Yazılım Mühendisliği
Mimari Tasarım

Bölüm 6: Mimari Tasarım

Genel Bakış

- Mimari tasarım kararları
- Mimari görünümler
- Mimari desenler
- Uygulama mimarileri

Not : Bu belge içinde numarası verilen şekilleri ders kitabınızdan takip etmelisiniz.

Mimari Tasarım

- Yazılım projelerinde Mimari tasarım, yazılım sistemini oluşturan bileşenlerin kendi içinde nasıl organize edilmesi gerektiğini, anlama ve sistemin genel yapısının tasarım sürecidir.
- Yazılım geliştirme sürecinde tasarım aşaması mimari tasarım ile başlar.
- Mimari tasarım gereksinim mühendisliği aşaması ile tasarım aşaması arasında yer alır.
- Yazılım sisteminin temel yapısal bileşenleri ve bileşenlerin aralarındaki ilişkileri tanımladığı için kritik aşamadır.

Mimari Tasarım

- Mimari tasarım sürecinin çıktısı, sistemin etkileşimli bileşenlerinin nasıl yapılandırıldığının mimari bir modeldir.
- Plan güdümlü yazılım geliştirme süreçlerinin yanı sıra Çevik süreçlerde de erken aşamada yazılım mimarisi tasarlama benimsenmiştir.
- Yazılım mimarisi için artımlı geliştirme yaklaşımı sorunlara neden olabilir.
- Sistemdeki birçok bileşeni etkilediği için sistem mimarisini yeniden düzenlemek genellikle pahalıdır.
- Şekil 6.1 örnek bir sistem mimarisini göstermektedir.

Mimari Tasarım

Yazılım mimarisini doğrudan tasarlama ve belgelemenin avantajları:

- Paydaş iletişimi, mimari, sistem paydaşları tarafından bir tartışma odağı olarak kullanılabilir.
- Sistem Analizi, Sistemin işlevsel olmayan gereksinimlerini karşılayıp karşılayamayacağının analizinin mümkün olduğu anlamına gelir.
- Büyük ölçekli yeniden kullanım, Mimari, benzer bir dizi başka sistemde tekrar kullanılabilir.

Mimari Tasarım

Yazılım mimarisinin gösterimi

- Sistem mimarisi şekil 6.1 de görüldüğü gibi biçimsel olmayan blok diyagramlar ile modellenir.
- Her kutu bir bileşeni gösterir.
- Kutu içindeki daha küçük kutular ise alt bileşenleri ifade eder.
- Oklar bileşenden bileşen taşınan verinin yönünü gösterir.

Blok diagramları;

- sistem yapısının, sistem geliştiricilerin anlayabilecekleri şekilde üst seviyeden bir resmini gösterir.
- tasarım sürecindeki insanlar arasında iletişim için bir yoldur.

Mimari tasarım kararları

- Mimari tasarım yaratıcı bir süreçtir, bu nedenle süreç geliştirilmekte olan sistemin tipine bağlı olarak değişir.
- Bununla birlikte, bir dizi ortak karar tüm tasarım süreçlerini kapsar ve bu kararlar sistemin işlevsel olmayan özelliklerini etkiler.
- Sistem mimarları sistemi ve geliştirme sürecini etkileyen yapısal kararlar verirler.
- Sistem mimarları şekil 6.2 de belirtilen sorulara tecrübelerine dayanarak cevap ararlar.

Mimari tasarım kararları

- Benzer alandaki yazılım sistemleri genellikle alana özel kavramları yansıtır ve benzer mimarilere sahip olur.
- Uygulama ürün hatları, belirli müşteri gereksinimlerini yerine getiren versiyonlara sahip bir çekirdek mimari üzerine inşa edilir.
- Bir sistemin mimarisi, bir veya daha fazla mimari desen veya "stil" etrafında tasarlanabilir.
- Fonksiyonel olmayan gereksinimler ile yazılım mimarisi ilişkilidir.
- Fonksiyonel olmayan gereksinimler, mimari stil ve yapının seçimine etki eder.

Mimari tasarım kararları

Sözü edilen fonksiyonel olmayan gereksinimler:

- Performans: Kritik işlemler yerelleştirilmeli ve iletişimi en az seviyede olmalı. Bir çok küçük bileşen yerine daha az sayıda büyük bileşenler kullanılmalı.
- Güvenlik: İç katmanlarda kritik varlıkların olduğu katmanlı bir mimari kullanılmalı.
- Emniyet: Mimaride güvenlik açısından kritik öneme sahip özellikler az sayıda alt sisteme yerelleştirilmeli.
- Erişilebilirlik: Kritik gereksinimlere hata toleransı için yedek bileşenler ve mekanizmalar eklenmeli.
- Bakım kolaylığı: Küçük parçalı, değiştirilebilir bileşenler kullanılmalı.

