- Giriş -

BBS-651 Yazılım Mühendisliği

Hafta #1 (6 Ekim 2010)

İçerik

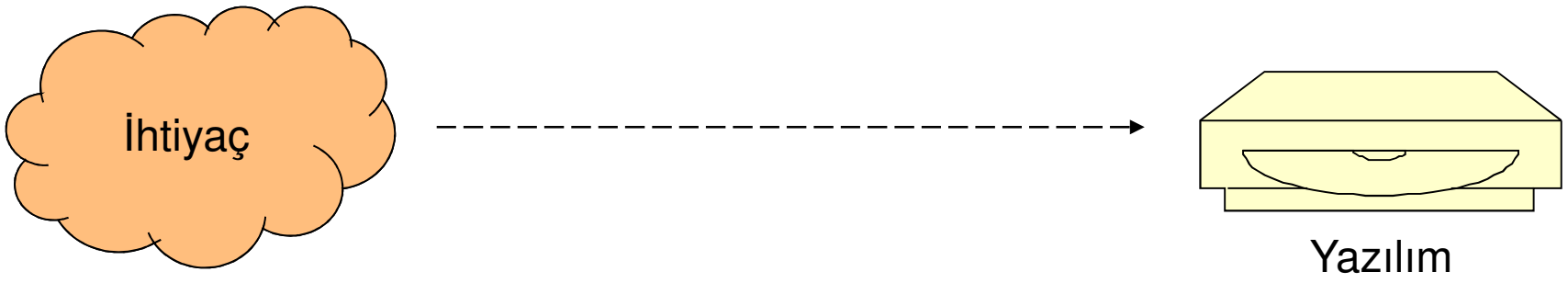
- Yazılım Mühendisliği
  - Tanımlar, tarihçe, kapsam
- Yazılım Süreç Modelleri
  - Yazılım süreci ve süreç modeli nedir?
  - Geleneksel modeller
    - Çağlayan (“Waterfall”), Evrimsel (“Evolutionary”), Bileşen-Tabanlı (“Component-Based”), Artırımlı (“Incremental”), Döngüsel (“Spiral”)
    - Rational Software’in Tümleşik Süreci (“Rational Unified Process”)

Yazılım Mühendisliği

- Tanımlar, Tarihçe, Kapsam -

Yazılım Nedir?

- Bilgisayar programları ile bunlarla ilişkili yapılandırma ve belgelerin bütünüdür.
  - Genel yazılım
    - Özellikleri pazardaki genel ihtiyaca göre belirlenerek geliştirilir.
    - “Generic software”, “commercial-off-the-shelf (COTS) software”
    - Örnek: MS Office yazılımı
  - Müşteriye özel yazılım
    - Özellikleri belirli bir müşterinin ihtiyacına göre belirlenerek geliştirilir.
    - “Custom software”
    - Örnek: Hava trafiği kontrol yazılımı



Yazılımın Gelişimi

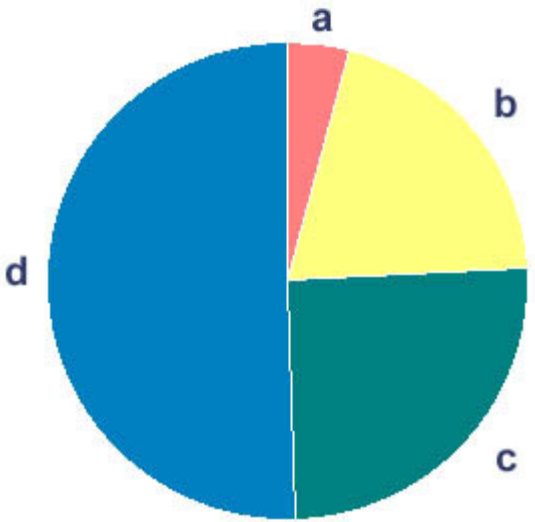
- 1940'lar:
  - Elle yazılan basit makine kodları
- 1950'ler:
  - Üretkenlik ve kaliteyi arttırmak için yazılan makro birleştiriciler ve yorumlayıcılar
  - İlk kuşak derleyiciler
- 1960'lar:
  - Üretkenlik ve kaliteyi arttırmak için yazılan ikinci kuşak derleyiciler
  - Yazılım Mühendisliği NATO Konferansı, 1968: "Yazılım mühendisliği" kavramının tartışılması
  - İlk büyük yazılım projeleri (1000 programcı)
  - Büyük iş alanları için ana-bilgisayarlar ve ticari yazılımlar
- 1970'ler:
  - UNIX ve kod kütüphaneleri için araçlar
  - Mini-bilgisayarlar ve küçük iş alanları için yazılımlar
- 1980'ler:
  - Kişisel bilgisayarlar ve iş istasyonları
  - Ticari yazılımlar
- 1990'lar:
  - Avuç-içi bilgisayarlar ve web teknolojileri
  - Teknolojideki gelişmeler sebebiyle düşen fiyatlar ve karmaşıklaşan iş talepleri
- 2000'ler:
  - Globalleşme sebebiyle artan talepler
  - Bütünleşik geliştirme ortamları

Yazılım Krizi

- 1960'lardan itibaren yazılım ürünlerine artan talepler karşısında otomasyonu da içeren değişik çözümler uygulandıysa da yeterli üretim kapasitesine erişilemedi. Bu yetersizlik, "yazılım krizi" söylemiyle dünya literatüründe yerini aldı.

- Yazılım krizinin temel sebebi, üstel olarak artan talebe karşılık doğrusal hızla artış gösteren üretim kapasitesidir. Diğer yandan yazılım projelerindeki başarısızlık oranı da şaşırtıcı derecede yüksek olup bu krizi beslemektedir.

Yazılım ürününe arz ve talep artışları



a: başarılı

b: büyük miktarda değişikliklerden sonra kullanılabilen

c: çok büyük değişikliklerden sonra kısmen kullanılabilen

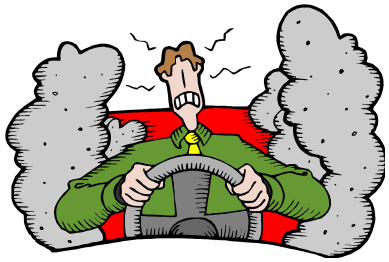
d: parası ödendiği halde kullanılamayan

Toplam maliyete göre yazılım başarısı

Yazılım Ürünlerine Talep

- Yazılım ürünlerine olan talep ve yazılım ürünlerinden beklentiler çok hızlı artıyor
  - Boeing 777
    - A.B.D. ve Japonya'da 1700 iş istasyonu
    - 4,000,000 Kod Satırı
    - "Kanatları olan yazılım"
  - Beyaz eşyalar, cep telefonları, otomobiller
  - Akıllı ev ve ofis sistemleri
  - …

