönlerini birleştiren süreçleri kullanır.

Örnekler

- Gereksinimleri iyi anlaşılmış bir proje, gereksinimleri ve sistem tasarımını belirlemek için değiştirilmiş bir şelale yaklaşımını ve ardından bir dizi çevik sprint kullanabilir.
- Belirsiz gereksinimleri olan bir proje, gereksinimleri açıklığa kavuşturmak için yinelemeli iyileştirmeyi kullanabilir ve ardından son sürümü oluşturmak için modifiye şelale modeli kullanabilir.
- Spiral geliştirme ile yeni bileşenler bir dizi sprint olarak geliştirilebilir.

- Kullanıcı arayüzlerinin kullanıcılarla test edilmesi gerekir. Bu, sistemin geri kalanı için hangi süreç kullanılırsa kullanılsın, yinelemeli geliştirmenin uygulanmasını zor hale getirir.

Karma Süreçler: Aşamalı (phased) Geliştirme

Büyük bir proje iki veya daha fazla aşamaya bölünebilir.

Sistemin temel işlevselliğe sahip ilk versiyonu hızlı bir şekilde üretime alınır (Aşama 1).

Sonraki aşamalar, önceki aşamanın kullanıcılarından edinilen deneyime göre şekillenir.

Avantajları

- Yapılan yatırımın geri dönüşü hızlı olur.
- Sonraki aşamalar geliştirilirken gereksinim daha net anlaşılır.
- Maliyetler daha uzun sürelere yayılabilir.

Yazılım Süreçleri Hakkında Gözlemler

Tamamlanan projeler tüm temel süreç adımlarını içermelidir
ama... Geliştirme sürecinin çoğu zaman yalnızca bir kısmı kısmen evrimseldir.

Risk şu şekilde azaltılabilir:

- Temel bileşenlerin **prototipini** oluşturma
- Sıklıkla **yayımlama** veya büyük projeleri **aşamalara** bölme
- **Kullanıcı** ve **müşterilerle** erkenden ve tekrar tekrar test etme
- Görünür bir yazılım süreci **takip etme**

Uygun Yazılım Sürecini Seçme

Yazılım geliştirme sürecindeki değişiklikler oldukça maliyetlidir.

- Gereksinimler yeterince anlaşılmamışsa veya değişmesi bekleniyorsa, esnekliği koruyan bir süreç seçin. Yinelemeli iyileştirme, Çevik Sprintler, aşamalı uygulama (phased implementation).
- Büyük bir yazılım sisteminin birbiriyle ilişkili birçok bileşeni varsa, geliştirme sırasında bir sistemin tasarımında büyük değişikliklerden kaçının. Modifiye şelale modeli.
- Yazılımın pazarı yeterince anlaşılmamışsa, yazılımı müşterilerin önüne mümkün olduğunca çabuk çıkaran bir süreç kullanın. Çevik sprintler.

Kurumsal Süreçler

Büyük yazılım geliştirme organizasyonlarının ihtiyaçları için tasarlanmış kendi iç süreçleri vardır. Mesela:

- Amazon (İnternet ticareti) çevik yöntemlerin kullanılmasında öncüdür. Çoğu yazılım geliştirme süreci, yaklaşık dört haftalık sprintlere bölünmüştür.
- Lockheed Martin (Havacılık), ABD hükümetinin yazılım sözleşmelerini yönetme biçimine uyan değiştirilmiş bir şelale modelini izler.
- Microsoft (PC yazılımı), çok çeşitli ekipmanlarla test etmeye ve geriye dönük uyumluluğa büyük önem vermektedir. Geliştirme sürecinde genellikle spiral bir süreç kullanır.
- Getir ve Trendyol, hızlı bir üretim sürecine ihtiyaç duyar. Geliştirme sürecinde çoğunlukla Çevik yöntemler kullanır.

