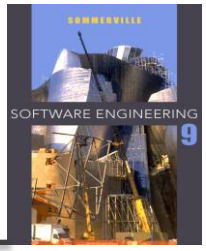
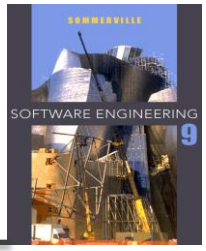


Bölüm 2– Yazılım Süreçleri

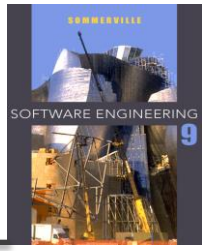
Ders 1

Konular



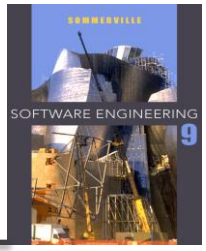
- ✧ Yazılım Süreç Modelleri
- ✧ Süreç Aktiviteleri
- ✧ Değişikliklerle Baş Etmek
- ✧ The Rational Unified Process (RUP)

Yazılım Süreci



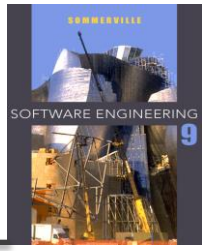
- Bir yazılım sistemini geliştirmek için gerekli olan aktiviteler kümesi
- Birçok farklı yazılım süreç modeli var fakat hepsi şunları içermeli:
  - Tanımlama – sitemin ne yapması gerektiğinin belirlenmesi;
  - Tasarım ve Gerçekleştirim – sistemin organizasyonunun belirlenmesi gerçekleştirilmesi;
  - Doğrulama – müşterinin istediği şeyleri yapıp yapmadığının kontrolü;
  - Evrim – değişen müşteri isteklerine göre sistemin değiştirilmesi.
- Bir yazılım süreç modeli gerçek bir sürecin özet gösterimidir

Yazılım Süreç Tanımlamaları



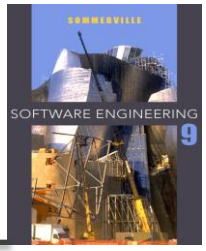
- Yazılım süreçleri ile ilgili konuştuğumuzda aslında bu süreçlerdeki veri modeli belirleme, kullanıcı arayüzü tasarımı gibi aktiviteler ve bu aktivitelerin sırası hakkında konuşuruz.
- Yazılım süreç tanımlamaları aşağıdakiler içerebilir:
  - Süreç aktivitesinin sonunda elde edilecek **ürünler**;
  - Süreçte yer alan kişilerin sorumluluklarını gösteren **roller**.
  - Süreç öncesinde belirlenen ve süreç sonrasında yerine getirilip getirilmediği kontrol edilen **koşullar**.

Plan tabanlı ve çevik (agile) süreçler



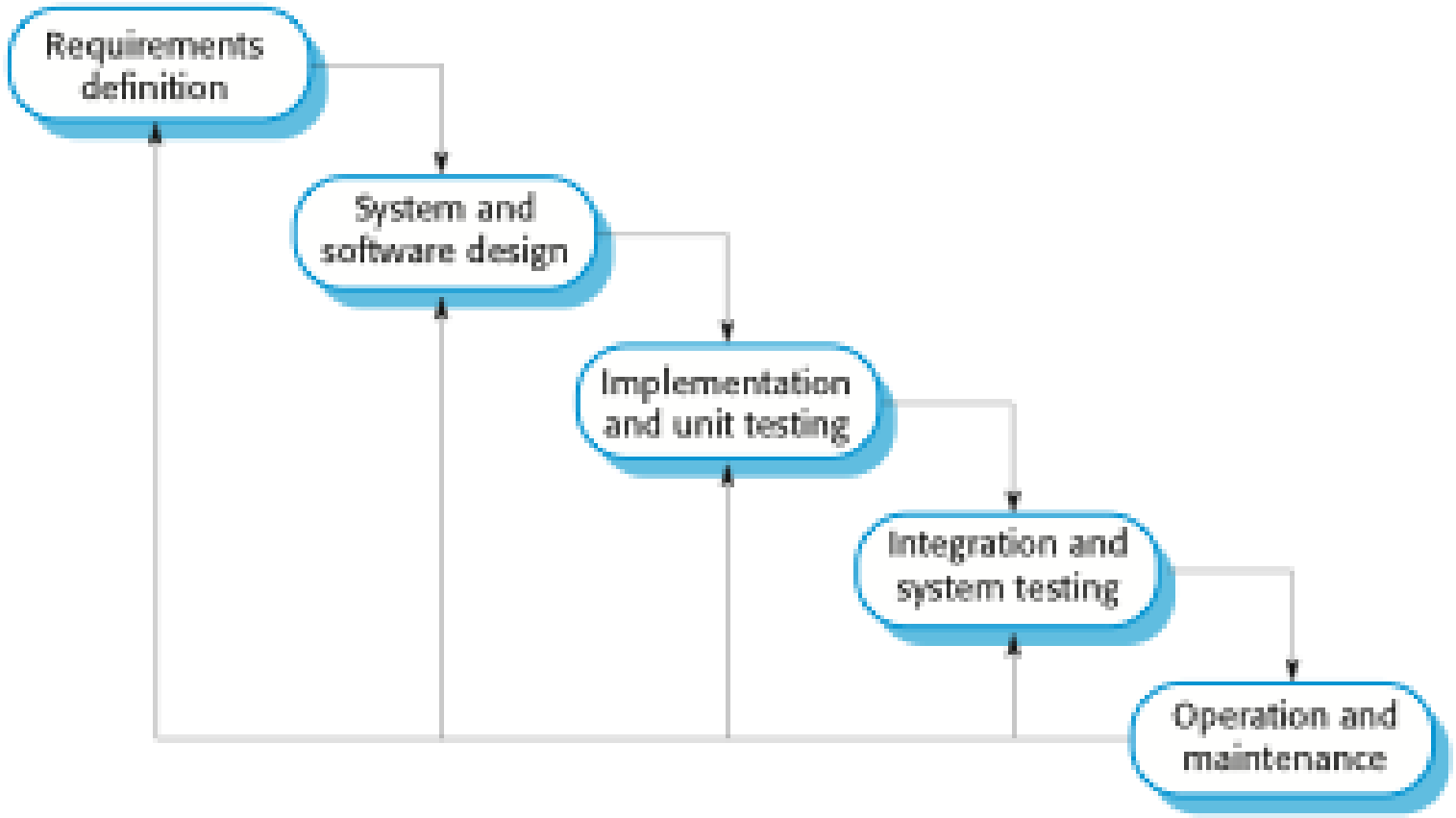
- Plan tabanlı süreçler, süreç aktivitelerinin önceden planlandığı ve ilerlemenin bu plana göre ölçüldüğü süreçlerdir.
- Çevik süreçlerde planlama artırımlı şekilde yapılır ve bu nedenle sürecin, değişen müşteri ihtiyaçlarını karşılayacak şekilde değiştirilmesi kolaydır.
- Pratikte, uygulanabilirliği en yüksek olan süreç modelleri plan tabanlı ve çevik yaklaşımlardan unsurlar barındıran modellerdir.
- Doğru ya da yanlış yazılım süreç modeli yoktur.