Bazı Dehşet Hikayeleri



- Denver havaalanı otomatik bagaj sistemi
  - Açılış 2 yıl gecikti
  - $27 milyon maliyet aşımı
  - $360 milyon gecikme maliyeti
- Hava trafik kontrol (FAA modernizasyon)
  - $5.6 milyar maliyet aşımı
  - 8 yıl gecikme
  - 4 sistemden 2 tanesi iptal edildi, üçüncü sistemin gereksinimlerinin %52'si karşılandı
- Comanche Helikopteri
  - 10 yılda maliyeti 10 kat arttı, $34.4 milyon
  - Gereksinimleri %74 azaltıldı

Ürün Büyüklüğü – Başarı İlişkisi

Kestirim (C Satır)	Önce	Zamanında	Gecikme	İptal
13.000	6.06%	74.77%	11.83%	7.33%
130.000	1.24%	60.76%	17.67%	20.33%
1.300.000	0.14%	28.03%	23.83%	48.00%
13.000.000	0.0%	13.67%	21.33%	65.00%

Referans:
"Patterns of Software Failure and Success", C. Jones

Büyük kapsamlı yazılım ürünleri için, mühendislik yaklaşımı zorunlu hale gelmiştir!

Yazılım Mühendisliği Nedir?

- Yazılım üretiminin tüm etkinliklerini kapsayan mühendislik disiplinidir.
  - Sistem tanımlama gibi erken aşamalardan sistemin kullanımı esnasındaki bakımına kadar uzanan etkinlikleri kapsar.
  - Sadece teknik etkinlikleri değil, aynı zamanda yönetim etkinliklerini de içerir.

Mühendislik Yaklaşımı !

- Yazılım Mühendisliği; yazılım ürününün geliştirilmesi, işletilmesi ve bakımı için uygulanan; sistematik, disiplinli ve ölçülebilir yaklaşımdır.

  [IEEE, 1990]

  - Mühendislik, herhangi bir bilim alanındaki bilgi birikimini sistematik olarak pratiğe geçirmeyi hedefler; bilimi ve matematiği kullanır.
    - Yönetim parametreleri: İşlev, maliyet, zaman
    - Kalite parametreleri: Dayanıklılık, bakım kolaylığı, güvenlik, kullanım kolaylığı, vb.

  - Tekrarlanabilir başarılar için mühendislik yaklaşımı şarttır.
    - Mühendislik öğretisi ile bir yöntem uygulandığında, benzer sonuçları her zaman elde etme güvenliği vardır.

Yazılım Mühendisliği - Bazı Tanımlar (1)

- "The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation to develop, operate and maintain them."

  [Boehm, 1976]

- "... the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates"

  [Fairley, 1985]

- "The application of a systematic, disciplined, *quantifiable* approach to the development, operation, and maintenance of software"

  [IEEE, 1990]

Yazılım Mühendisliği - Bazı Tanımlar (2)

- Software engineering is concerned with the definition, refinement and evaluation of principles, methods, techniques and tools to support:
  - Individual aspects of software development and maintenance (design, coding, etc.)
  - Planning of software development projects
  - Performing development, project management and quality assurance activities according to the plan
  - Assessing the performance of the development and improving products, methods, techniques and tools.

[Rombach and Verlage, 1995]

Yazılım Mühendisliği Neleri Kapsar?

- Yazılım Gereksinim Analizi
- Yazılım Tasarım
- Yazılım Gerçekleştirme ⇐ *Programlama*
- Yazılım Test
- Yazılım Bakım
- Yazılım Mühendislik Yönetimi
- Yazılım Konfigürasyon Yönetimi
- Yazılım Mühendisliği Süreçleri
- Yazılım Mühendisliği Araç ve Yöntemleri
- Yazılım Kalitesi

Yazılım Geliştirme Kapsamı

["Guide to the Software Engineering Body of Knowledge", 2004]

“İyi Yazılım” Ne Demektir?

- Yazılım ürünü, müşterinin beklediği işlevsellik ve performansın yanı sıra, aşağıdaki özellikleri de taşımalıdır.
  - Bakım-yapılabilirlik (“maintainability”)
    - Yazılım, değişen ihtiyaçlara göre değişebilir olmalıdır.
  - Güvenilirlik (“dependability”)
    - Yazılım, güvenilir olmalıdır.
  - Etkinlik (“efficiency”)
    - Yazılım, sistem kaynaklarının israfına sebep olmamalıdır.
  - Kabul-edilebilirlik (“acceptability”)
    - Yazılım, kullanıcıları tarafından kabul edilebilmelidir.

      (Diğer bir deyişle; anlaşılabilir ve kullanılabilir olmalı, diğer sistemlerle uyumlu çalışabilmelidir.)

Bilgisayar Bilimleri ve Yazılım Mühendisliği Arasındaki İlişki Nedir?

- Bilgisayar Bilimleri; bilgisayar ve yazılım sistemlerinin temelinde yatan teoriler ve yöntemlerle ilgilenir.
  - Algoritmalar ve veri yapıları, programlama dilleri, mimari, bilimsel hesaplama, işletim sistemleri, bilgi ve veri yönetimi, grafik görüntüleme ve çoklu-ortam, bilgisayar ağları, akıllı sistemler, vb.

    (Fizik bilimlerinin, Elektrik ve Elektronik Mühendisliğinin temelinde yattığı gibi)

- Yazılım Mühendisliği; yazılım üretmenin pratik problemleriyle ilgilenir.
  - Bunu yaparken Bilgisayar Bilimlerinin sunduğu kavramsal altyapıyı kullanır.
  - Bir yazılım mühendisinin, ilişkili uygulama alanına göre (örneğin veri-işleme, animasyon, vb.), altta yatan bilgisayar bilimine hakim olması zorunludur.

Sistem Mühendisliği ve Yazılım Mühendisliği Arasındaki İlişki Nedir?

- Sistem Mühendisliği; yazılımın önemli rol oynadığı, karmaşık bilgisayar sistemlerinin geliştirilmesi ve idamesi ile ilgilenir.
  - Örnek: ATM sistemi
  - Sistem Mühendisliği; sistemin genel çatısıyla ilgilidir.
    - Sistemin kullanılacağı alana ilişkin süreçlerin analiz edilmesi, sistem gereksinimlerinin tanımlanması, sistem mimarisinin oluşturulması, sistem bileşenlerinin tümleştirilmesi gibi aşamaları içerir.
  - Söz konusu bilgisayar sistemi; donanım, ağ, yazılım bileşenlerinden oluşur. Sistem Mühendisliği, bileşenlere ilişkin mühendislik etkinliklerinin detaylarıyla pek ilgilenmez.
    - Bu tür sistemler için Yazılım Mühendisliği (kapsamındaki tüm etkinliklerle birlikte), Sistem Mühendisliği altında ve onun bir parçası olarak uygulanır.
  - Sistem Mühendisliği, Yazılım Mühendisliğine kıyasla çok daha eski bir mühendislik disiplinidir.