Sözleşme

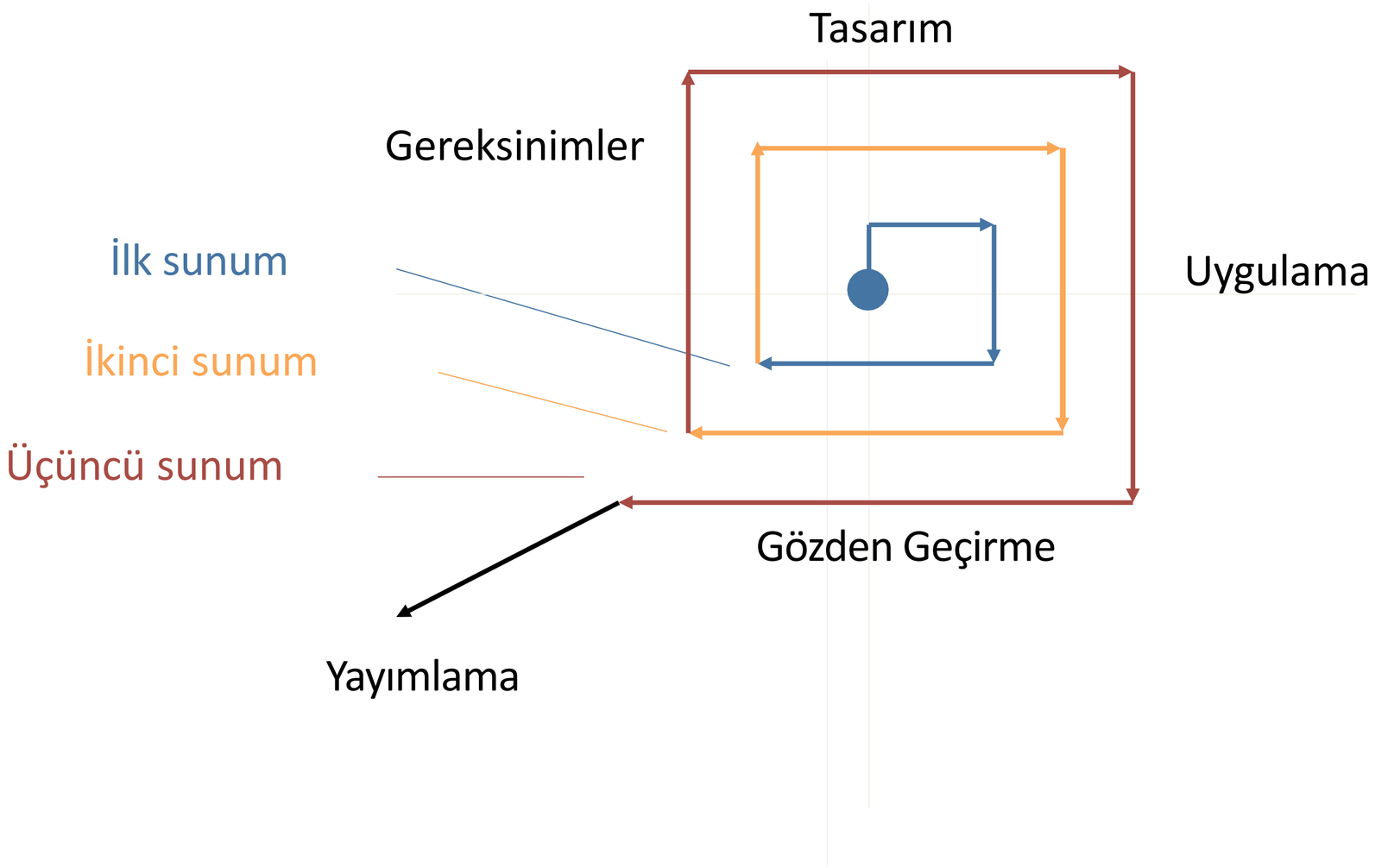
Yazılım geliştirme sözleşmeleri hakkında not

Bazı kuruluşlar, Şelale Modeli'nin her aşaması için ayrı sözleşmeler koyarak yazılım geliştirme için sözleşme yapar ve her aşamadan sonra ödeme yaparlar.