Mimari görünümler

- Yazılım sistem mimari tasarımı ve belgelendirme için hangi görünümler veya perspektifler faydalıdır ?
- Mimari modelleri göstermek için hangi notasyonlar kullanılmalıdır ?
- Her mimari model, sistemin yalnızca bir görünümünü veya perspektifini gösterir;
  - Yazılımın modüllere nasıl ayrıştırıldığını,
  - çalışma zamanı süreçlerinin etkileşimini,
  - sistem bileşenlerinin ağ üzerinden dağıtılma yollarını.

Mimari görünümler

- Şekil 6.3'te görüldüğü gibi bir yazılım mimari modeli ifade etmek için 4 farklı mimari görünüm olası önerilir.

Bunlar :

- Mantıksal görünüm : Soyutlamaları nesne veya nesne sınıfı olarak,
- Süreç görünümü : Yazılımın etkileşen süreçlerinin yapılanmasını,
- Geliştirme görünümü : Yazılımın geliştirme için nasıl ayrıştırıldığını,
- Fiziksel görünüm : Sistem donanım ve yazılım bileşenlerinin işlemciler arasında nasıl dağıtıldığını,

gösterirler.

Mimari desenler

- Yazılım mühendisliğinde desenler bilgiyi temsil etmenin, paylaşmanın ve yeniden kullanmanın bir yolu olmuştur.
- Mimari desen, farklı ortamlarda başarılı olmuş iyi tasarım uygulamasının genelleştirilmiş soyut bir betimlemesidir.
- Desenler, yararlı oldukları ve olmadıkları durumlar hakkında bilgi içermelidir.
- Desenler, şekil 6.4, 6.5 ve 6.6' da görüldüğü gibi diyagram ve yazılı öykü biçiminde temsil edilebilir.

Mimari desenler/Katmanlı mimari

- Mimarı tasarım bakımından, ayrıştırma ve bağımsızlaştırma iki temel kavramdır.
- Katmanlı mimari alt sistemlere ait arayüz modellemesi için kullanılır.
- Sistem her biri bir dizi hizmet sunan bir dizi katman şeklinde düzenlenir.
- Farklı katmanlardaki alt sistemlerin artımlı geliştirimini destekler.
- Mimari değiştirilebilir ve taşınabilir yapıya sahiptir.
- Katman arabirimi değiştiğinde, yalnızca komşu katman etkilenir.

Mimari desenler/ Katmanlı mimari

- Mimarı tasarım bakımından, ayrıştırma ve bağımsızlaştırma iki temel kavramdır. Bu mimari desen şekil 6.7’ de verilmiştir.
- Katmanlı mimari alt sistemlere ait arayüz modellemesi için kullanılır.
- Sistem her biri bir dizi hizmet sunan bir dizi katman şeklinde düzenlenir.
- Farklı katmanlardaki alt sistemlerin artımlı geliştirimini destekler.
- Mimari değiştirilebilir ve taşınabilir yapıya sahiptir.
- Katman arabirimi değiştiğinde, yalnızca komşu katman etkilenir.

Mimari desenler/Katmanlı mimari

- Şekil 6.8’ de dört katmana sahip bir katmanlı mimari örneği verilmiştir.
- Katmanlı mimari desenine ait bir başka örnek şekil 6.9’ da gösterilmiştir.

Mimari desenler/ Ambar mimarisi

- Bu mimari model, verinin bir bileşen tarafından oluşturulduğu ve bir başka bileşen tarafından kullanıldığı uygulamalar için uygundur.
- Bunun için alt sistemler veri alışverişi yapmalıdır ve bu iki şekilde yapılabilir.
  - Paylaşılan veriler tüm alt sistemler tarafından erişilebilen merkezi bir veri tabanında yada ambarda tutulur. Büyük miktarlarda veri paylaşılacak olduğunda bu model kullanımı yaygındır.
  - Alt sistemler kendisinin sorumluluğunda olan veri tabanına sahiptir ve verileri talep eden diğer alt sistemlere geçirir.
- Şekil 6.10’ da bir ambar deseni verilmiştir.