Yazılım süreç modelleri

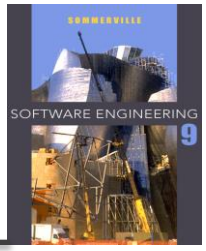


- ✧ Çağlayan (waterfall) modeli
  - Plan tabanlı bir modeldir. Tanımlama ve geliştirme aşamaları ayrıdır.
- ✧ Artırımlı (Incremental) geliştirme
  - Tanımlama, geliştirme ve doğrulama aşamaları ard arda gelir. Plan tabanlı veya çevik olabilir.
- ✧ Tekrar kullanım tabanlı yazılım mühendisliği
  - Sistem, var olan bileşenlerden inşa edilir. Plan tabanlı veya çevik olabilir.
- ✧ Pratikte birçok büyük sistem bu modellerin unsurlarının bir arada kullanılması ile geliştirilir.

Çağlayan (waterfall) modeli



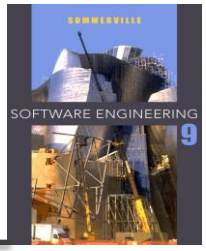
Çağlayan (waterfall) modelinin aşamaları



- Aşamalar:
  - İhtiyaç analizi
  - Sistem ve yazılım tasarımı
  - Geliştirme ve birim testi
  - Entegrasyon ve sistem testi
  - Uygulama ve bakım

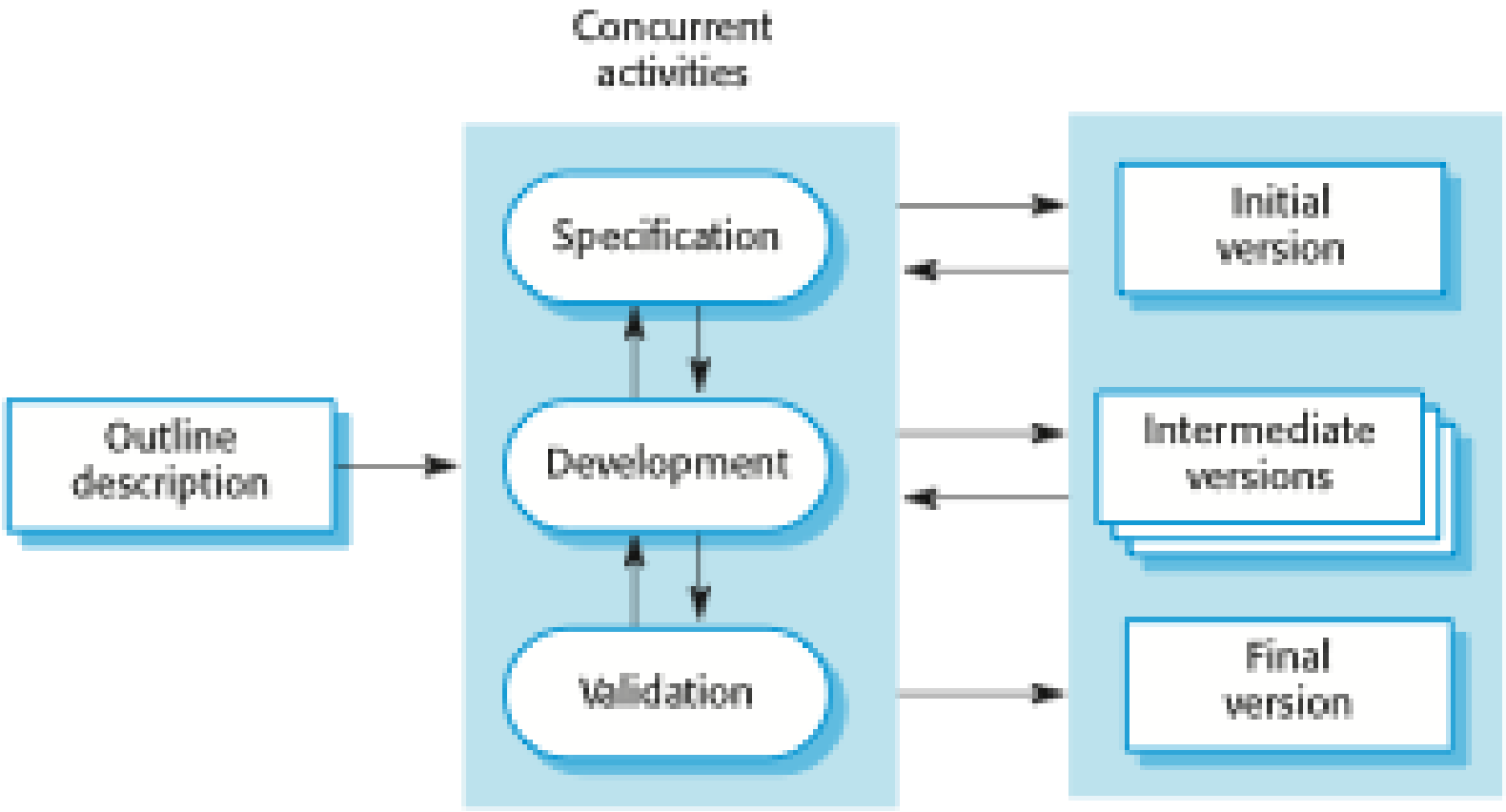
- Bu modelin en önemli eksikliği, süreç devam ediyorken bir değişikliği adapte etmekteki zorluktur.
- İlkesel olarak, bir sonraki aşamaya geçmeden önce bir önceki aşamanın tamamlanmış olması gerekir.

Çağlayan (waterfall) modelinin problemleri

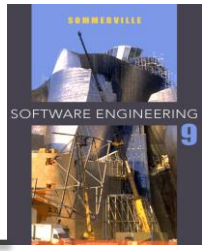


- Projeyi esnek olmayan alt aşamalara ayırmak, müşteri ihtiyacındaki değişikliğe cevap vermeyi zorlaştırır
  - Bundan dolayı bu model yalnızca çok iyi tanımlanmış ve anlaşılmış ihtiyaçlar üzerinde ve hemen hemen hiç değişiklik olmayan durumlarda kullanışlıdır.
  - Ancak, çok az iş alanı stabil ihtiyaçlara sahiptir.

Artırımlı (Incremental) geliştirme

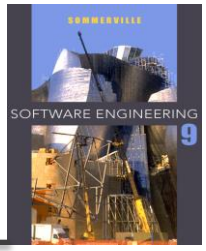


Artırımlı (Incremental) geliştirmenin faydaları



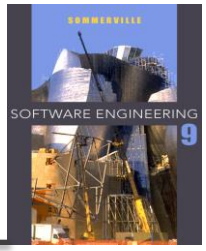
- Müşteri ihtiyaçlarındaki değişikliklerin yerine getirilme maliyetini düşürür
  - Yeniden yapılması gereken analiz ve dokümantasyon işleri, çağlayan modelinde olması gerekenden daha azdır.
- Geliştirme süresi ile ilgili müşteri geri bildirimlerini almak daha kolaydır.
  - Müşteri, yazılım gösterimleri (demonstrations) üzerine yorum yapabilir ve işin ne kadarının yapıldığını görebilir.
- Yazılım teslim süresi çağlayan modeline göre daha kısadır

Artırımlı (Incremental) geliştirmenin problemleri



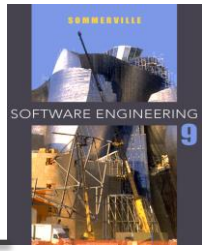
- ✧ Süreç görülebilir değildir.
  - Yöneticiler, süreçle ilgili düzenli olarak bilgi almak ister. Eğer sistem çabuk bir biçimde geliştiriliyorsa, sistemin her bir versiyonu için yöneticilere sunulacak raporlar hazırlamak maliyet-etkin olmaz.
- ✧ Her bir değişiklik sistemin yapısında bozulmalara neden olur.
  - Yazılımın iç yapısının yeniden düzenlenmesi yazılımı geliştirse de; devamlı değişiklik yapmak bir süre sonra yeni değişikliklerin daha zor ve maliyetli yapılmasına neden olur.