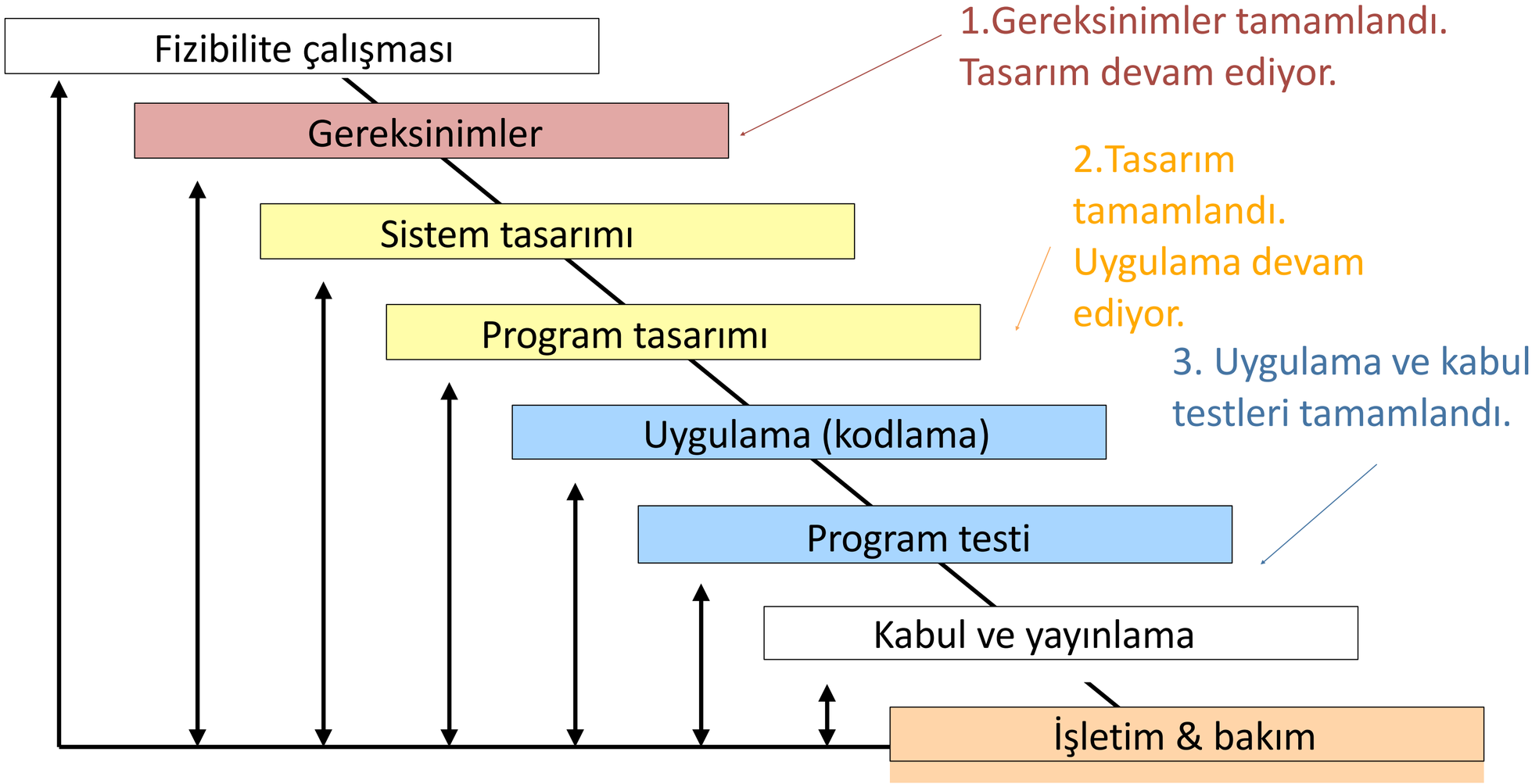
Çoğu zaman, müşterinin her işlem adımından sonra sözleşmeyi kabul edip imzalaması beklenir. Her adımın tamamlanması finansal bir müzakere ile gerçekleştirilir.