Mimari desenler/ İstemci sunucu mimarisi

- Verilerin ve veri işlemenin bileşenlere dağıtıldığı sistem modelidir. Bu sistemin ana bileşenleri;
  - Belirli hizmetleri sağlayan bağımsız sunucular kümesi,
  - Bu hizmetleri çağıran istemciler kümesi,
  - İstemcilerin sunuculara erişmesini sağlayan bilgisayar ağı.
- Ayrıklık ve bağımsızlık sağlar,
- Sistemin tamamı etkilenmeden servisler ve sunucular değiştirilebilir,
- İstemcilerin sunucular ve servis isimlerini bilmesi gerekebilir.
- Şekil 6.12’ de istemci sunucu deseni verilmiştir. Şekil 6.13 örnek bir istemci sunucu mimariyi göstermektedir.

Mimari desenler/Boru ve Süzgeç mimarisi

- Bir süreçten diğerine veri akımı sağlayan boru ve süzgeç kavramı Unix işletim siteminden gelmektedir.
- Bu yüzden boru ve süzgeç modeli olarak adlandırılabilir.
- Fonksiyonel dönüşümler çıktı üretmek için girdi verilerini işlerler.
- Bu yaklaşımın farklı varyantları çok yaygındır.
- Yaygın olarak kullanılan faturalama sistemleri gibi dönüşümlerin sıralı olduğu, yığınlar halinde veri işleme sistemlerinde model 'yığın sıralı model' olarak isimlendirilir.
- Şekil 6.14' de boru süzgeç deseni verilmiştir. Şekil 6.15'te örnek bir boru süzgeç mimarisi gösterilmektedir.

Uygulama Mimarileri

- Uygulama sistemleri organizasyonel bir ihtiyacı karşılamak üzere tasarlanmıştır.
- İşletmelerin ortak noktası çok olduğundan, uygulama sistemleri de uygulama gereksinimlerini yansıtan ortak bir mimariye sahip olma eğilimindedir.
- Genel bir uygulama mimarisi, bir tür yazılım sistemi için bir mimaridir.
- Uygulama mimarisi belirli gereksinimleri karşılayan bir sistem oluşturmak üzere yapılandırılabilir ve uyarlanabilir.

Uygulama Mimarileri

- Bir yazılım tasarımcısı, uygulama mimari modellerini çeşitli şekilde kullanabilir. Bunlar;
- Mimari tasarım için bir başlangıç noktası olarak,
- Bir tasarım kontrol listesi olarak,
- Geliştirme takımının çalışmalarını organize etmenin bir yolu olarak,
- Bileşenleri yeniden kullanım için değerlendirme aracı olarak,
- Uygulama türleri hakkında konuşmak için bir nevi sözlük olarak

kullanılabilir.

Uygulama Mimarileri

- Bir çok farklı uygulama sistemi tipi vardır.

  Genel olarak uygulamalar;
- Hareket işleme sistemleri,
- Bilgi sistemleri,
- Dil işleme sistemleri

  tipinde olabilir.

Uygulama Mimarileri/Hareket işleme sistemleri

- Hareket işleme sistemleri: Veri tabanlarından bilgi alma/güncelleme taleplerini işleme için tasarlanan sistemlerdir.
  - E-ticaret sistemleri,
  - Rezervasyon sistemleri vb.
- Kullanıcı açısından bakıldığında bir işlem:

  veri tabanından yapılacak herhangi bir sorgu işlemi
- Kullanıcılar, bir işlem yöneticisi tarafından asenkron olarak işlenen hizmet talebinde bulunurlar ve istek uygulamanın iş mantığına dayanan bir bileşen tarafından işlenir. Şekil 6.16 hareket işleme uygulamalarının yapısını göstermektedir.

Uygulama Mimarileri/Bilgi sistemleri

- Bilgi sistemleri, katmanlı bir mimari olarak düzenlenebilen genel bir mimariye sahiptir.
- Sistemle etkileşim genellikle veri tabanı işlemlerini içerdiğinden bunlar işlem tabanlı sistemlerdir.
- Şekil 6.18 de bilgi sisteminin genel bir modeli verilmiştir.

buradaki katmanlar şunları içerir:

- Kullanıcı arayüzü,
- Kullanıcı iletişimi,
- Bilgi alma ve değiştirme,
- Hareket yönetimi, veri tabanı.

Uygulama Mimarileri/Bilgi sistemleri

- Bilgi sistemleri, artık genel olarak kullanıcı arayüzünün web tarayıcı üzerinden çalıştığı web tabanlı sistemlerdir.
- Bu sistemler genellikle çok katmanlı istemci sunucu mimariler tipindedir.
  - Web sunucusu,
  - Uygulama sunucusu,
  - Veritabanı sunucusu.