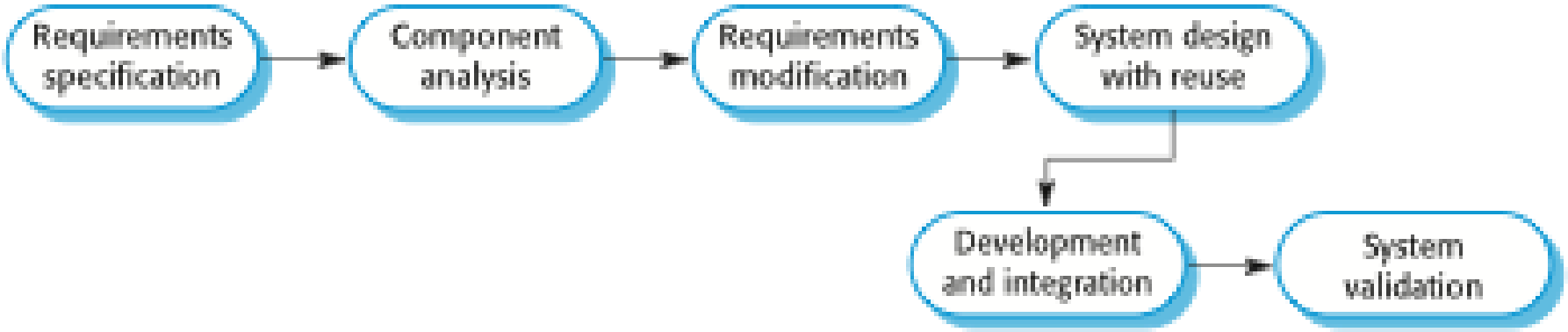
Tekrar kullanım tabanlı yazılım mühendisliği



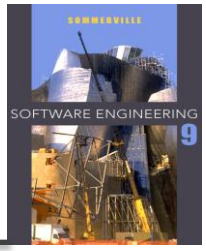
- Var olan veya ticari olarak temin edilebilen (COTS - Commercial-off-the-shelf) bileşenlerin bir araya getirilmesi ile üretilen sistemlerdir.
- Süreç aşamaları
  - Bileşen analizi
  - Gerekli değişikliklerin yapılması
  - Bileşenleri kullanarak sistemin tasarlanması
  - Geliştirme ve bütünleştirme

Tekrar kullanım tabanlı yazılım mühendisliği

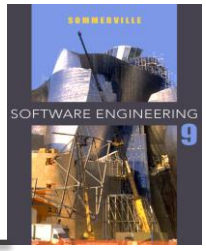




Tekrar kullanım tabanlı yazılım mühendisliği
Yazılım bileşen tipleri

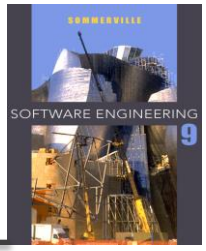


- Web servisleri
- Paket haline getirilmiş yazılım nesneleri
- Belirli bir amaç için konfigüre edilebilen ve yalnız başına çalışabilen yazılım sistemleri (COTS)



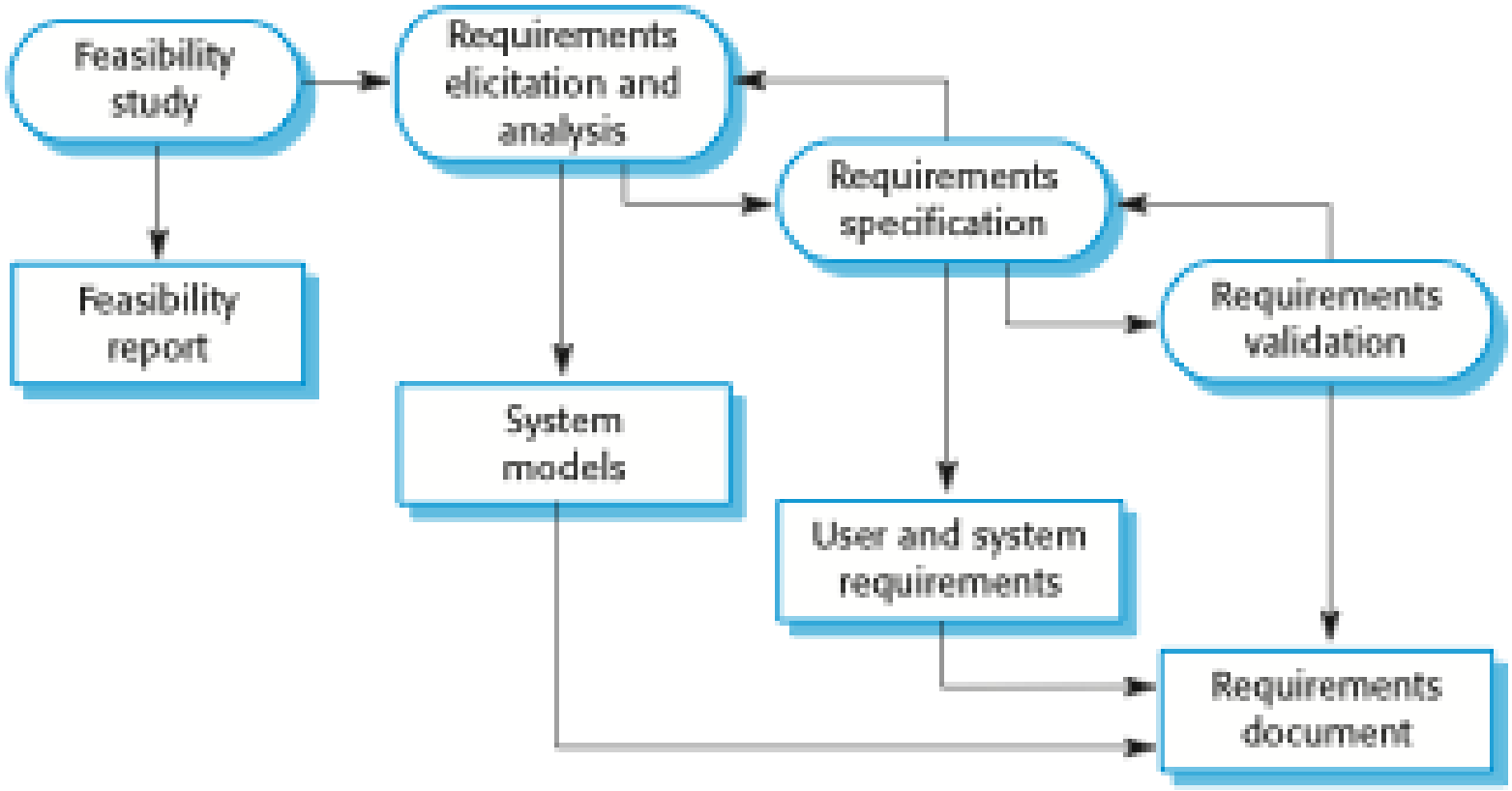
Süreç Aktiviteleri

Yazılım tanımlama (İhtiyaç Mühendisliği)