Donanım Mühendisliği ve Yazılım Mühendisliği Arasındaki Farklar Nelerdir?

- Donanım mühendislikleri ile yazılım mühendisliğinin en belirgin farkı ürünlerindedir.
  - Yazılım ürünü diğer mühendislik ürünlerine oranla daha soyuttur.
  - Yazılım projesi geliştirme ile sonlanırken donanım projelerinde ek olarak imalat safhası vardır.
  - Seri üretim, yazılım geliştirme içerisinde neredeyse hiçten ibarettir.
  - Donanım ürünleri kullanıldıkça aşınır; yazılım ürünlerinde ise aşınma olmaz. Yalnızca baştan beri gizli bulunan hatalar, yazılım kullanıldıkça ortaya çıkar.

- Donanım mühendisliklerinde maliyet odağı seri üretim ve yıpranmayken, yazılım mühendisliğinde maliyet odağı geliştirmedir.

Yazılım Mühendisliğinde Maliyetlerin Dağılımı Nedir?

- Kabaca söylersek, yazılım maliyetinin;

  %60 : Geliştirme maliyeti,

  %40 : Test maliyetidir (test maliyetine bulunan hataları düzeltmenin maliyeti de dahildir.)

- Müşteriye özel üretilen yazılımlar için idame (bakım) maliyeti, geliştirme maliyetinin birkaç katına çıkmaktadır.

Yazılım Mühendisliğinde Hedefler / Zorluklar

- Heterojenlik (“heterogeneity”)
  - Yazılım geliştirme için, heterojen platformları ve çalıştırma ortamlarını destekleyecek teknikler geliştirmek
- Teslim (“delivery”)
  - Yazılımın zamanında teslimi için teknikler geliştirmek
- Güven (“trust”)
  - Yazılımın kullanıcıları tarafından güvenilebileceğini gösteren teknikler geliştirmek

Yazılım Mühendisliğinde Profesyonel ve Etik Gereklilik

- Yazılım mühendisliği, teknik yetkinliğin uygulanmasının yanında, aşağıdaki konularda sorumluluk gerektirir:
  - Gizlilik
    - Örnek: Çalışanların ve müşterilerin gizlilik haklarına saygı göstermek
  - Rekabet
    - Örnek: Kendi yetkinliğiniz dışında iş kabul etmemek
  - Özlük hakları
    - Örnek: Patent, mülkiyet, vb. haklarına dikkat etmek
  - Bilgisayarın amaç-dışı kullanımı
    - Örnek: Başkasına ait bir makinede oyun oynayarak virüs bulaştırmamak

Bir Meslek Olarak Yazılım Mühendisliği

- Sektörde çalışanların yaklaşık yarısı “Bilgisayar Mühendisliği” lisans eğitimine sahip
  - Alana göre farklılık gösteren lisans dereceleri var
    - Örneğin; Yönetim Bilgi Sistemleri uygulamaları için İşletme lisansı, gömülü uygulamalar için Elektrik ve Elektronik Mühendisliği lisansı
- Yazılım mühendisliği geniş kesimlerce bir meslek olarak algılanmakta
- Üniversitelerin Yazılım Mühendisliği lisans ve yüksek lisans programları var

Yazılım Süreç Modelleri

Yazılım Süreç Modelleri

- Yazılım geliştirmenin bahsedilen zorluklarıyla başedebilmek için, geliştirmeyi sistematik hale getirmeyi hedefleyen çeşitli 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)

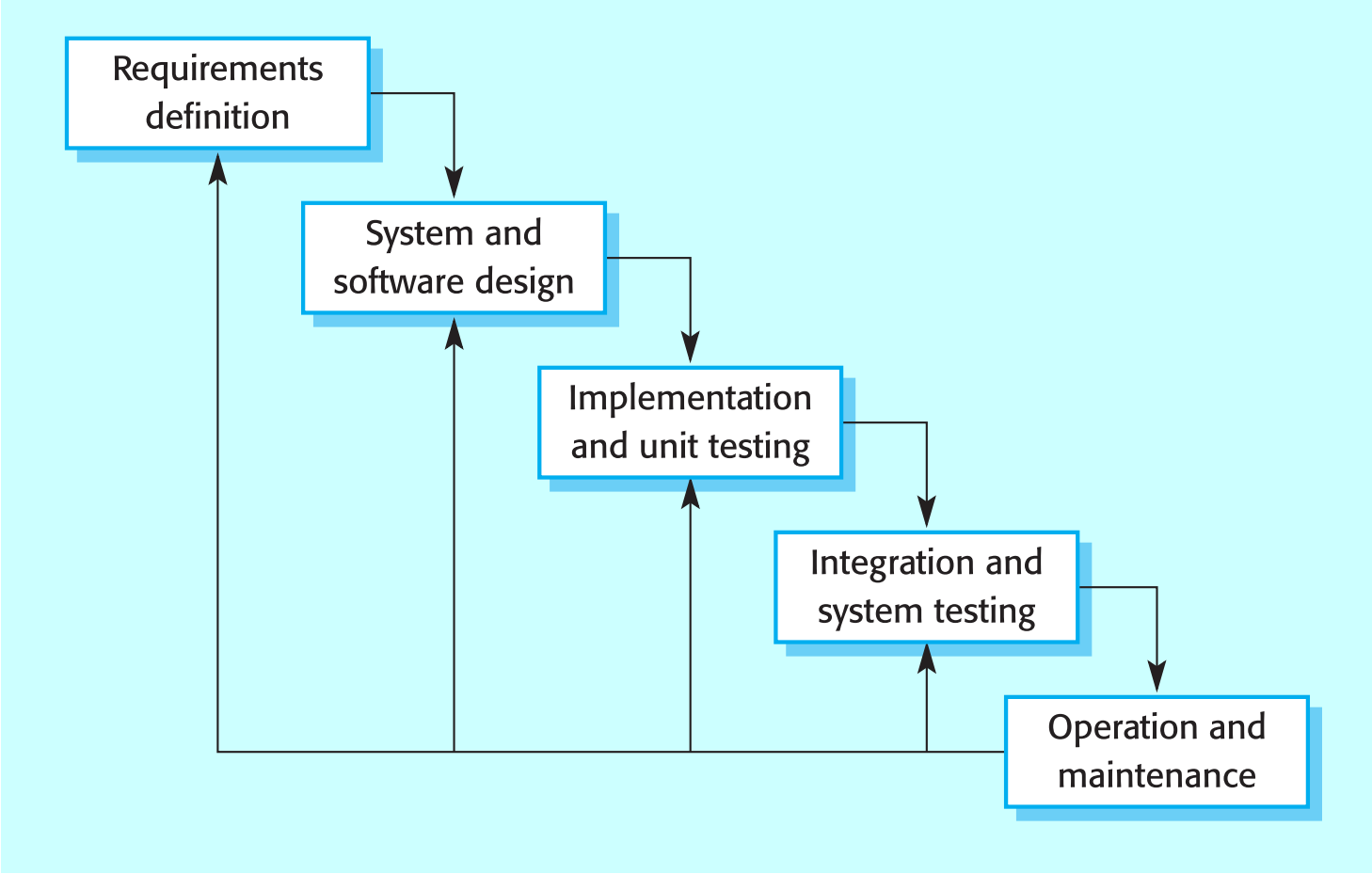
Yazılım Süreci ve Süreç Modeli

- 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?

