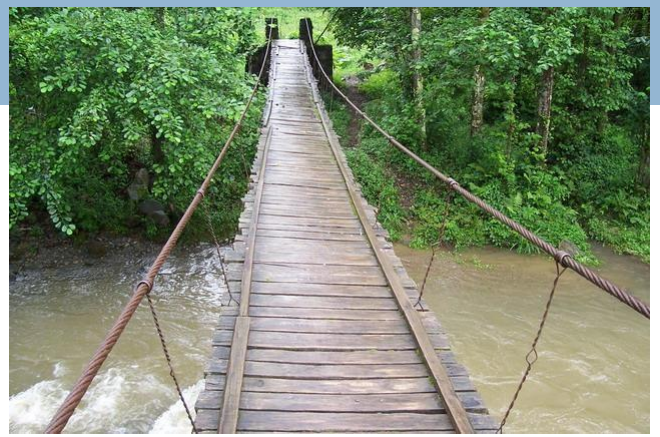
Hafta 2: Yazılım Süreç Modelleri

MODERN DÜNYADA YAZILIMIN YERİ

- Modern dünya yazılım olmadan işlemiyor.
  - Ulusal altyapı ve hizmetler
  - Elektrikli ürünler
  - Giyilebilir teknoloji
  - Endüstriyel üretim
  - Eğlence
  - …
    - → Yazılım her yerde!!

BAŞARISIZLIK HİKAYELERİ -TEMEL SEBEPLER



- *Artan talepler*
  - Hızlı geliştirme ve teslimat
  - Daha büyük ve karmaşık sistemler
  - Daha önce mümkün olmadığı düşünülen özellikler
  - Eskiyen geliştirme yaklaşım ve yöntemleri
- *Düşük beklentiler*
  - Metodoloji kullanmadan programlama kolaylığı
  - Mevcut fakat bilinmeyen/kullanılmayan geliştirme yöntemleri
  - Daha pahalı ve daha az güvenilir yazılım

"Profesyonel Yazılım"

→ Ekip işi !

→ Yazılım mühendisliği önemli !!

YAZILIM SÜREÇ MODELLERİ

- Yazılım geliştirmenin bahsedilen zorluklarıyla başedebilmek için, geliştirmeyi sistematik hale getirmeyi hedefleyen süreç modelleri ortaya çıkmıştır.
  - Bu modellerin temel hedefi; proje başarısı için, yazılım geliştirme yaşam döngüsü ("software development life cycle") boyunca izlenmesi önerilen mühendislik süreçlerini tanımlamaktır.
    - *Yazılım geliştirme yaşam döngüsü*: Bir yazılım ürününün ihtiyacının ortaya çıkmasından kullanımdan kalkmasına kadar geçen dönemdir.
  - Modellerin ortaya çıkmasında, ilgili dönemin donanım ve yazılım teknolojileri ile sektör ihtiyaçları önemli rol oynamıştır.
  - Örnek:
    - Geleneksel modeller (örneğin; çağlayan ("waterfall") modeli)
    - Çevik ("agile") modeller (örneğin; uçdeğer ("extreme") programlama modeli - XP)

YAZILIM SÜRECİ VE SÜREÇ MODELİ

- Süreç nedir?
  - Belirli bir hedef için gerçekleştirilen adımlar zinciridir. [IEEE]
- Yazılım süreci nedir?
  - Yazılımı ve ilişkili ürünlerini geliştirmek ve idame ettirmek için kullanılan etkinlikler, yöntemler, pratikler ve dönüşümlerdir. [SEI]
  - Yazılım geliştirme ve idame amacı güden etkinlikler setidir.
  - Yazılım süreç modeli nedir?
  - Bir yazılım sürecinin belirli bir bakış açısıyla gösterilmiş, basitleştirilmiş temsilidir.
- Örnek bakış açıları:
  - İş-akışı → etkinlikler nasıl sıralı?
  - Veri-akış → bilgiler nasıl sıralı?
  - Rol-hareket → kim ne yapıyor?

Kim, neyi, hangi sırada yapıyor?

Ne kullanıyor?

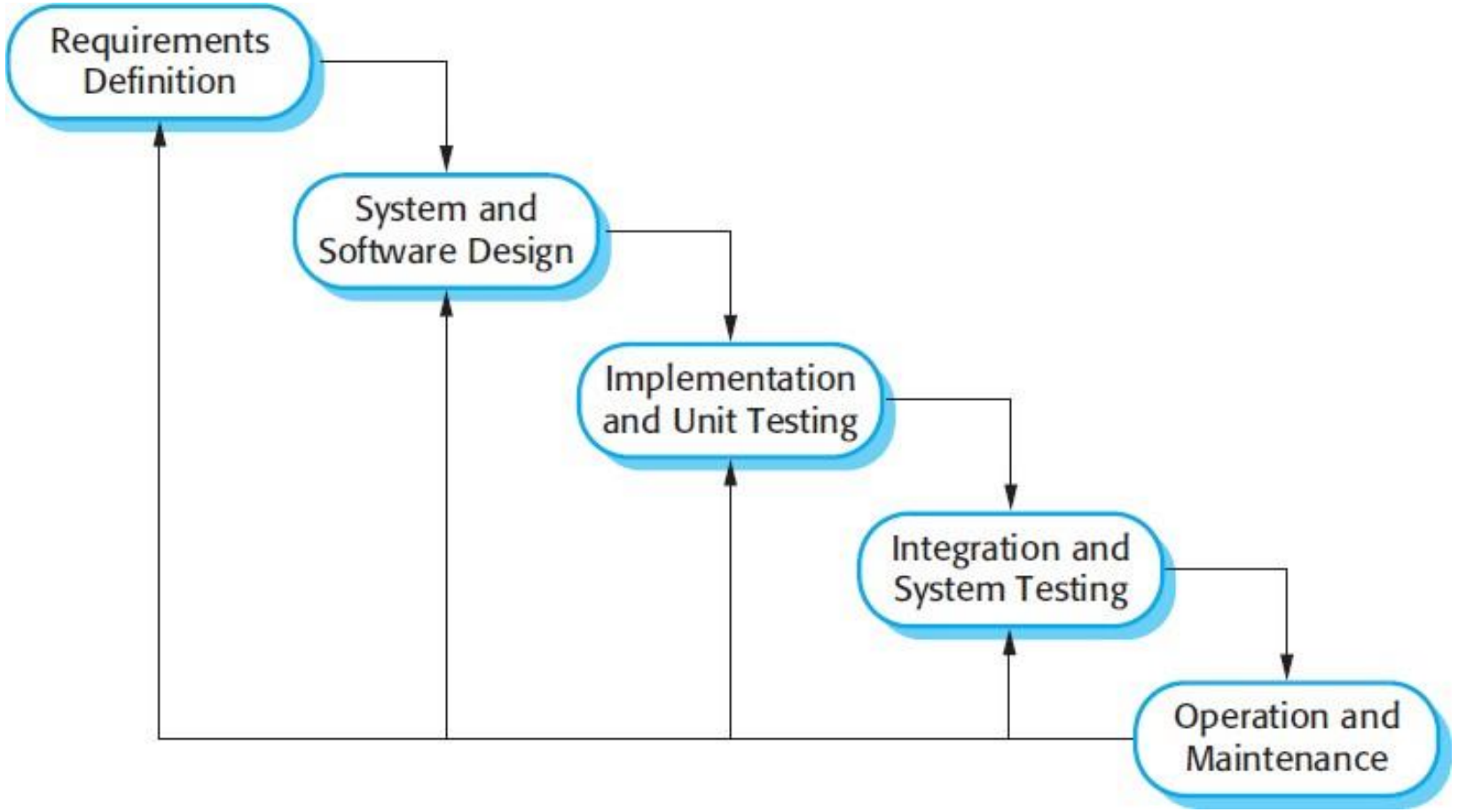
Ne üretiyor?

GELENEKSEL (“TRADITIONAL”) YAZILIM SÜREÇ MODELLERİ

GELENEKSEL YAZILIM SÜREÇ MODELLERİ

- Çağlayan (“waterfall”) modeli
- Evrimsel (“evolutionary”) model
- Bileşen-tabanlı (“component-based”) model
- Artırımlı (“incremental”) model
- öngüsel (“spiral”) model

ÇAĞLAYAN ("WATERFALL") MODELİ



ÇAĞLAYAN MODELİ-AŞAMALAR

- Gereksinim Tanımlama: Gerçekleştirilecek sistemin gereksinimlerini belirleme işidir.
  - Müşteri ne istiyor? Ürün ne yapacak, ne işlevsellik gösterecek?
- Tasarım: Gereksinimleri belirlenmiş bir sistemin yapısal ve detay tasarımını oluşturma işidir.
  - Ürün, müşterinin beklediği işlevselliği nasıl sağlayacak?
- Gerçekleştirme ve Birim Test: Tasarımı yapılmış bir yazılım sisteminin kodlanarak gerçekleştirilmesi işidir.
  - Yazılım ürünü, tasarımı gerçekleştirecek şekilde kodlandı mı?
- Tümleştirme ve Test: Gerçekleştirilmiş sistemin beklenen işlevselliği gösterip göstermediğini sınama işlemidir.
  - Ürün, müşterinin beklediği işlevselliği sağlıyor mu?
- İşletme ve Bakım: Müşteriye teslim edilmiş ürünü, değişen ihtiyaçlara ve ek müşteri taleplerine göre güncelleme işidir.
  - Ürün müşteri tarafından memnuniyetle kullanılabiliyor mu?