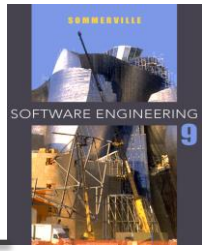


- Hangi hizmetlerin gerekli olduğunun tespit edildiği ve sistemin geliştirilmesi ve uygulaması üzerindeki kısıtların ortaya konulması
- İhtiyaç mühendisliği süreci aktiviteleri (müşteri ile anlaşma)
  1. Uygulanabilirlik çalışması (Feasibility study)
     - Teknik ve finansal olarak bu sistemi geliştirmek mümkün mü?
  2. İhtiyaçları belirleme ve analiz etme (Requirements elicitation and analysis)
     - Sistemin paydaşlarının sistemden beklentileri nelerdir?
  3. İhtiyaçları tanımlama
     - İhtiyaçların detaylıca tanımlanması
  4. İhtiyaçların doğrulanması
     - İhtiyaçların doğruluklarının kontrol edilmesi

İhtiyaç mühendisliği süreci

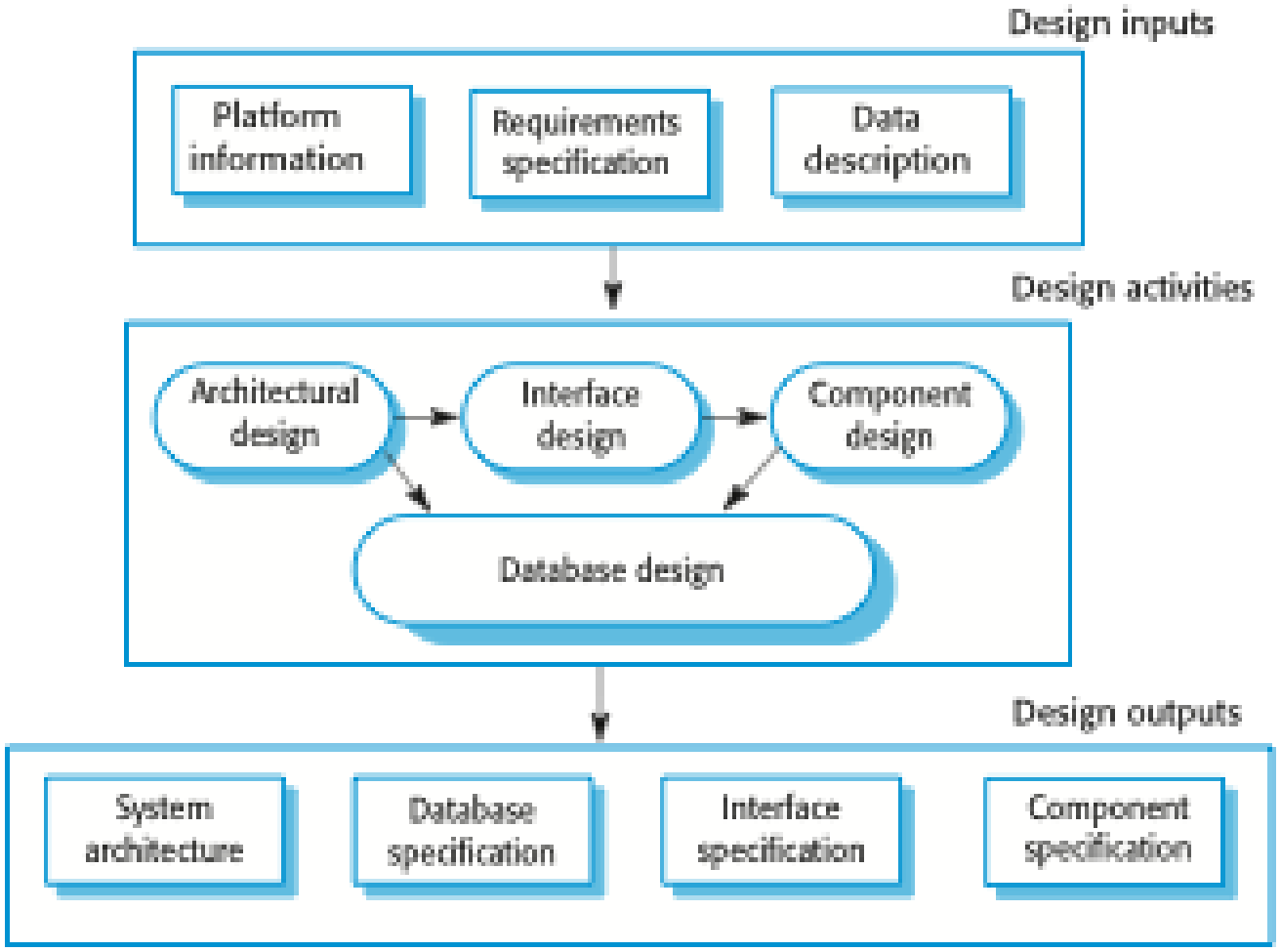


Yazılım tasarım ve gerçekleştirme

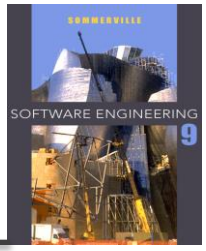


- Sistem ihtiyaçlarının çalıştırılabilir (executable) bir sistem haline getirilmesi
- Yazılım tasarımı
  - İhtiyaçları cevap verebilecek bir yazılım yapısı tasarlama
- Gerçekleştirim
  - Tasarlanan yapının çalıştırılabilir bir sistem haline getirilmesi ;
- Tasarlama ve gerçekleştirme faaliyetleri yakından ilgilidir ve iç içe geçmiş şekilde yapılabilir.

Tasarım sürecinin genel bir modeli

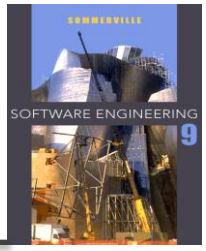


Tasarım aktiviteleri



- *Mimari tasarım,* bütün bir sistemin yapısının, ana bileşenlerinin (alt sistem veya modüllerin) ve bunlar arasındaki ilişkilerin yapısı
- *Arayüz tasarımı,* sistem bileşenleri arasındaki arayüzlerin tasarlanması
- *Bileşen tasarımı,* her bir sistem bileşeninin nasıl çalışacağının tasarlanması.
- *Veritabanı tasarımı,* sistemin kullanacağı veri yapılarının tasarlanması ve bunların bir veritabanında nasıl tutulacağının belirlenmesi.

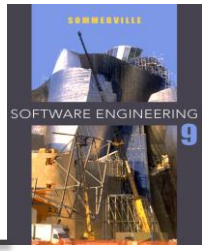
Yazılım doğrulama



- Tetkik ve Tasdik (Verification and validation - V & V), bir sistemin daha önceden belirlenmiş olan ihtiyaçları karşılayıp karşılamadığını belirlemek üzere gerçekleştirilir.
- Kontrol, gözden geçirme ve sistem testini kapsar
- Sistem testi, sistem tarafından kullanılacak olan gerçek veriler ile gerçekleştirilir.
- Test, en yaygın kullanılan doğrulama aktivitesidir.