Geleneksel Yazılım Süreç Modelleri

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

Çağlayan ("Waterfall") Modeli



Çağlayan Modeli – Aşamalar

- Gereksinim Tanımlama: Gerçekleştirilecek sistemin gereksinimlerinin belirlenmesi 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 Modeli – Zorluklar

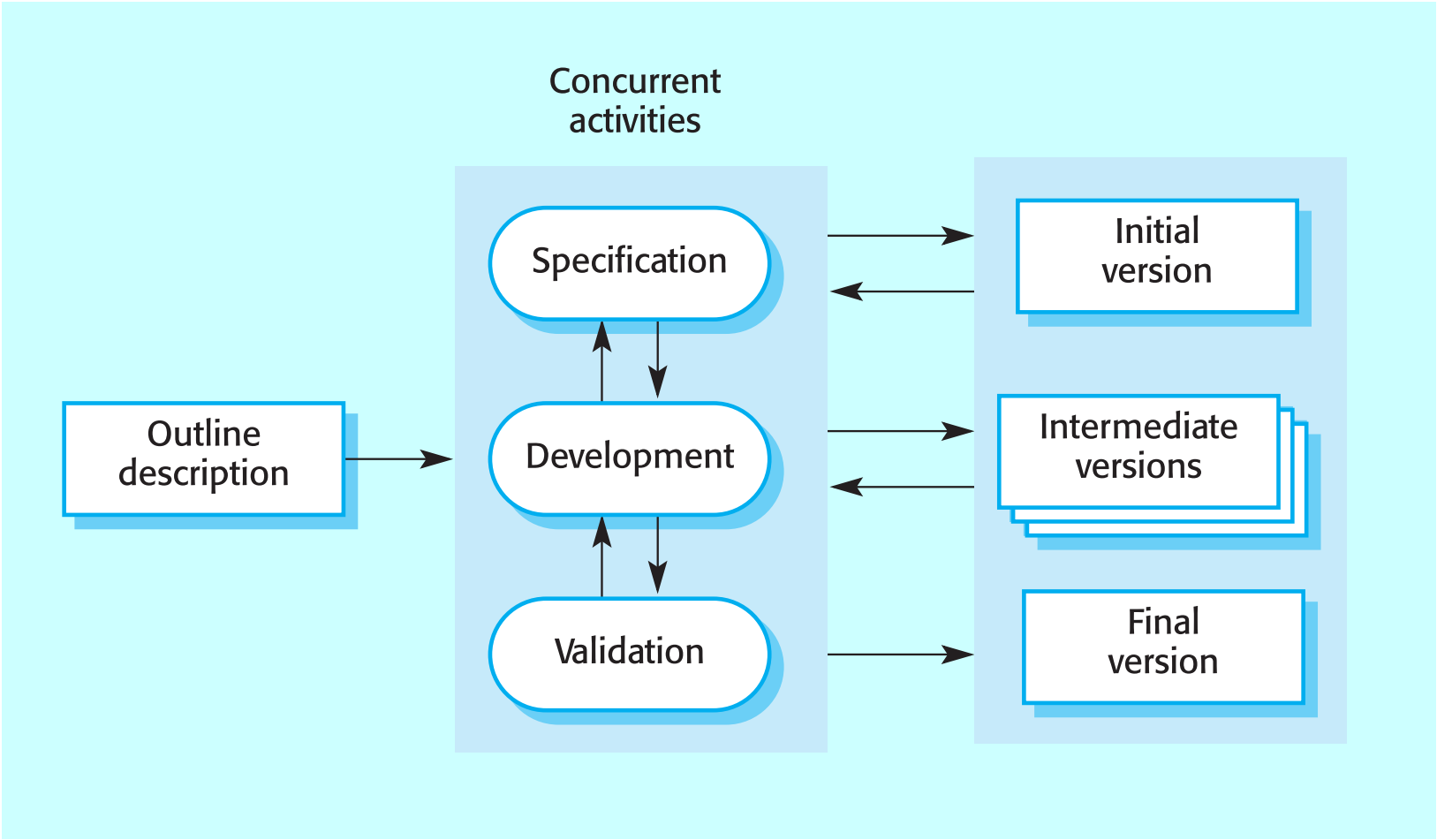
- Bir sonraki aşamaya geçmeden, önceki aşama neredeyse tümüyle tamamlanmış olmalıdır (örneğin, gereksinim tanımlama aşaması bitmeden tasarım aşamasına geçilemez.)
  - Bu şekilde geliştirme boyunca değişen müşteri isteklerinin sisteme yansıtılması zorlaşır.
  - Önceki nedenle bu model, gereksinimleri 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.

Evrimsel ("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.

Evrimsel Model – Adımlar



Evrimsel 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.