ÇAĞLAYAN MODELİ-ZORLUKLAR

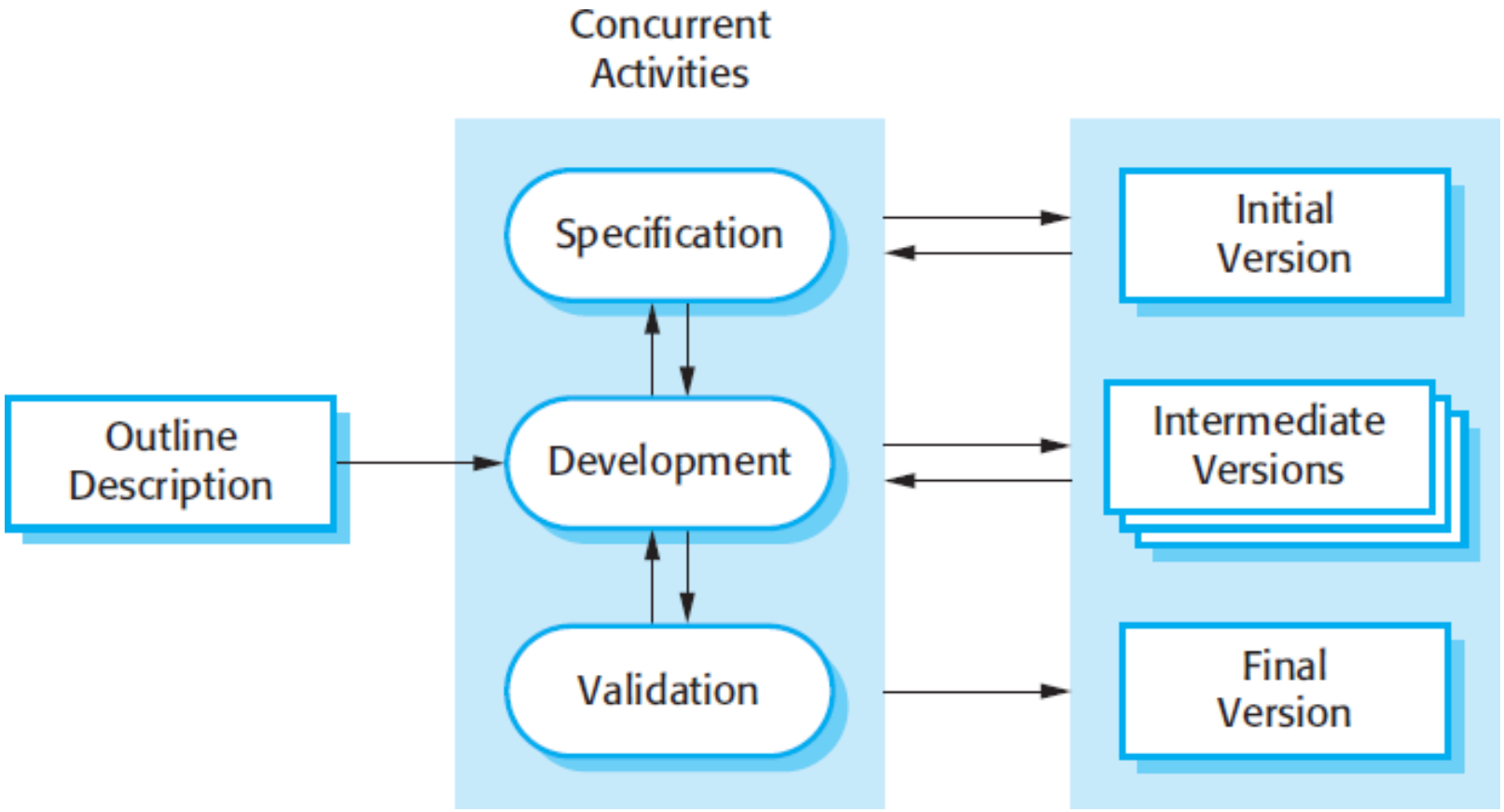
- Bir sonraki aşamaya geçmeden, önceki aşama neredeyse tümüyle tamamlanmış olmalıdır (örneğin, gereksinim tanımlama aşaması bitmeden tasarım aşamasına geçilmez.)
  - Bu şekilde geliştirme boyunca değişen müşteri isteklerinin sisteme yansıtılması zorlaşır.
  - Önceki nedenle bu model, gereksinimleri iyi tanımlı ve değişiklik oranı az olacak sistemler için daha uygundur.
  - Çok az sayıda iş sisteminin gereksinimleri başlangıçta iyi şekilde tanımlanabilir. Bu zorluğu aşmak için; gereksinim tanımlama aşamasından önce iş gereksinimlerinin anlaşılması ve tanımlanması faydalı olabilir.
  - Daha çok, geniş kapsamlı sistem mühendisliği projeleri için tercih edilir.

EVRİMSEL ("EVOLUTIONARY") MODEL

- Sistem, zaman içinde kazanılan anlayışa göre gelişir.
  - Amaç, müşteriyle birlikte çalışarak taslak bir sistem gereksinimleri tanımından çalışan bir sisteme ulaşmaktır.
  - En iyi bilinen gereksinimlerle başlanır ve müşteri tarafından talep edildikçe yeni özellikler eklenir.

- Öğrenme amacıyla, sonradan atılabilecek prototipler ("throw-away prototyping") geliştirilir.
  - Amaç, sistem gereksinimlerini anlamaktır.
  - En az bilinen gereksinimlerle başlanır ve gerçek ihtiyaç anlaşılmaya çalışılır.

EVRİMSEL MODEL – ADIMLAR



EVRİMSEL MODEL – ZORLUKLAR

- Geliştirme süreci izlenebilir değildir. Her seferinde eklemelerle çalışan sistem, müşteriyle gözden geçirilir.
- Zaman içinde kazanılan anlayışa göre geliştirilen sistemler, sıklıkla kötü tasarlanır.
- Küçük- ve orta-ölçekli, etkileşimli ("interactive") sistemler için uygulanabilir.
- Daha büyük ölçekli sistemlerin belirli bir bölümü (örneğin, kullanıcı arayüzleri) için uygulanabilir.
- İdamesi nispeten kısa sürecek sistemler için uygulanması önerilir.
  - Uzun yıllar idame edilecek sistemler, kötü / kötüleşen tasarım sebebiyle etkin çalışmayacaktır.

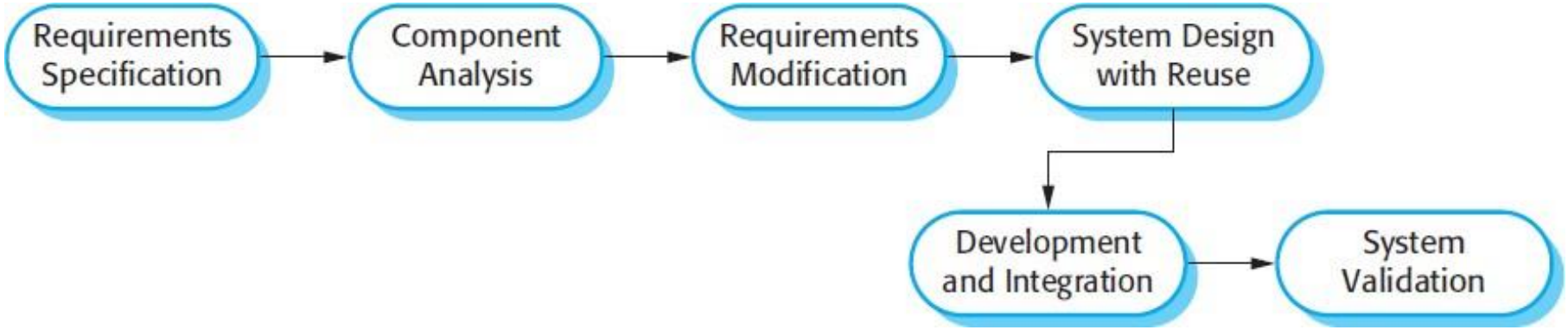
BİLEŞEN-TABANLI (“COMPONENT-BASED”) MODEL

- Sistemin COTS (“commercial-off-the-shelf”) adı verilen hazır bileşenler kullanılarak tümleştirilmesi esasına dayanır.

- Süreç adımları:
  - Bileşen analizi
  - Gereksinim günleme
  - Bileşenler kullanarak sistem tasarımı
  - Geliştirme ve tümleştirme

- Bu yaklaşım, bileşen standartlarındaki gelişmeler ilerledikçe daha yaygın olarak kullanılmaya başlanmıştır.

BİLEŞEN-TABANLI ("COMPONENT-BASED") MODEL – ADIMLAR



YAZILIM SÜREÇ MODELERİNDE SÜREÇ TEKRARI ("PROCESS ITERATION")

- Yazılım süreç modelleri tek bir defada uygulanmak yerine, birkaç tekrarda uygulanabilir.
  - Örneğin, geniş kapsamlı 5 alt sistemden oluşan bir sistemin; ilk alt sistemi için çağlayan modeli uygulandıktan sonra, geri kalanı için çağlayan modeli tekrar uygulanabilir.
  - Bu şekilde geliştirme riskleri en aza indirilerek ilk tekrarda kazanılan deneyimden, sistemin geri kalanı geliştirilirken faydalanılabilir.