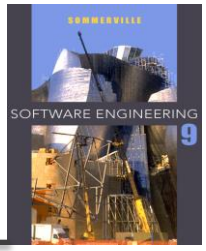
Test aşamaları

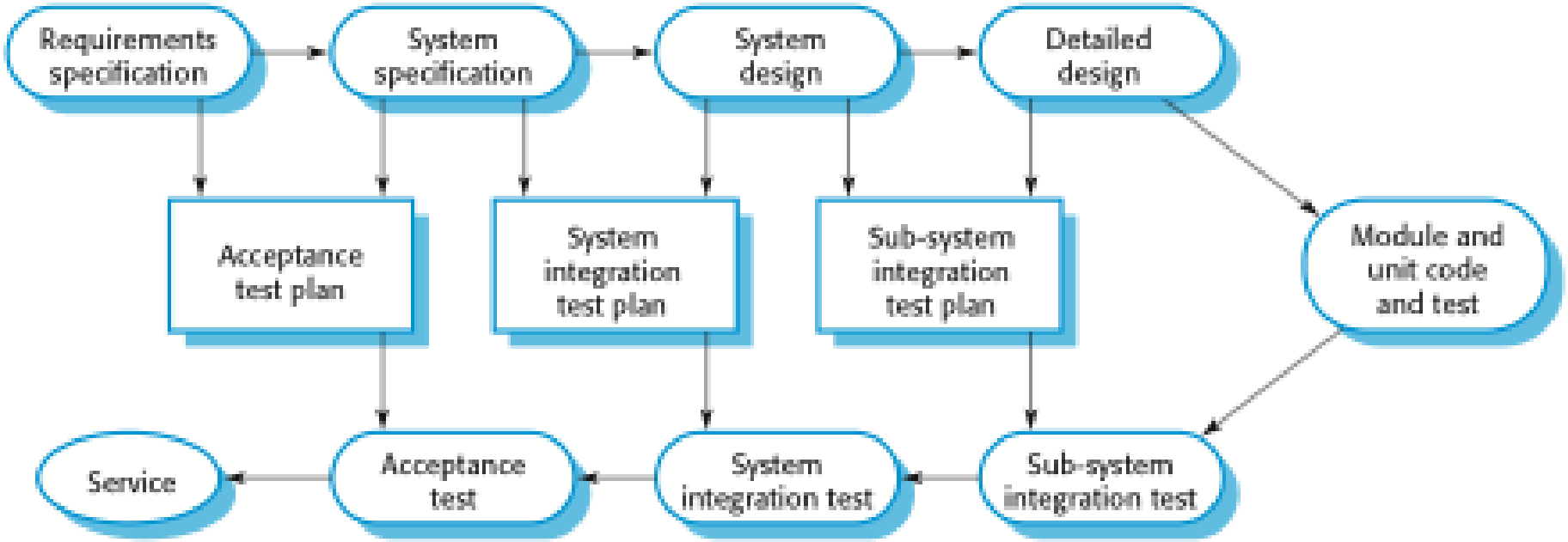
Test aşamaları



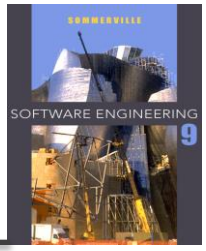
- ✧ Geliştirme veya bileşen testi
  - Birbirinden ayrı bileşenler ayrı ayrı test edilir;
  - Bir bileşen, fonksiyon, nesne vb. olabilir.
- ✧ Sistem testi
  - Sistemin bütün olarak test edilmesi.
- ✧ Kabul testi
  - Müşteri ihtiyaçlarının karşılandığını göstermek için müşterinin verileri ile sistemin test edilmesi

Plan tabanlı bir süreçteki test aşamaları



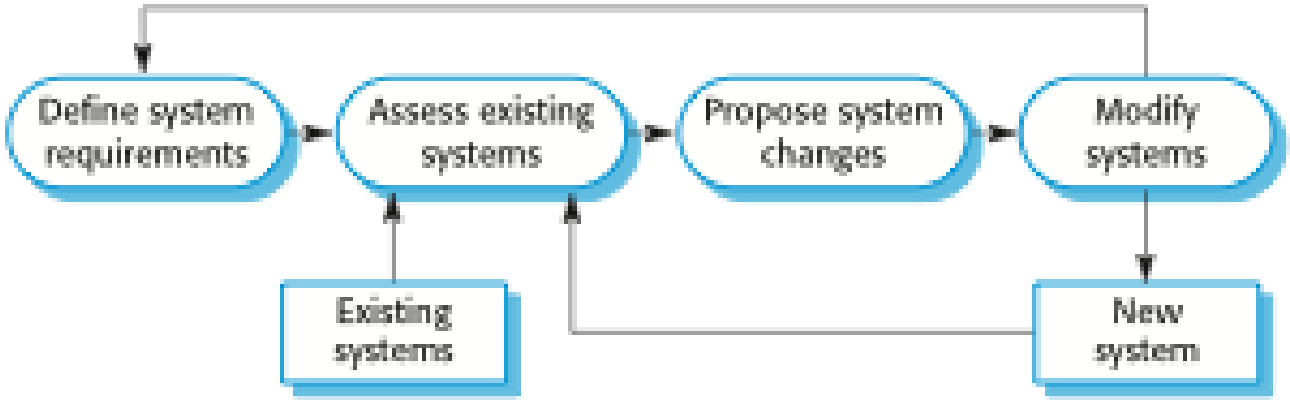


Yazılım evrimi

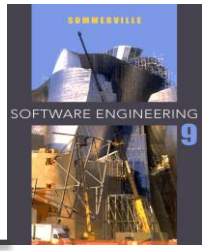


- Yazılım, doğası gereği esnektir ve değişebilir.
- Değişen iş hayatı koşulları ile değişen ihtiyaçların yazılım tarafından desteklenmesi gerekir.
- Yazılım geliştirme ile evrim (bakım, güncelleme) arasında bir sınır olmasına rağmen, yeni sistemlerde bu sınır gittikçe belirsizleşmekte.

Yazılım evrimi

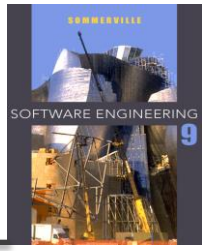


Özet



- Yazılım süreçleri, yazılım üretimindeki aktivitelerdir. Yazılım süreç modeller ise bu süreçlerin özet bir gösterimidir.
- Çağlayan modeli, artırımlı model gibi yazılım süreç modelleri, yazılım süreçlerinin organizasyonunu tanımlayan modellerdir.

Özet

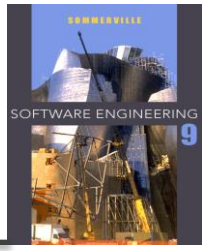


- İhtiyaç mühendisliği, yazılım tanımlama sürecidir.
- Tasarım ve gerçekleştirim, belirlenen ihtiyaçların çalışır bir yazılım sistemi haline getirilmesi ile ilgilenir.
- Yazılım doğrulama, yazılımın daha önceden tanımlanan işleri yapabildiğinin ve kullanıcı isteklerini karşıladığının doğrulanmasıdır.
- Yazılım evrimi, yeni ihtiyaçları karşılamak üzere yazılım üzerinde yapılan değişikliklerdir.

Bölüm 2– Yazılım Süreçleri (Kısım 2)

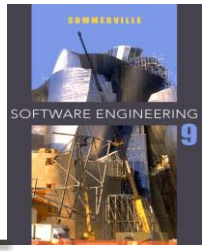
Ders 1

Değişikliklerle baş etmek



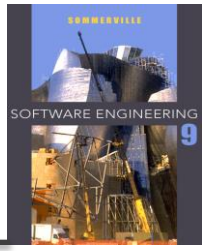
- Bütün büyük yazılım projelerinde değişiklik kaçınılmazdır.
  - Değişen iş gereksinimi
  - Yeni yazılım geliştirme teknolojileri
  - Değişen platformlar
- Değişiklik, yeniden çalışmak demektir.
  - Yeniden çalışmak yalnızca kodları değiştirmek değil. yeniden analiz, yeniden tasarım…

Yeniden çalışma maliyetini azaltmak



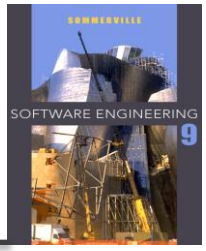
- ✧ Prototipleme
  - Ana hatları tamamlanmış bir sürümü müşteriye göster; istediği değişiklikleri yap.
- ✧ Artırımlı geliştirme
  - Değişiklikleri, artırım aşamalarında yap.
  - Mümkün değilse, yalnızca bir artırıma indirgemeye çalış.