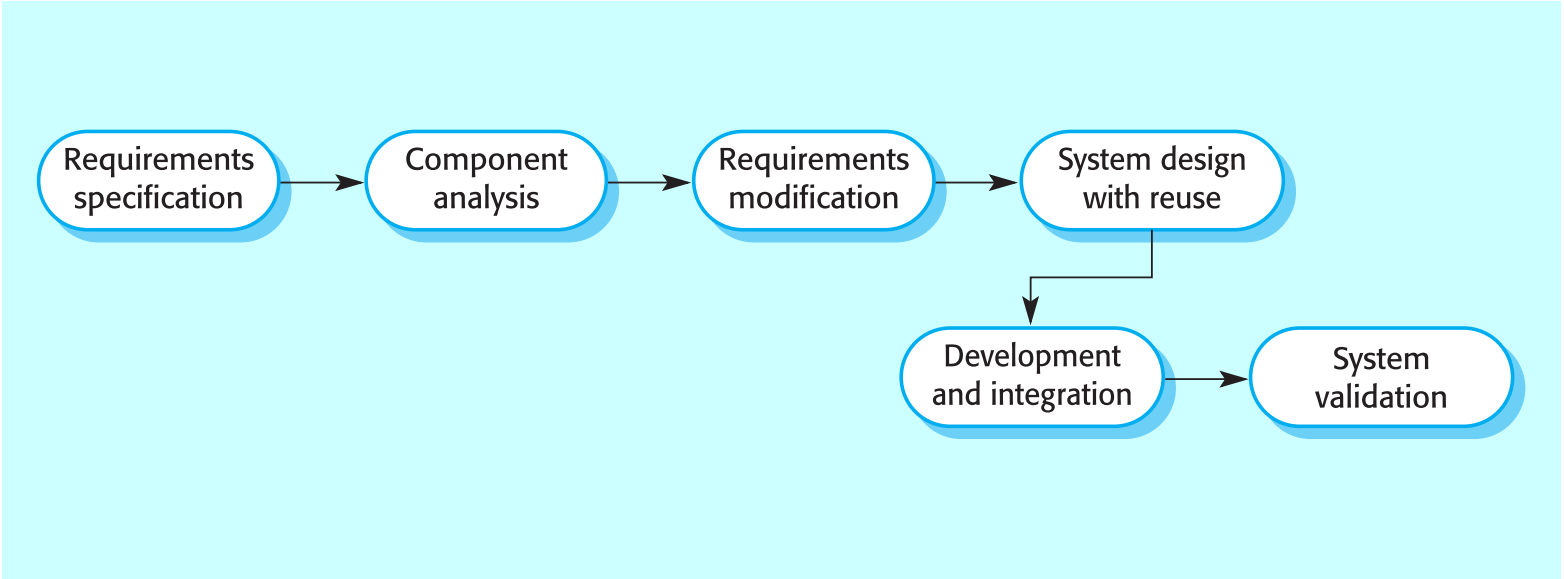
Bileşen-Tabanlı ("Component-Based") Model

- 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.

Bileşen-Tabanlı ("Component-Based") Model – Adımlar



Yazılım Süreç Modellerinde Süreç Tekrarı ("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.

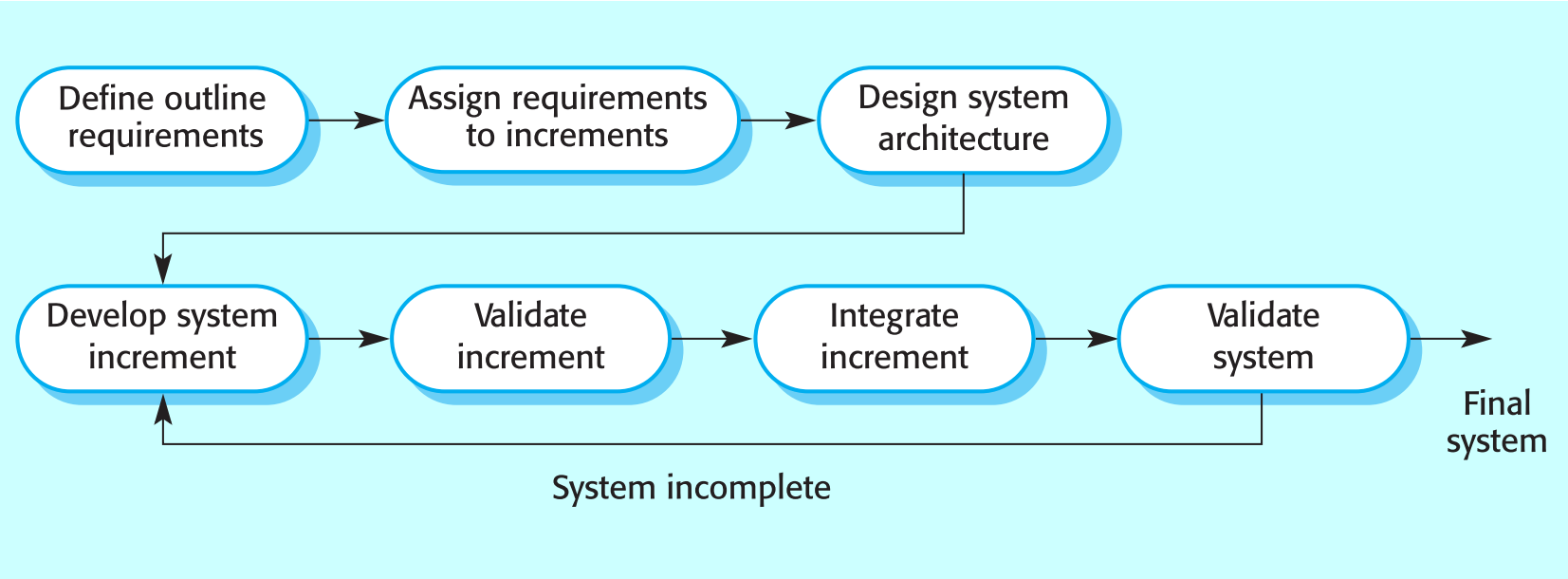
- Hangi süreç modelinin, sistemin hangi bölümleri için ve kaç tekrarda uygulanacağına proje başında karar verilir.

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

Artırımlı ("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.