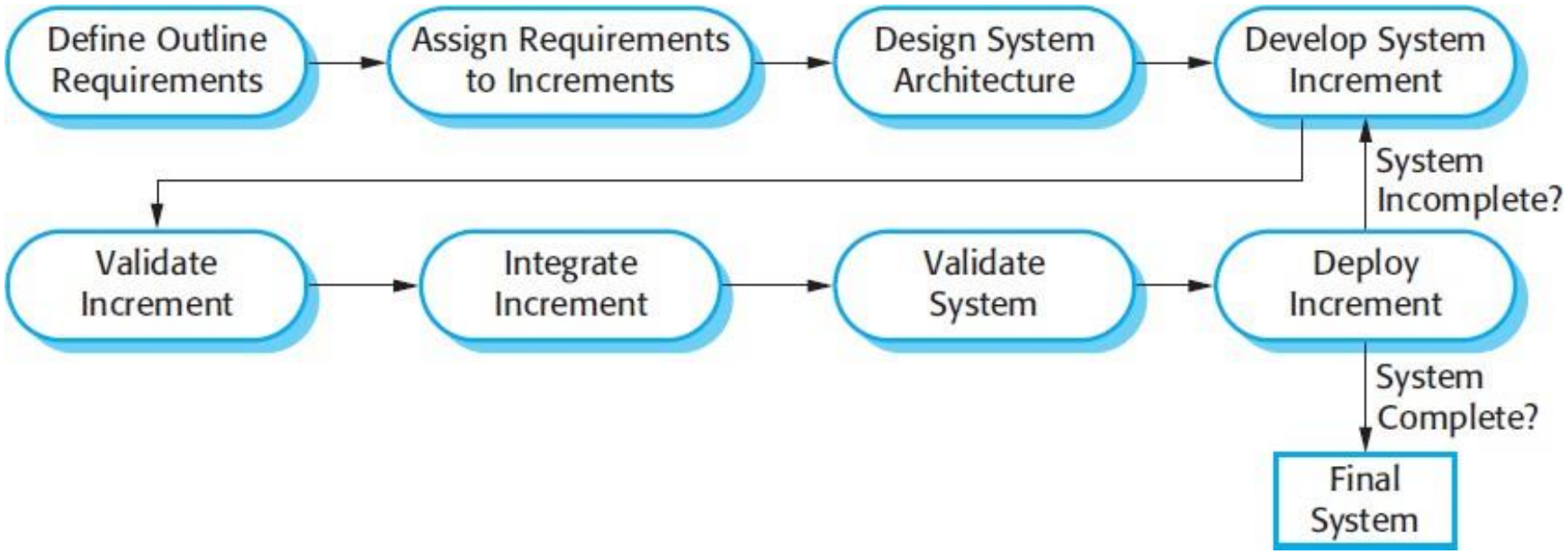
- Hangi süreç modelinin, sistemin hangi bölümleri için ve kaç tekrarda uygulanacağına proje başında karar verilir.

- Süreç tekrarıyla yakından ilişkili iki geleneksel model vardır:
  - Artırımlı ("incremental") model
  - Döngüsel ("spiral") model

ARTIRIMLI ("INCREMENTAL") MODEL

- Sistemi tek seferde teslim etmek yerine, geliştirme ve teslim parçalara bölünür. Her teslim beklenen işlevselliğin bir parçasını karşılar.
- Kullanıcı gereksinimleri önceliklendirilir ve öncelikli gereksinimler erken teslimlere dahil edilir.
- Bir parçanın geliştirmesi başladığında, gereksinimleri dondurulur. Olası değişiklikler sonraki teslimlerde ele alınır.

ARTIRIMLI MODEL – ADIMLAR



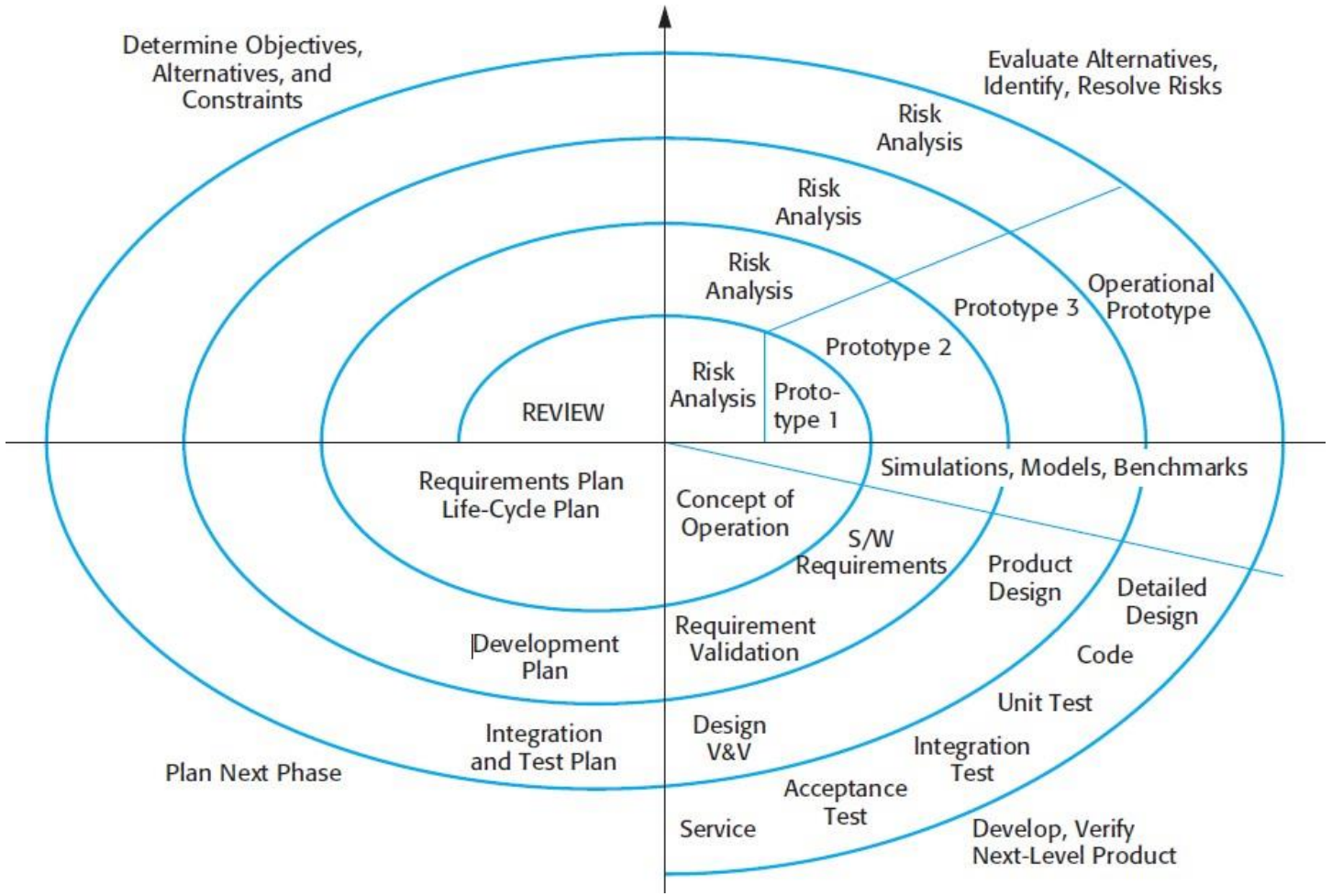
ARTIRIMLI MODEL – KAZANÇLAR

- Her teslimle birlikte müşteriye görünen bir değer döndüğünden, sistemin işlevselliği erken aşamalarda ortaya çıkar.
- Erken teslimler, sonraki teslimler için gereksinimleri çıkarmada prototip vazifesi görür.
- Projenin tümden batması riskini azaltır.
- Öncelikli gereksinimleri karşılayan sistem işlevleri daha çok test edilir.

DÖNGÜSEL ("SPIRAL") MODEL

- Süreç, geri dönüşümlü etkinlikler zinciri yerine döngüsel olarak ifade edilir.
- Her döngü, süreçteki bir aşamayı ifade eder.
- 4 sektörden oluşur:
  - Hedef belirleme: Aşamanın başarımı için somut hedefler belirlenir.
  - Risk değerlendirme ve azaltma: Riskler adreslenerek azaltıcı eylemler gerçekleştirilir.
  - Geliştirme ve doğrulama: Genel modeller içinden geliştirme için bir model seçilir.
  - Planlama: Proje gözden geçirilir ve bir sonraki aşama planlanır.
- Riskler süreç boyunca özel olarak ele alınır ve çözümlenir.
- Tanımlama ve tasarım gibi sabit aşamalar yoktur; her döngü ihtiyaca göre seçilir.