Bunlar pratikte kötü uygulamalardır ve çevik süreçlerin savunucuları tarafından haklı olarak eleştirilmektedir.

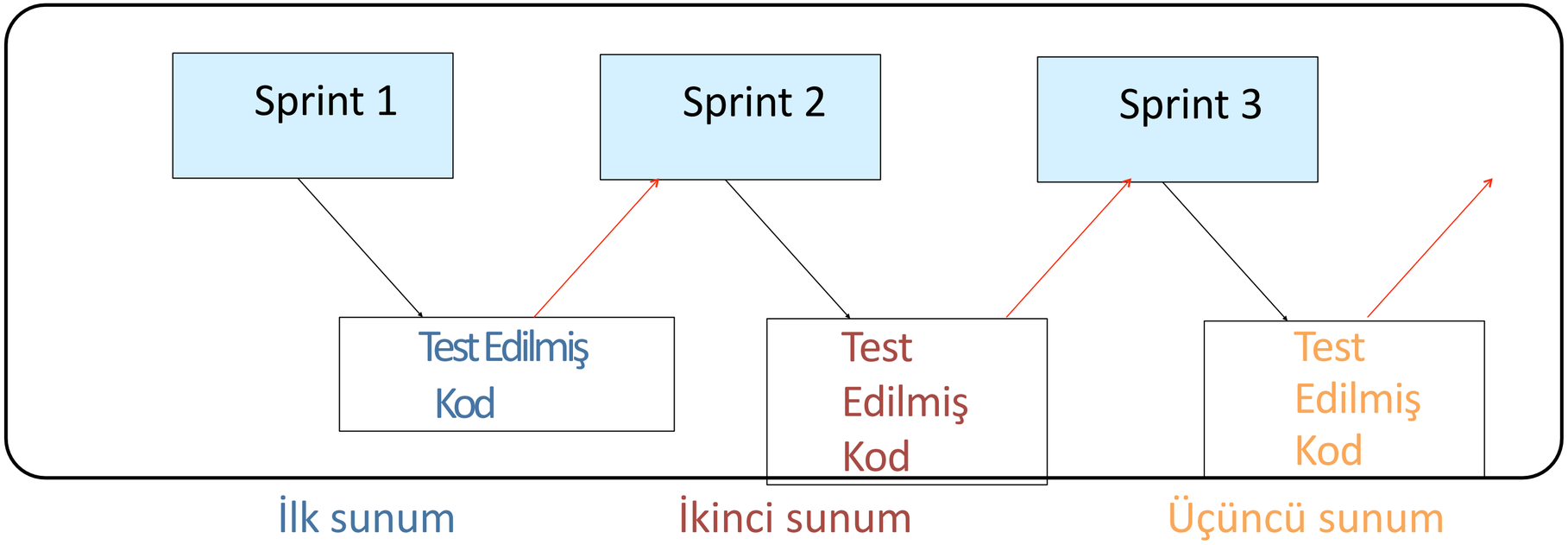
BZ 313 Projeleri: Yinelemeli İyileştirme



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 yazılım metodolojisi büyük bir tantana ile tanıtılır.

Bu vakitlerden örnek verecek olursak: heroic programming, agile, scrum, devOps, Jenkins pipeline, vs.

- Hepsi, yazılım geliştirmenin önceki yollarına göre büyük bir gelişme olduğunu iddia eder.
- Çoğu bazı iyi fikirler içerir, ancak diğerleri eski kavramların basit yeniden formülasyonlarıdır.
- Her yeni metodoloji yeni vızıltılı kelimeler icat eder.

Pazarlamaya aldanmayın.

- Her tür yazılım için çalışan bir geliştirme süreci yoktur.
- En iyi süreçler bile iyi, güvenli, güvenilir sistemler oluşturmak için yetenekli geliştiricilere ihtiyaç duyar.

BZ 313 Yazılım Mühendisliği

3. Yazılım Geliştirme Süreçleri

Dersin Sonu