Yazılım prototipleme



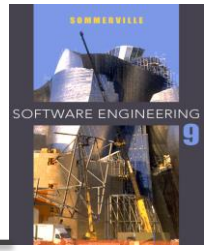
- Tasarım opsiyonlarını ve ana hatları gösteren taslak sürüm
- Prototipleme başka nelere yarar?
  - İhtiyaç mühendisliği sırasında ihtiyaçların elenmesi ve doğrulanması
  - Tasarım sürecinde arayüz alternatiflerinin değerlendirilmesi
  - Test aşamasına, testlerin arka arkaya yapılması

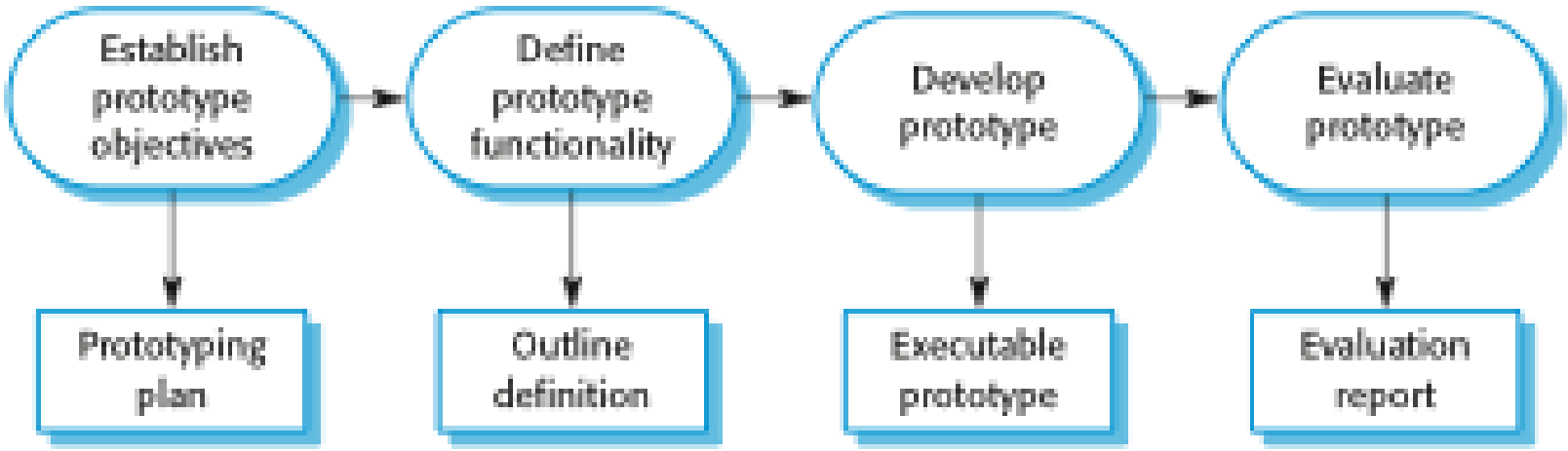
Prototiplemenin faydaları



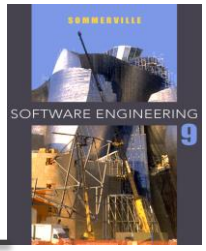
- ✧ Sistem kullanılabilirliğini arttırır.
- ✧ Sistemi, kullanıcının gerçek ihtiyaçlarına yaklaştırır
- ✧ Tasarım kalitesini arttırır
- ✧ Sürdürülebilirliği arttırır
- ✧ Geliştirme eforunu azaltır

Prototiplemenin aşamaları



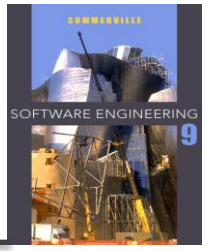


Prototip geliştirme



- ✧ Programlama dilleri veya görsel araçlarla yapılabilir.
- ✧ Şunları içerebilir
  - Prototipleme, ürünün iyi anlaşılmamış alanlarına odaklanmalı
  - Hata kontrolü gibi detayları prototipte bulunmayabilir
  - Fonksiyonel olmayan (güvenilirlik, güvenlik) ihtiyaçlardansa fonksiyonel ihtiyaçlara odaklan

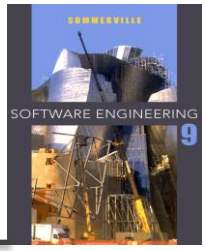
Prototipleri atın



✧ Son ürünün geliştirmesinde prototipi kullanmayın

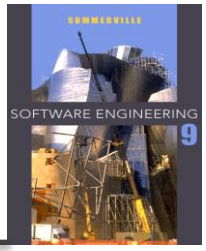
- Fonksiyonel olmayan sistem gereksinimlerini karşılamak imkansız olabilir.
- Prototipler genelde dokümante edilmezler
- Prototipin yapısı, hızlı değişikliklerden dolayı çabucak bozulur
- Prototipler, kalite standartlarını sağlamayabilir.

Artırımlı geliştirme



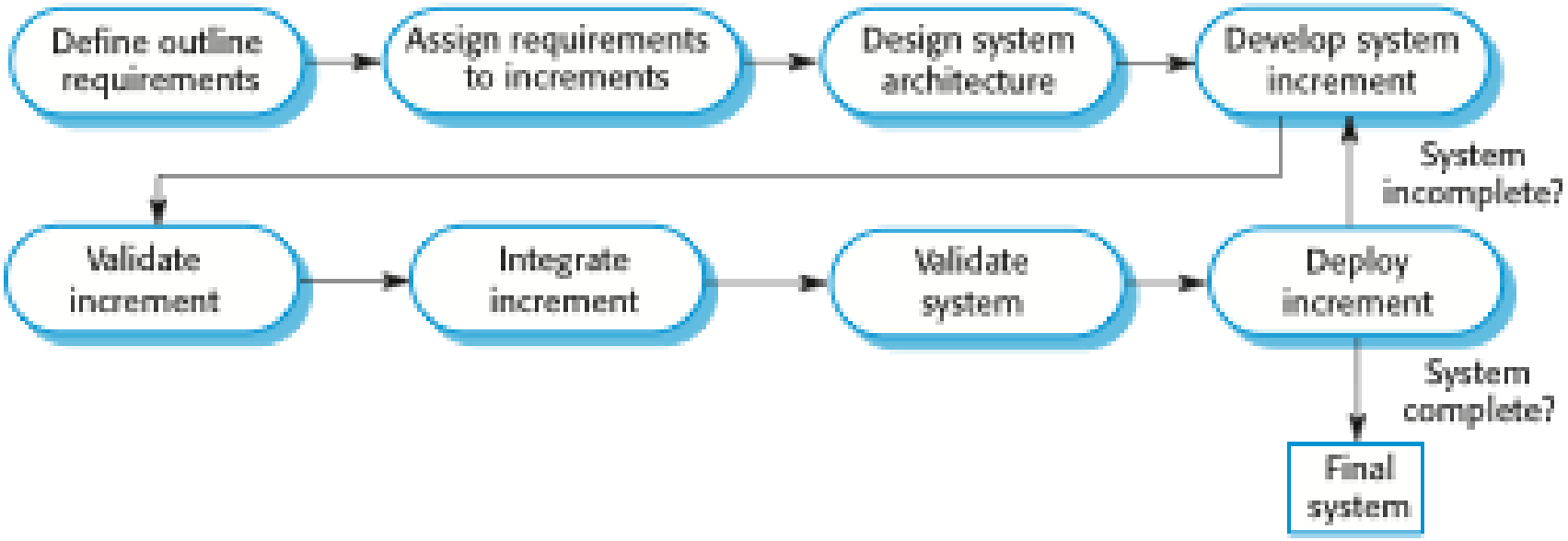
- Sistemi bütün olarak teslim etmek yerine, her seferinde bazı fonksiyonları tamamlayarak teslim etmek.
- Kullanıcı istekleri önceliklendirilir ve en öncelikli istekler ilk artırımlarda yapılır.