DÖNGÜSEL GELİŞTİRME – SEKTÖRLER



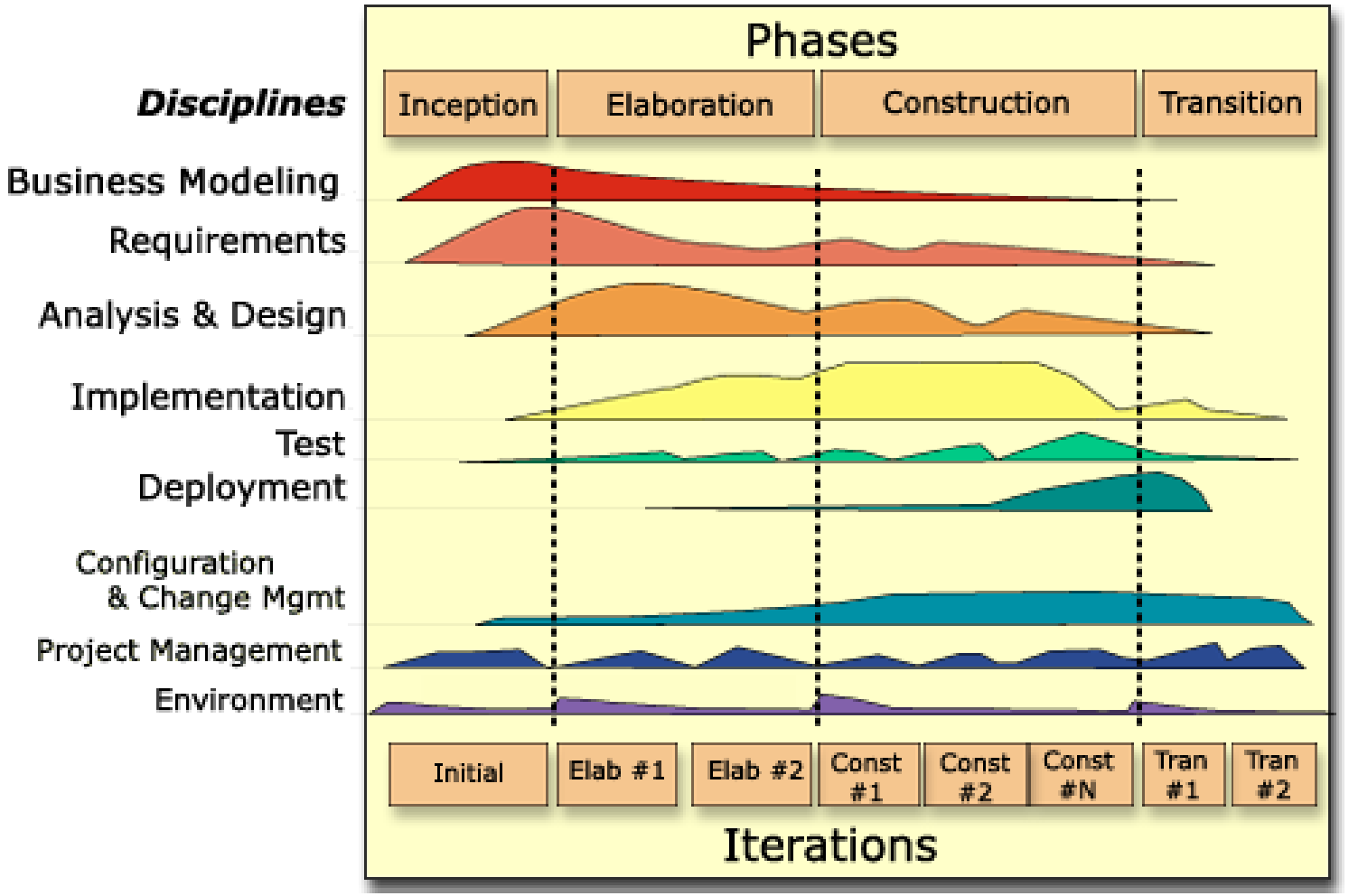
ÖRNEK BİR YAZILIM SÜREÇ MODELİ: TÜMLEŞİK SÜREÇ ("UNIFIED PROCESS")

- Bir süreç çatısıdır.
  - Tekrarlı ("iterative") ve artırımlı ("incremental")
  - "Use-case" esaslı
  - Mimari merkezli ("architecture centric")
  - Risk odaklı

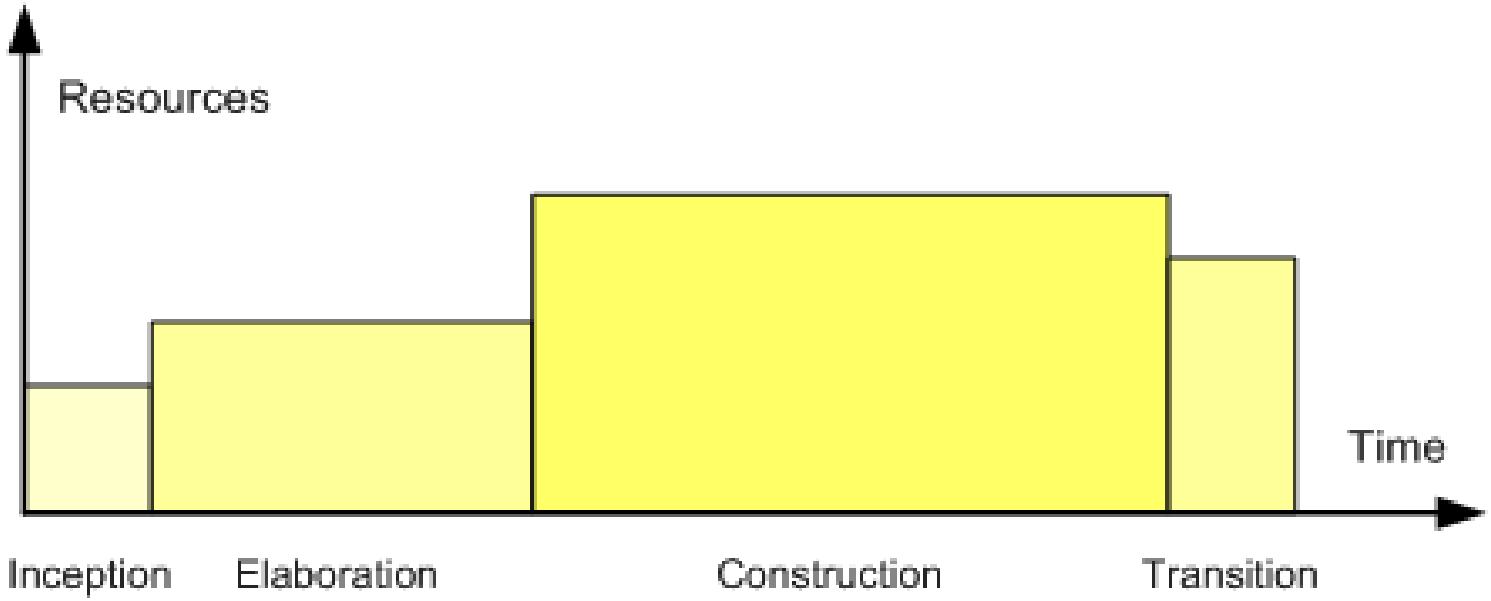
- Statik yapısına ek olarak, kurumların ve projelerinin özelliklerine göre uyarlanabilir bir yapıya sahiptir ("tailorable").

- Referans:
  - "*The Unified Software Development Process",* ISBN 0-201-57169-2, 1999.
    Ivar Jacobson, Grady Booch, James Rumbaugh.

"THE RATIONAL UNIFIED PROCESS" (RATIONAL SOFTWARE'IN TÜMLEŞİK SÜRECİ)



TÜMLEŞİK SÜREÇ – YAŞAM DÖNGÜSÜ



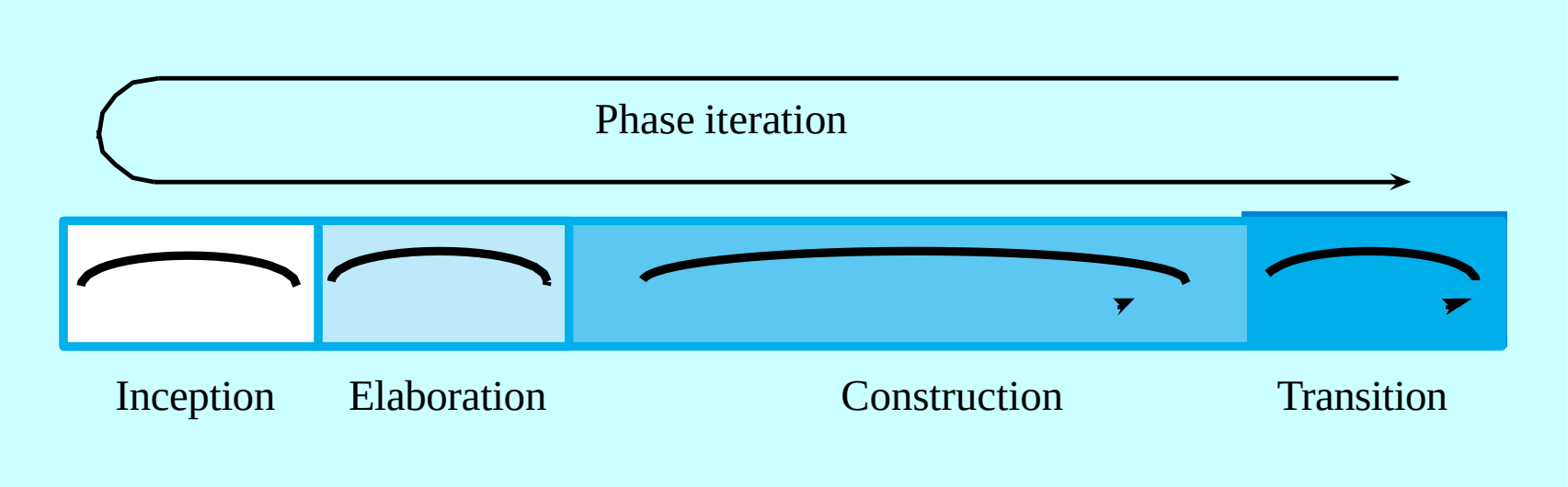
“Inception” – Projenin kapsamını tanımla ve iş durumunu (“business case”) geliştir