Artırımlı Model – Adımlar



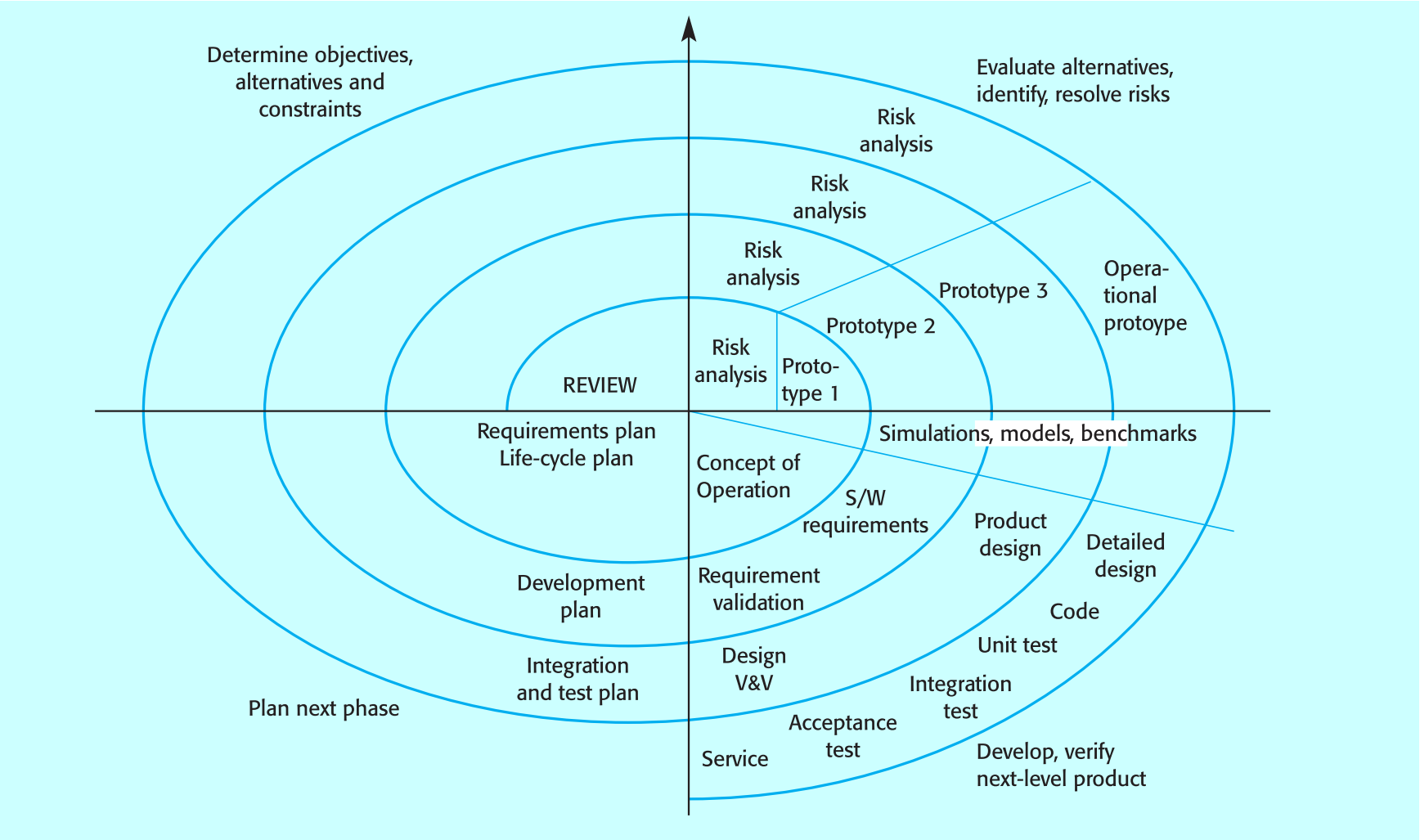
Artırımlı Model – Kazançlar

- 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.

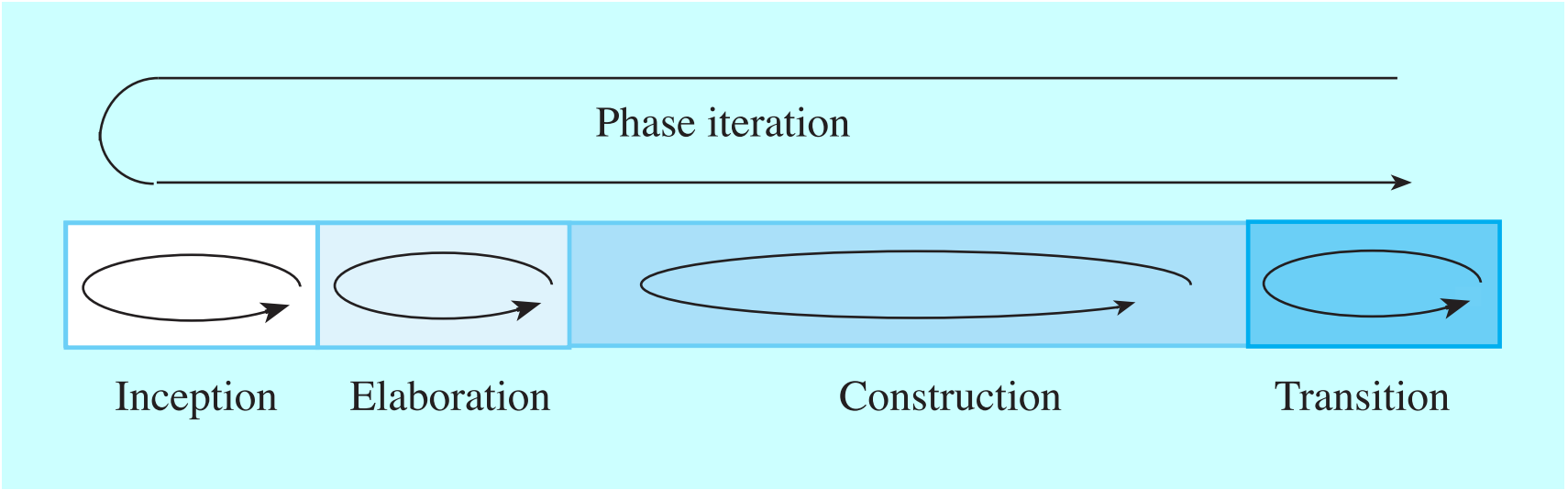
Döngüsel Geliştirme – Sektörler



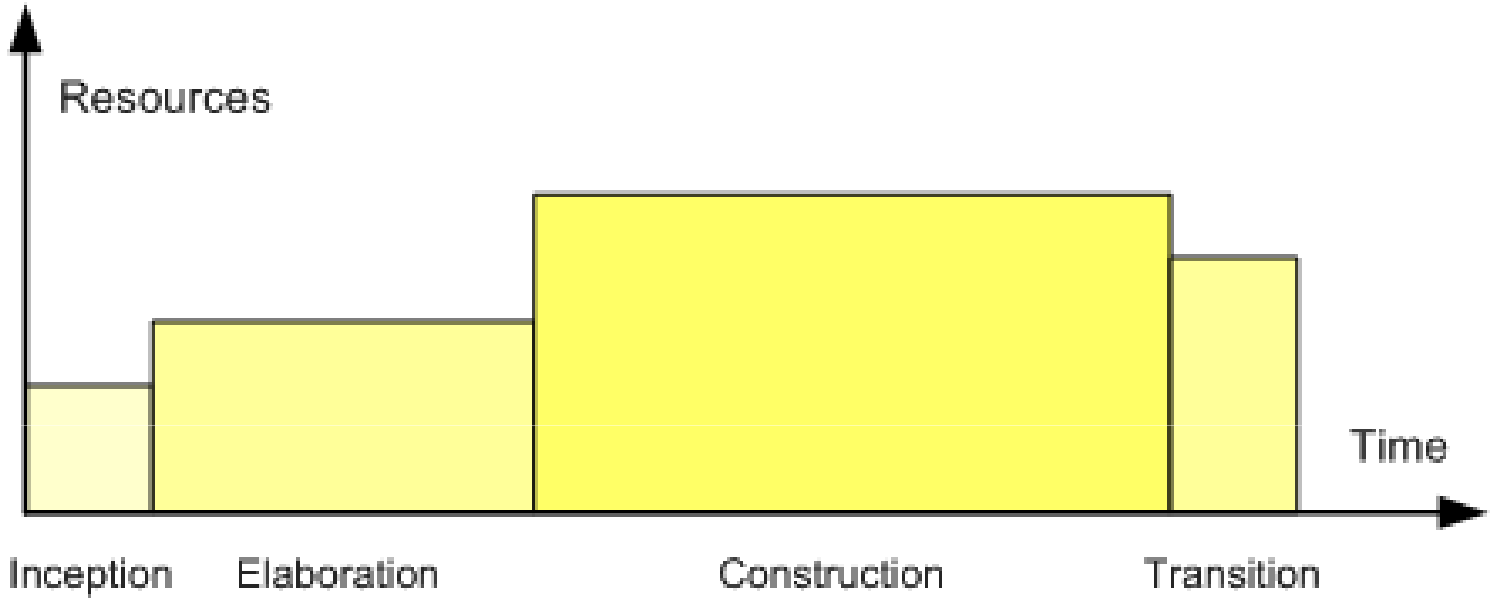
Örnek Bir Yazılım Süreç Modeli: Tümleşik Süreç ("Unified Process")

- 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.