Artırımlı geliştirme ve teslim

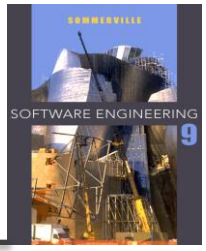


- ✧ Artırımlı geliştirme
  - Sistemi artırarak geliştir ve her artırımdan önce eldeki ürünü test et
- ✧ Artırımlı teslim
  - Her artırımı müşterinin kullanımına sun
  - Yazılımın kullanılması sayesinde daha gerçekçi değerlendirme sağlar

Artırımlı teslim

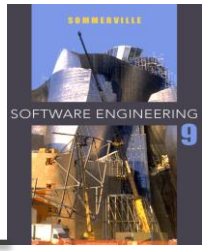


Artırımlı teslimin avantajları



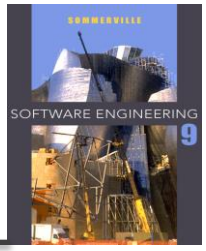
- Sistemin fonksiyonelliği erken aşamalarda başlar.
- İlk artırımlar prototip işlevi görürler ve sonraki artırımlar için öncelikli gereksinimlerin belirlenmesini sağlarlar
- Projenin tamamen başarısız olma riski düşer
- En çok testin, en öncelikli sistem hizmetleri için yapılması sağlanır

Artırımlı teslimin problemleri



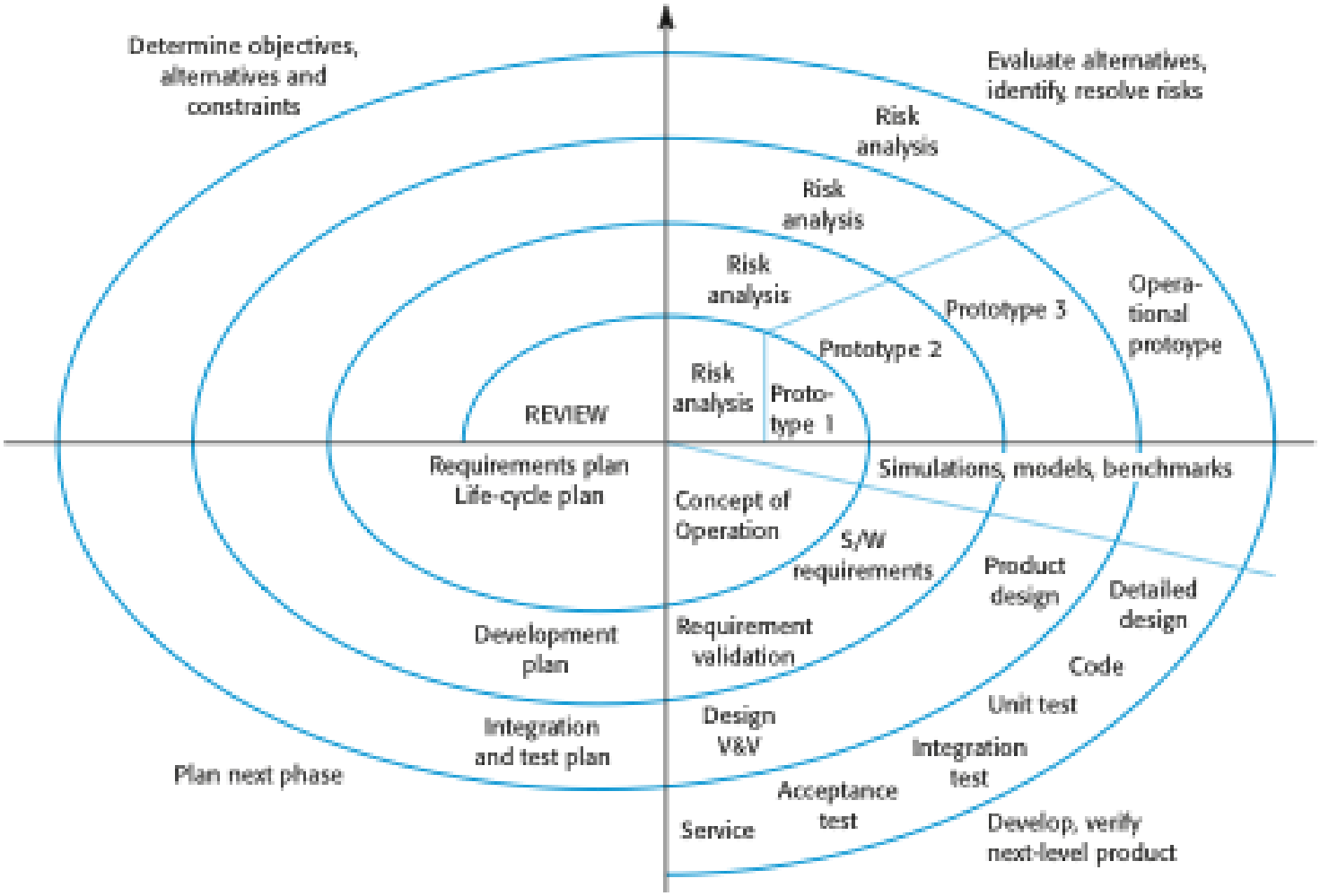
- Birçok sistem, sistemin değişik parçalarının ortak olarak kullanacağı fonksiyonelliği gerektirir
- Artırımlı geliştirmenin temeli, sistem tanımlamasının geliştirme aşamasında yapılmasıdır.
  - Sistem tanımlamasının tamamı, son artırımda bitirilmiş olur. Bu duruma müşterinin ikna edilmesi zor olabilir.

Boehm'in spiral modeli

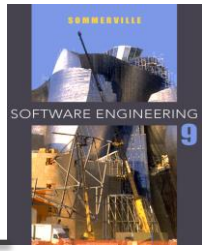


- Süreç, birbirini izleyen aşamalar yerine spiral olarak gösterilir
- Spiraldeki her döngü, süreçteki bir aşamayı gösterir
- Tanımlama, tasarlama gibi sabit aşamalar yoktur. Her bir döngünün içerisinde neye ihtiyaç duyuluyorsa o aşama seçilir
- Riskler, süreç boyunca açık bir şekilde belirlenir ve çözülür.