“Elaboration” – “Projeyi planla, özelliklerini tanımla, mimariyi dayanağı (“baseline”) oluştur

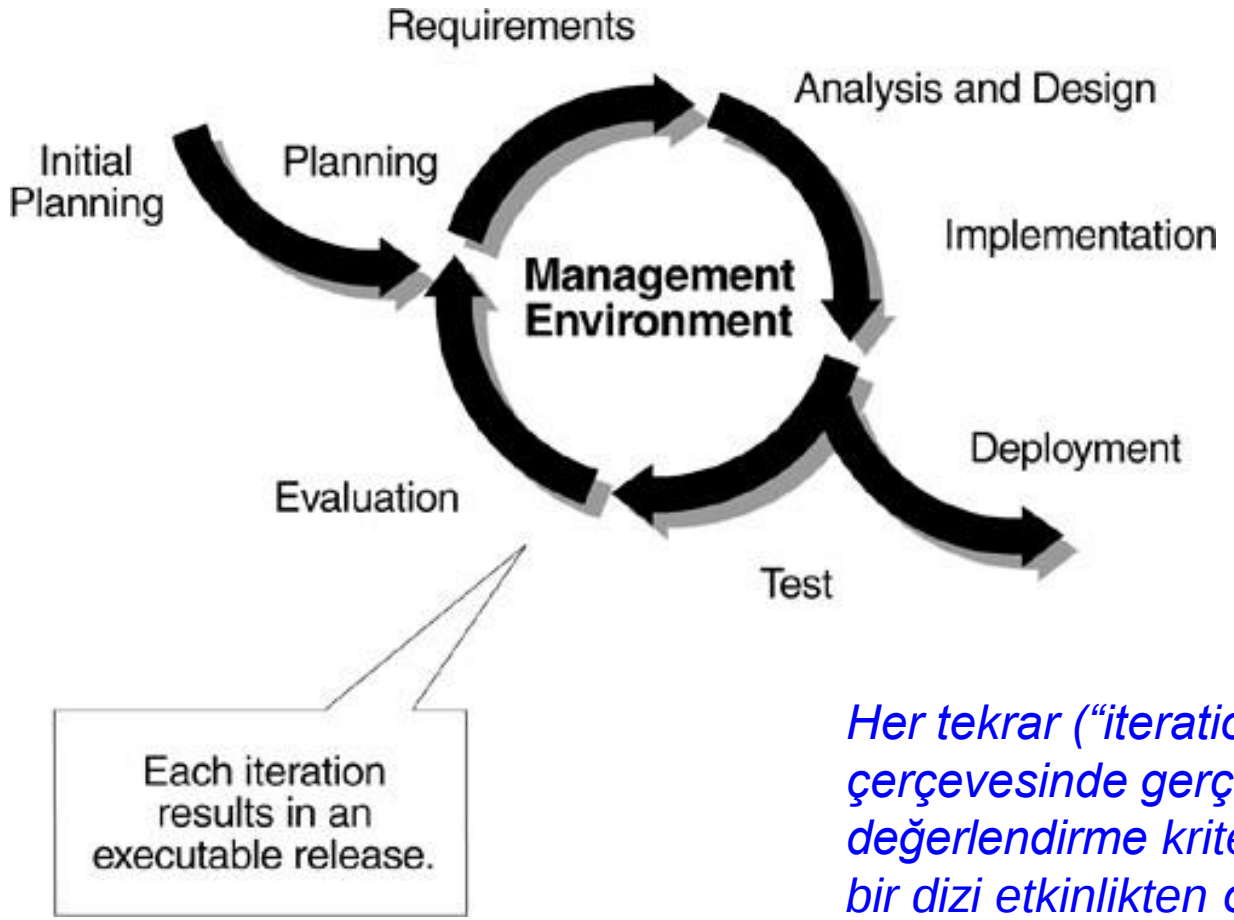
“Construction” – Ürünü gerçekleştir

“Transition” – Ürünü kullanıcılarına teslim et

TÜMLEŞİK SÜREÇ – AŞAMALAR

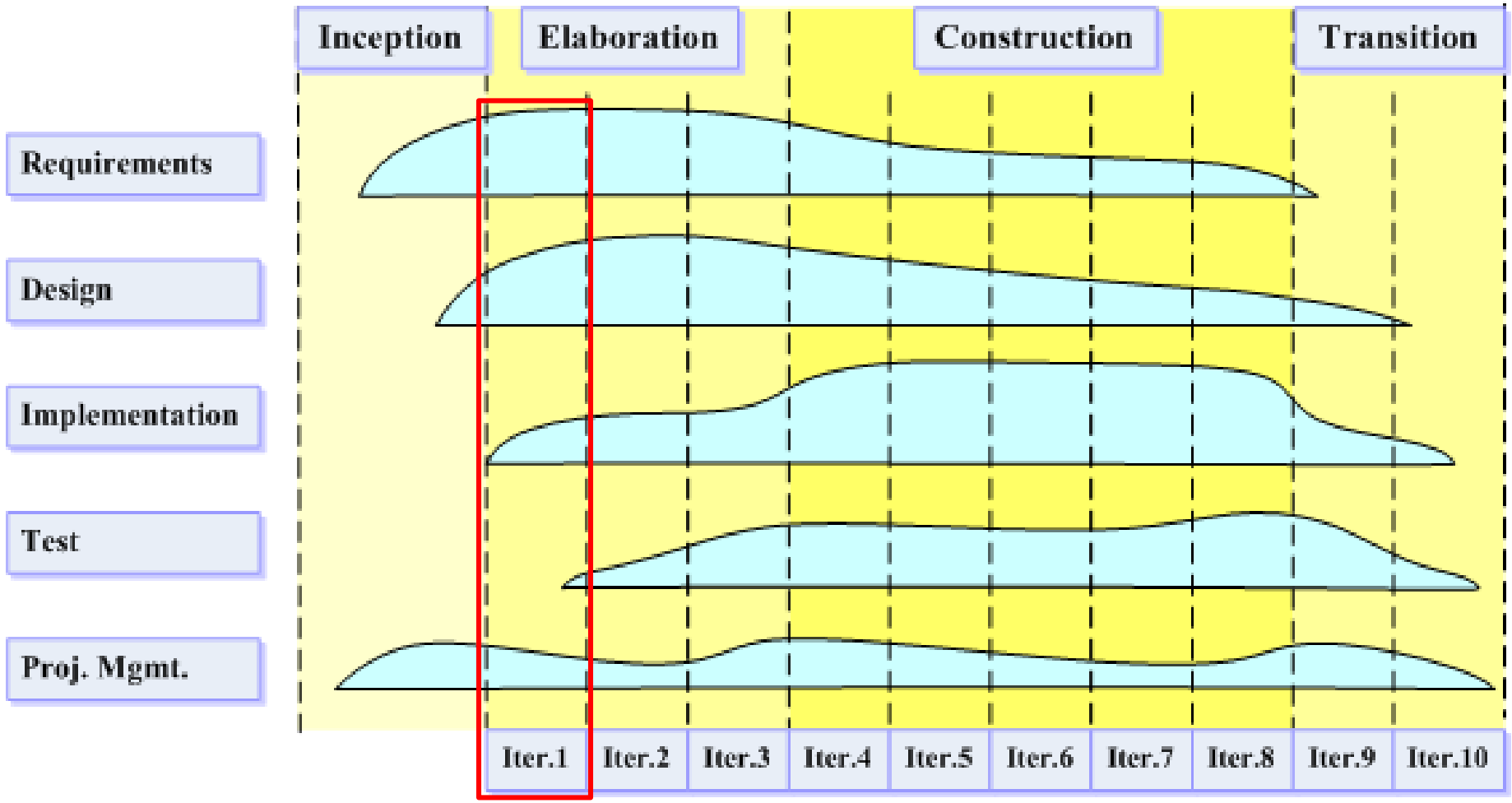


TEKRARLI GELİŞTİRME ("ITERATİVE DEVELOPMENT")



Her tekrar ("iteration"), tanımlı bir plan çerçevesinde gerçekleştirilen ve değerlendirme kriterleri tanımlanmış bir dizi etkinlikten oluşur.

TÜMLEŞİK SÜREÇ – YAPI



ÇEVİK (“AGILE”) YAZILIM SÜREÇ MODELLERİ

ÇEVİK ("AGİLE") YAZILIM SÜREÇ MODELLERİ

- Çevik modeller, mevcut geleneksel modellere alternatif olarak, 1990'larda ortaya çıkmaya başlamıştır.
  - Çevik: *Kolaylık ve çabuklukla davranan, tetik, atik*. [TDK]
  - 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 GELİŞTİRME MANİFESTOSU

- 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;

  - Bireyler ve aralarındaki etkileşim, kullanılan araç ve süreçlerden;
  - Çalışan yazılım, detaylı belgelerden;
  - Müşteri ile işbirliği, sözleşmedeki kesin kurallardan;
  - Değişikliklere uyum sağlayabilmek, mevcut planı takip etmekten;

daha önemli ve önceliklidir.

(Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas)

ÇEVİK YAZILIM GELİŞTİRME PRENSİPLERİ

- Çevik yazılım geliştirme manifestosu maddelerinin altında yatan temel prensipler şunlardır:
  - İlk öncelik, sürekli, kaliteli yazılım teslimatıyla müşteri memnuniyetini sağlamaktır.
  - Proje ne kadar ilerlemiş olursa olsun değişiklikler kabul edilir. Çevik yazılım süreçleri değişiklikleri müşteri avantajına dönüştürür.
  - Mümkün olduğunca kısa zaman aralıklarıyla (2–6 hafta arası) çalışan, kaliteli yazılım teslimatı yapılır.
  - Analistler, uzmanlar, yazılımcılar, testçiler, vs. tüm ekip elemanları bire bir iletişim halinde, günlük olarak birlikte çalışır.
  - İyi projeler motivasyonu yüksek bireyler etrafında kurulur. Ekip elemanlarına gerekli destek verilmeli, ihtiyaçları karşılanarak proje ile ilgili tam güvenilmelidir.

ÇEVİK YAZILIM GELİŞTİRME PRENSİPLERİ

- (Temel prensipler – devam)
  - Ekip içerisinde kaliteli bilgi akışı için yüz yüze iletişim önemlidir.
  - Çalışan yazılım, projenin ilk gelişim ölçütüdür.
  - Çevik süreçler mümkün olduğunca sabit hızlı, sürdürülebilir geliştirmeye önem verir.
  - Sağlam teknik alt yapı ve tasarım çevikliği arttırır.
  - Basitlik önemlidir.
  - En iyi mimariler, gereksinimler ve tasarımlar kendini organize edebilen ekipler tarafından yaratılır.
  - Düzenli aralıklarla ekip kendi yöntemlerini gözden geçirerek verimliliği arttırmak için gerekli iyileştirmeleri yapar.

ÇEVİK YAZILIM SÜREÇ MODELLERİ

- Uçdeğer Programlama (“Extreme Programming – XP”)
- Scrum
- Özellik Güdümlü Geliştirme (“Feature-Driven Development – FDD”)
- Çevik Tümleşik Süreç (“Agile Unified Process – AUP”)

UÇDEĞER PROGRAMLAMA ("EXTREME PROGRAMMİNG - XP")

- Uçdeğer programlama, Kent Beck tarafından 1999 yılında bir yazılım geliştirme disiplini olarak ortaya çıkarılmıştır.
- 4 prensip etrafında toplanır: Basitlik, İletişim, Geri bildirim, Cesaret
- 12 temel pratiğin birlikte uygulanmasını gerektirir:
  - Planlama oyunu
  - Kısa aralıklı sürümler
  - Müşteri katılımı
  - Yeniden yapılandırma ("refactoring")
  - Önce test ("test first")
  - Ortak kod sahiplenme
  - Basit tasarım
  - Metafor
  - Eşli programlama ("pair programming")
  - Kodlama standardı
  - Sürekli entegrasyon ("continuous integration")
  - Haftada 40 saat çalışma

SCRUM

- 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 iş 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.

ÖZELLİK GÜDÜMLÜ GELİŞTİRME ("FEATURE DRIVEN DEVELOPMENT - FDD")

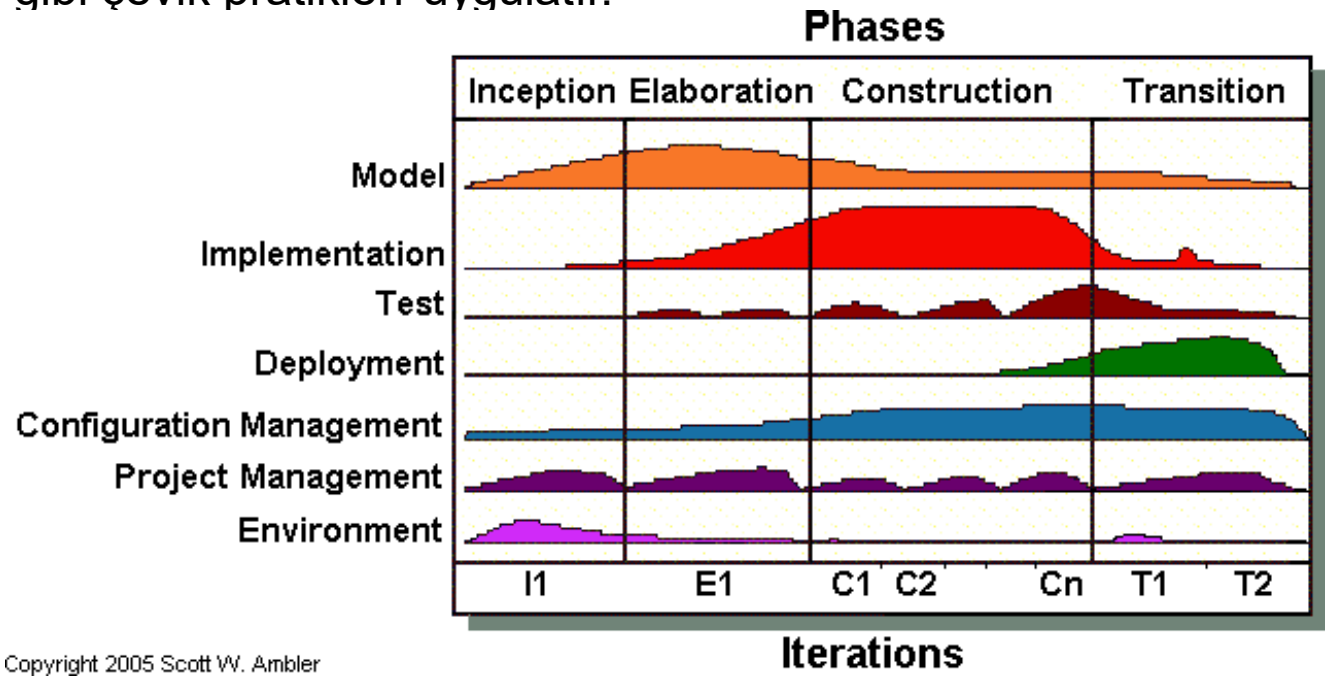
- Jeff De Luca ve Peter Coad tarafından 1997'de geliştirilen ve özellik tabanlı gerçekleştirimi esas alan bir modeldir.

- Süreç beş ana basamaktan oluşur; her bir basamak için çıkış şartları tanımlanır ve bunlara uyulur.
  - Genel sistem modelinin geliştirilmesi,
  - Özellik listesinin oluşturulması,
  - Özellik güdümlü bir planlama yapılması,
  - Özellik güdümlü tasarımın oluşturulması,
  - Özellik güdümlü geliştirmenin yapılması.

- Sisteme yeni bir özellik kazandırılmadan önce, detaylı bir tasarım çalışması yapılarak bu özelliği kapsayan mimari yapı oluşturulur.

ÇEVİK TÜMLEŞİK SÜREÇ ("AGILE UNIFIED PROCESS")

- Çevik Tümleşik Süreç (Agile Unified Process – AUP"); Rational Unified Process (RUP)'un, çevik yaklaşıma göre adapte edilerek basitleştirilmiş halidir.
- Test-güdümlü geliştirme ("test driven development – TDD"), çevik değişiklik yönetimi, veritabanını yeniden yapılandırma gibi çevik pratikleri uygulatır.
- RUP'dan farklı olarak, 7 adet disiplin içerir.



TARTIŞMA

- Hangi model (geleneksel ya da çevik) daha iyi?

- Hangi modeli ne zaman kullanalım?