Tümleşik Süreç – Aşamalar



Tümleşik Süreç – Yaşam Döngüsü



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

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

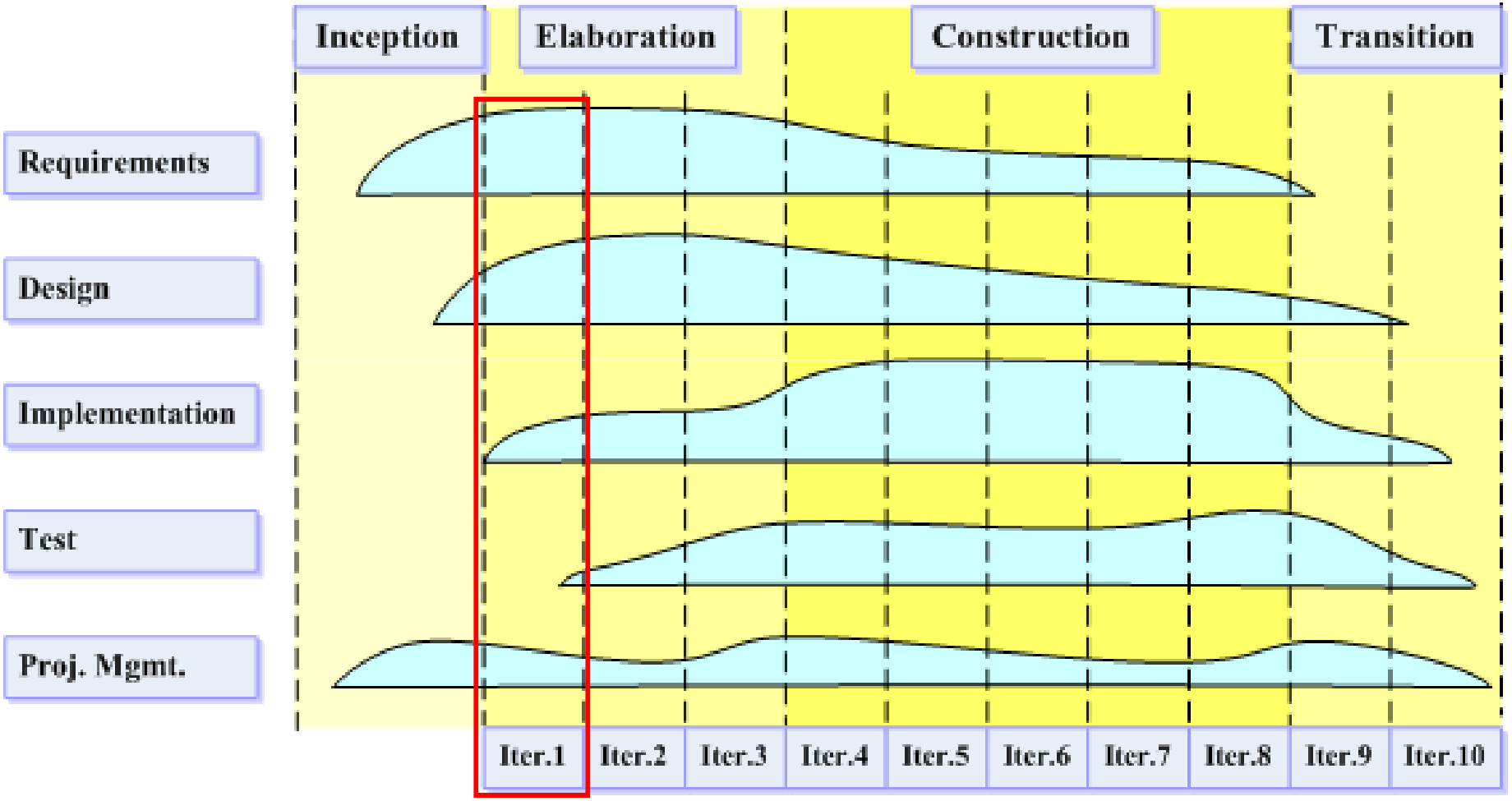
“Construction” – Ürünü gerçekleştir

“Transition” – Ürünü kullanıcılarına teslim et

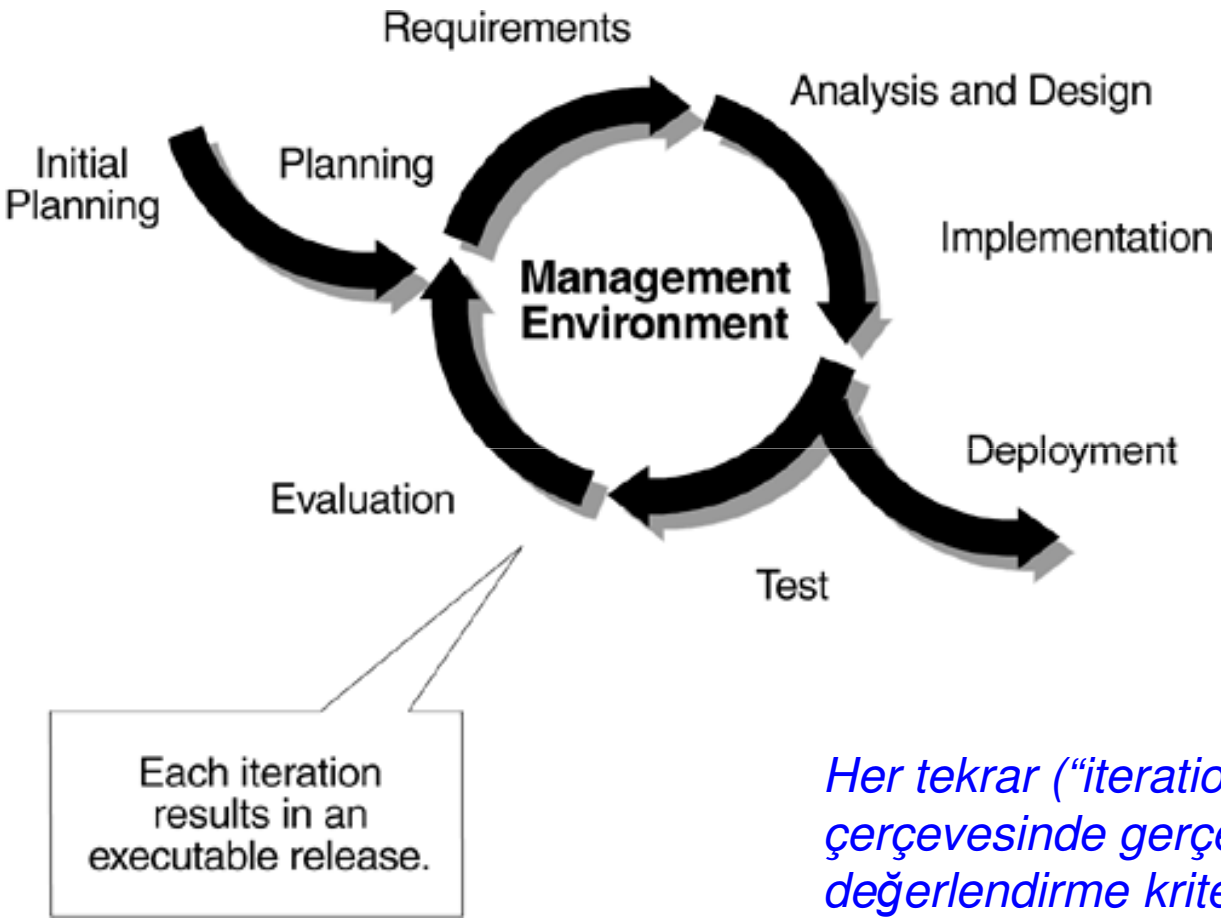
Tümleşik Süreç – İlkeler

- Yazılımı tekrarlı (“iteratively”) geliştir
- Gereksinimleri yönet
- Bileşen tabanlı (“component-based”) mimari kullan
- Yazılımı görsel modelle
- Yazılım kalitesini sürekli doğrula (“verification”)
- Yazılımla ilgili değişiklikleri kontrol et

Tümleşik Süreç – Yapı

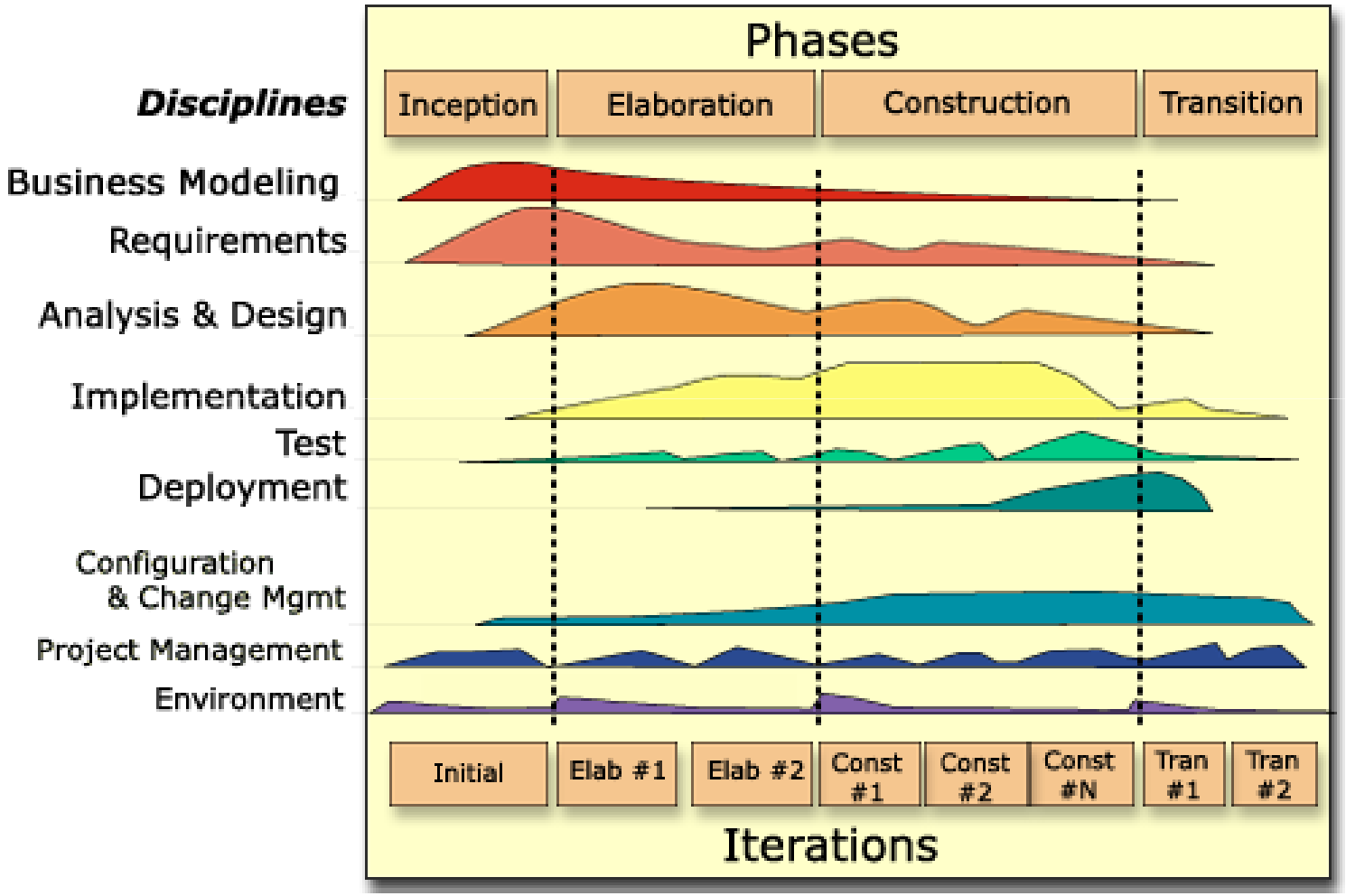


Tekrarlı Geliştirme ("Iterative Development")



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

“The Rational Unified Process” (Rational Software’in Tümleşik Süreci)



Kaynaklar ...

- Kaynaklar:
  - "Software Engineering" (8th. Ed.), Ian Sommerville, 2007.
  - *"Guide to the Software Engineering Body of Knowledge", 2004.*
  - "The Unified Software Development Process*",* ISBN 0-201-57169-2, 1999. Ivar Jacobson, Grady Booch, James Rumbaugh.
  - "*CMMI for Development*", v1.2, 2006.
  - "*A Guide to the Project Management Body of Knowledge*", 1996.

- Bilgi paylaşım alanı:
  - **http://ftp.cs.hacettepe.edu.tr/pub/dersler/BBS6XX/BBS651_YM/**
  - Dersi alan her öğrenci, her hafta bu alanı kontrol ve takip etmekle yükümlüdür!