Boehm'in spiral modeli

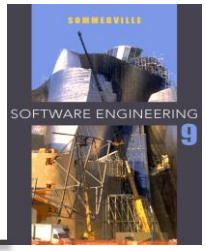


Boehm’in spiral modelinin kullanımı



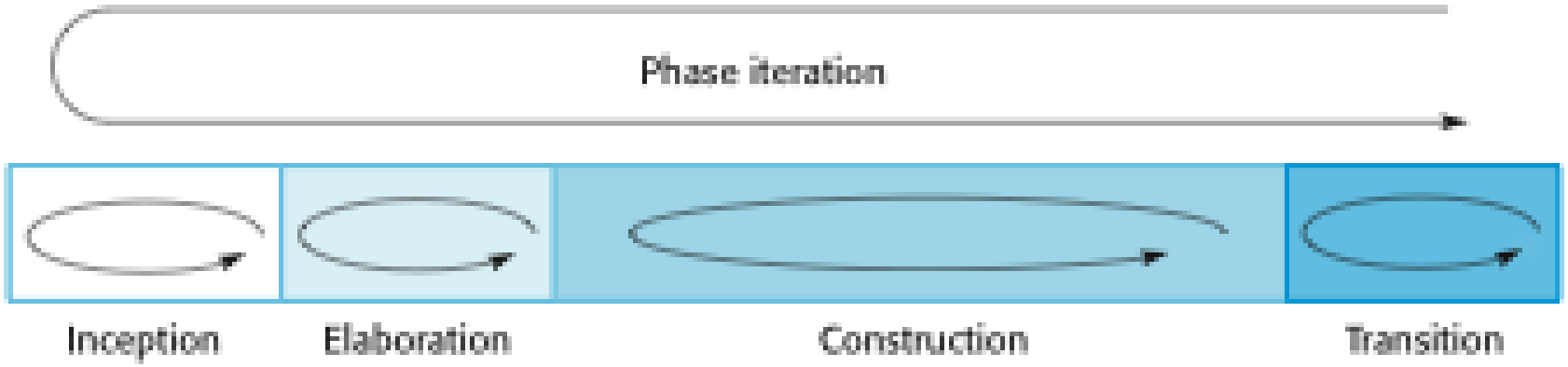
- ✧ Riskleri değerlendirme ve yok etmede başarılı.
- ✧ Ancak pratikte uygulaması hemen hemen yok.

The Rational Unified Process

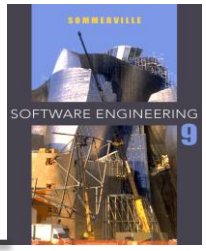


- UML ve ilgili süreçler üzerinde çalışan modern bir genel süreç modeli
- Daha önceden açıkladığımız 3 tane genel süreç modelinin farklı yönlerini bir araya getirir.
- 3 bakış açısı ile süreç tarif edilir
  - Aşamaları zamana göre gösteren bakış açısı
  - Süreç aktivitelerini gösteren statik bakış açısı
  - İyi uygulamaları öneren pratik bir bakış açısı

Rational Unified Process'deki aşamalar

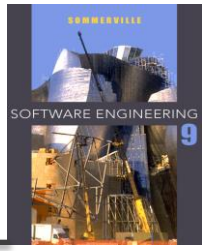


RUP aşamaları



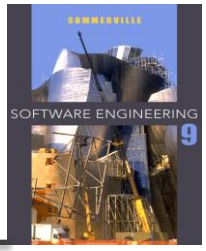
- ✧ Başlangıç
  - Sistemin olurluğunu kanıtla
- ✧ Ayrıntılandırma
  - Problemin alanını belirle ve sistem mimarisini geliştir
- ✧ İnşa
  - Sistem tasarımı, programlama ve test.
- ✧ Geçiş
  - Sistemi, çalışacağı ortamda yayınla.

RUP iterasyonu (süreçteki tekrarlar)



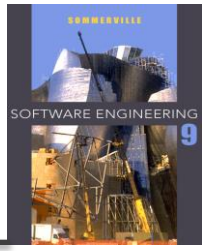
- ✧ Aşama içi iterasyon
  - Artırımlı geliştirmenin sonucu olarak her aşama iteratiftir.
- ✧ Aşamalar arası iterasyon
  - RUP modelindeki döngüde gösterildiği gibi bütün aşamaların kümesi artırımlı şekilde harekete geçirilebilir.

Rational Unified Process'de Statik İş Akışları



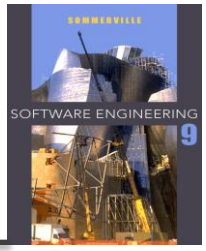
İş Akışı	Tanım
İşin Modellenmesi	Kullanım senaryoları ile iş süreçlerinin modellenmesi.
İhtiyaçlar	Sistemle etkileşecek olan kulllanıcı tiplerinin tespiti ve sistem ihtiyaçlarını modellemek için kullanım senaryolarının kullanılması.
Analiz ve Tasarım	Mimari modeller, bileşen modelleri, nesne modelleri ve diziliş modelleri kullanılarak bir tasarım modelinin yaratılması ve dokümante edilmesi.
Gerçekleştirme	Sistemdeki bileşenler, alt sistemlerin gerçekleştirilmesi şeklinde yapılandırılıp gerçekleştirilirler. Tasarım modellerinden otomatik kod oluşturan araçlar bu aşamanın hızlanmasını sağlayabilir.

Rational Unified Process'de Statik İş Akışları



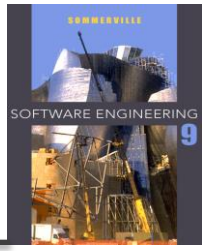
İş Akışı	Tanım
Test	Test, gerçekleştirim ile birlikte yürütülen iteratif bir süreçtir. Sistem testi ise gerçekleştirimin bitmesinden sonra yapılır.
Yayınlama	Ürünün bir sürümünün oluşturulması, kullanıcılara dağıtılması ve çalışma alanına yüklenmesi.
Konfigürasyon ve değişim yönetimi	Bu yardımcı iş akışı sistem değişikliklerini yönetir.
Proje yönetimi	Bu yardımcı iş akışı sistem gelişim sürecini yönetir.
Çevre	Bu iş akışı, yazılım geliştirme ekibine uygun yazılım geliştirme araçlarının temin edilmesi ile ilgilenir.

RUP'un kullandığı iyi uygulamalar (good practice)



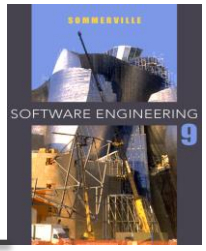
- ✧ Yazılımı İteratif Geliştir
  - Her bir küçük sürümü müşteri ihtyacının önceliğine göre geliştir ve en öncelikli ihtiyaçları ilk ulaştır.
- ✧ İhtiyaçları Yönet
  - Müşteri ihtiyaçlarını açık bir biçimde belgelendir ve ihtiyaçlardaki değişiklikleri takip et.
- ✧ Bileşen Tabanlı Mimariler Kullan
  - Sistem mimarisini yeniden kullanılabilir bileşenlerin bir kümesi olarak organize et.

RUP'un kullandığı iyi uygulamalar (good practice)



- ✧ Yazılımı Görsel Olarak Modelle
  - Yazılımın statik ve dinamik bakış açılarını oluşturmak için grafiksel UML modelleri kullan.
- ✧ Yazılımın Kalitesini Doğrula
  - Yazılımın gerekli kalite standartlarını sağladığından emin ol.
- ✧ Yazılım Değişikliklerini Kontrol Et
  - Bir tane değişiklik yönetim sistemi ve konfigürasyon yönetim sistemi kullanarak yazılımdaki değişiklikleri yönet.
    - Fazlası için: ITIL, IBM Tivoli…

Özet



- Süreçler, değişikliklerle baş edebilecek aktiviteleri barındırmalı.
  - Prototipleme gibi.

- Süreçler, artırımlı geliştirme ve teslim için tasarlanabilir. Böylece, değişikliklerin bütün sistemi etkilemesi engellenir.

- The Rational Unified Process aşamalar halinde organize edilen (başlangıç, ayrıntılandırma, inşa ve geçiş) fakat aktiviteleri de (ihtiyaçlar, analizler ve tasarım gibi) ayıran modern bir genel süreç modelidir.