ÖRNEK YAŞAM DÖNGÜSÜ: ATM PROJE PLANI

1.2	**İŞ GEREKSİNİMLERİNİN BELİRLENMESİ**
1.2.1	İş gereksinimlerinin incelenmesi
1.2.2	Iş gereksinimleri belgesinin geliştirilmesi
1.2.3	Iş gereksinimleri belgesinin gözden geçirilmesi
1.2.4	Iş gereksinimleri belgesinin günlenmesi
1.2.5	İş gereksinimleri belgesinin yapılandırma kontrolu altına alınması
1.2.6	İş gereksinimleri belgesi
1.3	**SİSTEM GEREKSİNİMLERİNİN VE MİMARİSİNİN BELİRLENMESİ**
1.3.1	**Sistem gereksinimleri analizi**
1.3.2	Sistem gereksinimleri belgesi
1.3.3	**Sistem mimarisinin belirlenmesi**
1.3.4	Sistem mimari dokumani
1.4	**DONANIM GELİŞTİRME**
1.5	**YAZILIM GELİŞTİRME**
1.5.1	**Çevrim-1: ATM Kart Yazılım Paketi Geliştirme**
1.5.1.1	**Çevrim-1: Yazılım gereksinimlerinin belirlenmesi**
1.5.1.1.1	Yazılım gereksinimleri analizi
1.5.1.1.2	Yazılım gereksinimleri belgesinin hazırlanması
1.5.1.1.3	Yazılım gereksinimleri belgesinin gozden gecirilmesi
1.5.1.1.4	Yazılım gereksinimleri belgesinin gunlenmesi
1.5.1.1.5	Yazılım gereksinimleri belgesinin yapılandırma kontrolu altına alınması
1.5.1.1.6	Yazılım gereksinimleri belgesi
1.5.1.2	**Çevrim-1: Yazılım tasarımının yapılması**
1.5.1.2.1	Yazılım ust duzey tasarımının yapılması
1.5.1.2.2	Yazılım ust duzey tasarımının gozden gecirilmesi
1.5.1.2.3	Yazılım ust duzey tasarımının gunlenmesi
1.5.1.2.4	Yazılım detay tasarımının yapılması
1.5.1.2.5	Yazılım detay tasarımının gozden gecirilmesi
1.5.1.2.6	Yazılım detay tasarımının gunlenmesi
1.5.1.2.7	Yazılım tasarımının belgelendirilmesi
1.5.1.2.8	Yazılım tasarım belgesinin gozden gecirilmesi
1.5.1.2.9	Yazılım tasarım belgesinin gunlenmesi
1.5.1.2.10	Yazılım tasarım belgesinin yapılandırma kontrolu altına alınması
1.5.1.2.11	Yazılım tasarım belgesi
1.5.1.3	**Çevrim-1: Yazılım gerçekleştirme ve birim test**
1.5.1.3.1	Yazılım kodlama
1.5.1.3.2	Yazılım kodu
1.5.1.3.3	Yazılım birim test
1.5.1.3.4	Yazılım birim test raporu
1.5.1.4	Çevrim-1: Yazılım tümleştirme ve test
1.5.1.5	Yazılım tümleştirme test raporu
1.5.1.6	Yazılım günleme
1.5.1.7	Çevrim-1: Yazılım işlevsel test
1.5.1.8	Yazılım işlevsel test raporu
1.5.1.9	Yazılım günleme

ISO/IEC 12207 (IEEE STD 12207-2008):

Yazılım Yaşam Döngüsü Süreçlerinin Detayı İçin
Sıklıkla Temel Alınan Bir Standart

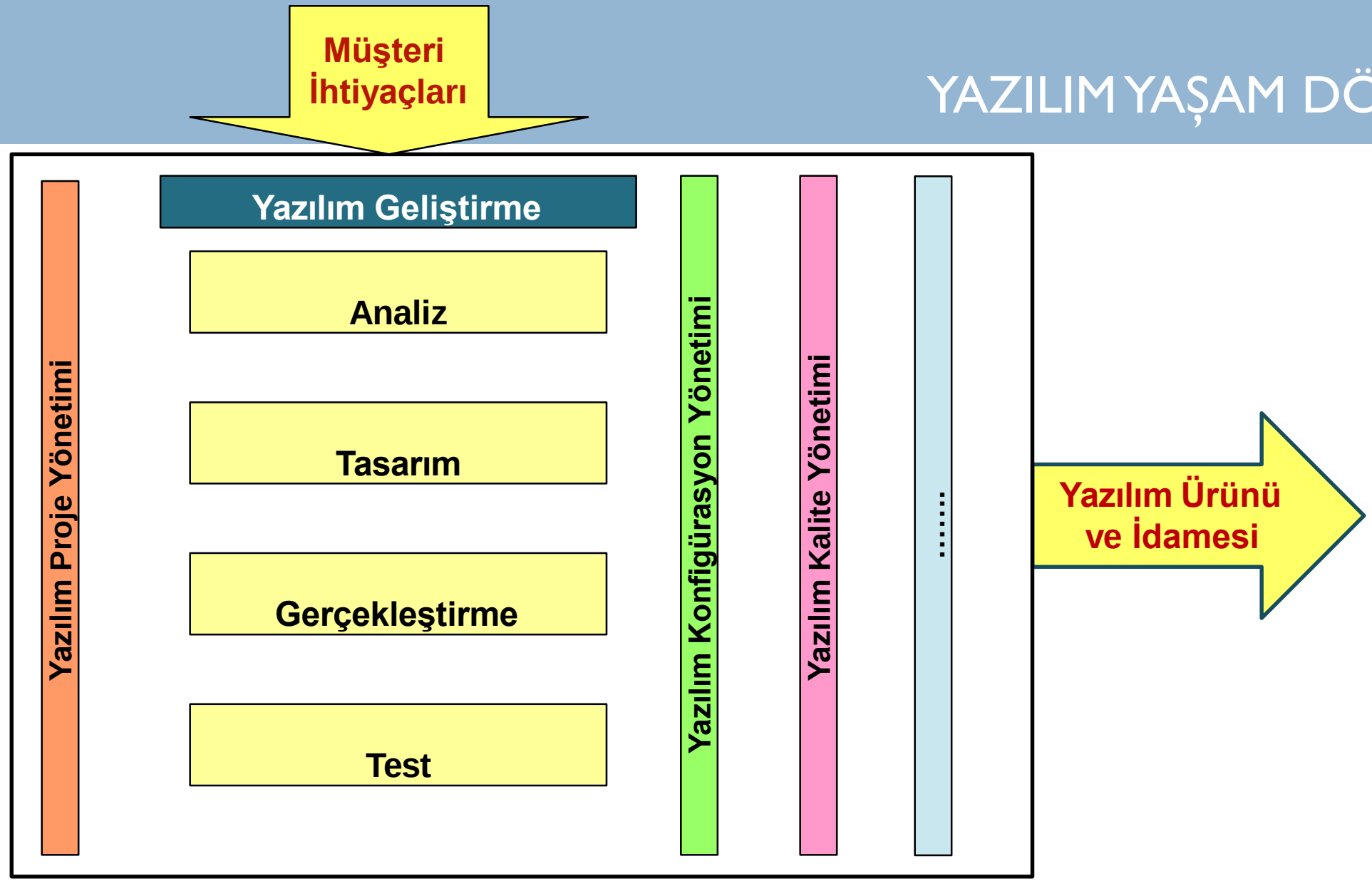
YAZILIM YAŞAM DÖNGÜSÜ

- Girdi: İş gereksinimleri (iş alanı bilgisini gerektirir)

- Yazılım ürününün geliştirilmesi ve yönetilmesi için izlenmesi gereken adımların bütünüdür
  - Analiz, tasarım, kodlama, test, ...
  - Proje yönetimi, kalite yönetimi, yapılandırma yönetimi, ...
- Yazılım ürününe mühendislik etkinliklerinin uygulanmasına olanak sağlar
  - Örnek: ISO/IEC 12207 Yazılım Yaşam Döngüsü Süreçleri

- Çıktı: Yazılım sistemi (çözüm alanı bilgisini gerektirir)

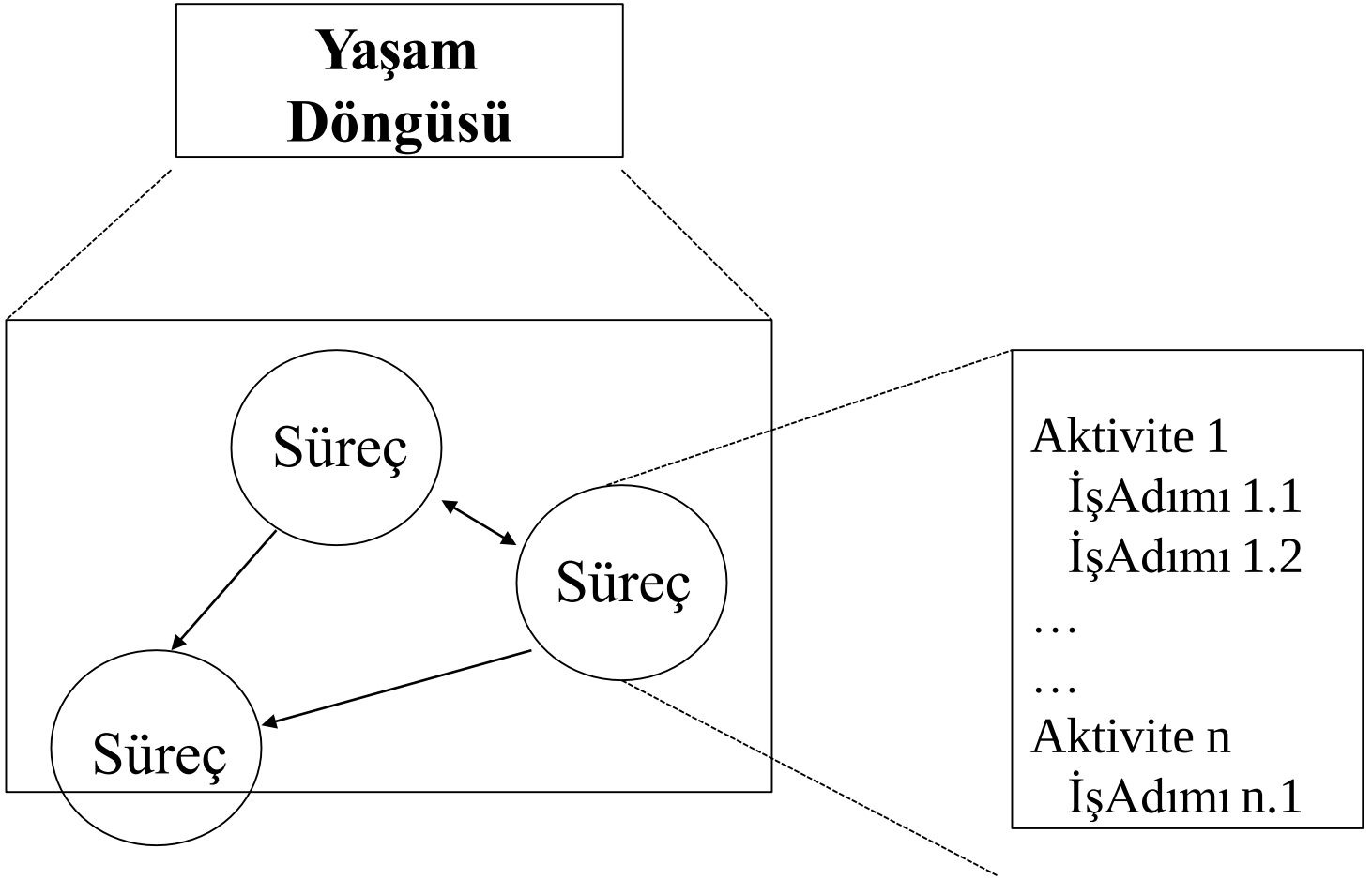
YAZILIM YAŞAM DÖNGÜSÜ



ISO/IEC 12207 (IEEE STD 12207-2008)

- Sistem ve yazılım mühendisliği – Yazılım yaşam döngüsü süreçleri
- 1995'den bu yana sektörde yaygın olarak kullanılmaktadır.
- Kurumun veya projenin ihtiyaçlarına göre uyarlanabilir.
  - Standartta tanımlanan uyarlama kuralları ve önerileri dikkate alınmalıdır.
  - Tam uyumluluk / Seçerek uyarlama

ISO/IEC 12207 GENEL YAPISI



ISO/IEC 12207 KAPSAMI

- Daha büyük bir sistemin parçası olarak veya tek başına *yazılım ürünü veya hizmeti* için, bir grup yaşam döngüsü süreci ile bunların altındaki etkinlikleri ve görevleri tanımlar.

- Yazılım ürününün veya hizmetinin satınalınması, tedariği, geliştirilmesi, işletilmesi, bakımı ve elden çıkarılması boyunca referans alınabilir.

- ISO/IEC 15288 standardı ile yapı, terimler ilişkili süreçler açılarından tam olarak uyumludur.
  - ISO/IEC 15288-2008: Sistem ve yazılım mühendisliği – Sistem yaşam döngüsü süreçleri

ISO/IEC 12207 KISITLARI

- Herhangi bir yaşam döngüsü modelinin, geliştirme yaklaşımının veya yönteminin kullanılmasını öngörmez.

- Her sürecin beklenen çıktıları üreterek amacına ulaşması için uygulanacak etkinlikleri ve görevleri tanımlar.
  - Süreci, yöntem ve teknikleri içerecek şekilde detaylandırmaz.
  - Sürecin amacına ulaşması için neler yapılması gerektiğini tanımlar.
  - Süreç kapsamındaki etkinlikler ve görevler doğal bir sırada tanımlanmış olsa da bunlar, sürecin nasıl uygulanacağı konusunda yönlendirme vermez.

- Bilgi ürünlerinin (belgelerin) adı, biçimi, içeriği ve materyali ile ilgili detayları tanımlamaz.

Yazılım yaşam döngüsü süreçleri (15207)

Anlaşma Süreçleri

- Satınalma Süreci (6.1.1)
- Tedarik Süreci (6.1.2)

Proje-Sağlayıcı Süreçler

- Yaşam Döngüsü Modeli Yönetimi Süreci (6.2.1)
- Altyapı Yönetimi Süreci (6.2.2)
- Proje Portföy Yönetimi Süreci (6.2.3)
- İnsan Kaynakları Yönetimi Süreci (6.2.4)
- Kalite Yönetimi Süreci (6.2.5)

Proje Süreçleri

- Proje Planlama Süreci (6.3.1)
- Proje Değerlendirme ve Kontrol Süreci (6.3.2)
- Karar Yönetimi Süreci (6.3.3)
- Risk Yönetimi Süreci (6.3.4)
- Konfigürasyon Yönetimi Süreci (6.3.5)
- Bilgi Yönetimi Süreci (6.3.6)
- Ölçme Süreci (6.3.7)

Teknik Süreçler

- Paydaş Gereksinimlerini Tanımlama Süreci (6.4.1)
- Sistem Gereksinimleri Analizi Süreci (6.4.2)
- Sistem Mimari Tasarımı Süreci (6.4.3)
- Gerçekleştirme Süreci (6.4.4) → 7.1.1
- Sistem Tümleştirme Süreci (6.4.5)
- Sistem Nitelik Testi Süreci (6.4.6)
- Yazılım Kuruluş Süreci (6.4.7)
- Yazılım Kabul Destek Süreci (6.4.8)
- Yazılımı İşletme Süreci (6.4.9)
- Yazılım Bakım Süreci (6.4.10)
- Yazılımı Elden Çıkarma Süreci (6.4.11)

Yazılım yaşam döngüsü süreçleri

Yazılım Gerçekleştirme Süreçleri

- Yazılım Gerçekleştirme Süreci (7.1.1)
- Yazılım Gereksinimleri Analizi Süreci (7.1.2)
- Yazılım Mimari Tasarımı Süreci (7.1.3)
- Yazılım Detaylı Tasarım Süreci (7.1.4)
- Yazılım İnşa Süreci (7.1.5)
- Yazılım Tümleştirme Süreci (7.1.6)
- Yazılım Nitelik Testi Süreci (7.1.7)

Yazılım Destek Süreçleri

- Yazılım Belgelendirme Yönetimi Süreci (7.2.1)
- Yazılım Konfigürasyon Yönetimi Süreci (7.2.2)
- Yazılım Kalite Güvence Süreci (7.2.3)
- Yazılım Doğrulama Süreci (7.2.4)
- Yazılım Geçerleme Süreci (7.2.5)
- Yazılım Gözden Geçirme Süreci (7.2.6)
- Yazılım Denetleme Süreci (7.2.7)
- Yazılım Problem Çözümleme Süreci (7.2.8)

Yazılımı Tekrar Kullanma Süreçleri

- Alan Mühendisliği Süreci (7.3.1)
- Tekrar Kullanma Varlık Yönetimi Süreci (7.3.2)
- Tekrar Kullanma Program Yönetimi Süreci (7.3.3)

Sistem Kapsamlı Süreçler

Anlaşma Süreçleri

- Satınalma Süreci (6.1.1)
- Tedarik Süreci (6.1.2)

Proje-Sağlayıcı Süreçler

- Yaşam Döngüsü Modeli Yönetimi Süreci (6.2.1)
- Altyapı Yönetimi Süreci (6.2.2)
- Proje Portföy Yönetimi Süreci (6.2.3)
- İnsan Kaynakları Yönetimi Süreci (6.2.4)
- Kalite Yönetimi Süreci (6.2.5)

Proje Süreçleri

- Proje Planlama Süreci (6.3.1)
- Proje Değerlendirme ve Kontrol Süreci (6.3.2)
- Karar Yönetimi Süreci (6.3.3)
- Risk Yönetimi Süreci (6.3.4)
- Konfigürasyon Yönetimi Süreci (6.3.5)
- Bilgi Yönetimi Süreci (6.3.6)
- Ölçme Süreci (6.3.7)

Teknik Süreçler

- Paydaş Gereksinimlerini Tanımlama Süreci (6.4.1)
- Gereksinim Analizi Süreci (6.4.2)
- Mimari Tasarım Süreci (6.4.3)
- Gerçekleştirme Süreci (6.4.4)
- Tümleştirme Süreci (6.4.5)
- Doğrulama Süreci (6.4.6)
- Geçiş Süreci (6.4.7)
- Geçerleme Süreci (6.4.8)
- İşletme Süreci (6.4.9)
- Bakım Süreci (6.4.10)
- Elden Çıkarma Süreci (6.4.11)

Yazılıma Özel Süreçler

Yazılım Gerçekleştirme Süreçleri

- Yazılım Gerçekleştirme Süreci (7.1.1)
- Yazılım Gereksinimleri Analizi Süreci (7.1.2)
- Yazılım Mimari Tasarımı Süreci (7.1.3)
- Yazılım Detaylı Tasarım Süreci (7.1.4)
- Yazılım İnşa Süreci (7.1.5)
- Yazılım Tümleştirme Süreci (7.1.6)
- Yazılım Nitelik Testi Süreci (7.1.7)

Yazılım Destek Süreçleri

- Yazılım Belgelendirme Yönetimi Süreci (7.2.1)
- Yazılım Konfigürasyon Yönetimi Süreci (7.2.2)
- Yazılım Kalite Güvence Süreci (7.2.3)
- Yazılım Doğrulama Süreci (7.2.4)
- Yazılım Geçerleme Süreci (7.2.5)
- Yazılım Gözden Geçirme Süreci (7.2.6)
- Yazılım Denetleme Süreci (7.2.7)
- Yazılım Problem Çözümleme Süreci (7.2.8)

Yazılımı Tekrar Kullanma Süreçleri

- Alan Mühendisliği Süreci (7.3.1)
- Tekrar Kullanma Varlık Yönetimi Süreci (7.3.2)
- Tekrar Kullanma Program Yönetimi Süreci (7.3.3)

REFERANSLAR

- Software Engineering (10th. Ed.); Ian Sommerville; 2015.
- Guide to Software Engineering Body of Knowledge (v3); 2014.
- Hacettepe Üniversitesi BBS-651, A. Tarhan, 2019.