Uygulama Mimarileri/Dil işleme sistemleri

- Dil işleme sistemleri doğal veya yapay bir dili alternatif bir gösterim biçimine çevirirler.
- Dil işleme sistemi bir programlama dili olduğu durumda çevirme işlemi yanı sıra çevrilen kodu aynı zamanda çalıştırabilir.
- Şekil 6.20'de bir programlama dili için dil işleme sistemine ait bir mimari verilmiştir.
- Şekil 6.21 de bir programlama dili derleyicisi için dil işleme sistemi ambar mimarisi görülmektedir.
- Doğal dil çevirisi yapan farklı dil işleme sistemlerinde sözlük gibi bileşenlere de ihtiyaç duyulmaktadır.

Yazılım Mühendisliği

Kaynakça

- *Ian Sommerville, Çeviri editörü: Prof Dr. N. Yasemin Topaloğlu, Yazılım Mühendisliği, Nobel akademik yayıncılık , 2018, 10. basımdan çeviri. (Ders kitabı)*

- https://iansommerville.com/software-engineering-book/slides/

Yazılım Mühendisliği
Tasarım ve Gerçekleştirme

Genel Bakış

- UML kullanarak nesne yönelimli tasarım
  - Sistem bağlamı ve etkileşimler
  - Mimari tasarım
  - Nesne sınıfı belirleme
  - Tasarım modelleri
  - Arayüz spesifikasyonları
- Tasarım desenleri

Not : Bu belge içinde numarası verilen şekilleri ders kitabınızdan takip etmelisiniz.

Tasarım ve gerçekleştirme

- Yazılım tasarımı ve gerçekleştirimi yazılım mühendisliği sürecinde, çalıştırılabilir bir yazılım sisteminin geliştirildiği aşamadır.

- Büyük yazılım sistemlerinin geliştirilmesinde, yazılım mühendisliği diğer etkinlikleri ile bütün olarak yürütülür.

- Daha az karmaşık olan küçük yazılım sistemlerinin geliştirilmesinde ise diğer yazılım mühendisliği etkinlikleri ile birleştirilmesi yoluna gidilebilir.

Tasarım ve gerçekleştirme

- Yazılım tasarımı ve gerçekleştirme etkinlikleri birbirini izler.
- Yazılım tasarımı, müşterinin gereksinimlerine göre yazılım bileşenlerinin ve ilişkilerinin tanımlandığı yaratıcı bir etkinliktir.
- Tasarım geliştirilirken, gerçekleştirme sürecinin dikkate alınması, gerçekleştirme aşamasında karşılaşılacak sorunları engeller(örnek yapılan tasarım ile gerçekleştirme aracı olan programlama dili yeteneklerinin uyumlu olmaması gibi).
- Gerçekleştirme, tasarımı bir programlama dili aracılığı ile kaynak koda ve sonrasında da çalıştırılabilir programa dönüştürme sürecidir.

Tasarım ve gerçekleştirme

- Bir yazılımın geliştirme projesinin, planlama aşamasında gerçekleştirilen, yapılabilirlik çalışması sürecinde maliyetlerin çıkarılması da yapılır.

- Planlama aşamasında verilmesi gereken kararlardan birisi de yazılımı geliştirmenin mi yoksa mevcut hazır uyarlanabilir bir yazılımı satın almanın mı doğru olacağıdır(örneğin hazır bir Hastane Bilgi Sistemi satın alınarak kurumun ihtiyaçlarına uyarlanabilir).

- Hazır yazılım satın almak hem ucuz olacak hem de daha hızlı yazılım kullanımına başlanabilecektir.

Tasarım ve gerçekleştirme

- Önceden mevcut yazılımın bileşenlerinin yeniden kullanımı yoluyla bir uygulama geliştirildiği durumlarda yeni ürün içine mevcut bileşenlerin nasıl uyarlanıp yapılandırılacağı ile ilgilenilir.

- Program tasarımı ve gerçekleştiriminin kodlamaya yönelik olan kurallarını bildiğiniz varsayılarak (kurum bazlı kodlama standartlarının varlığı, örnek bkz http://nws.noaa.gov/oh/hrl/developers_docs/General_Software_Standards.pdf, hata yakalama düzeltme yöntemleri vb.) bu bölümde sistem modelleme ve mimari tasarımın nesne yönelimli yazılım tasarımında nasıl yer bulduğuna değinilmektedir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Bağımsız varlıklar olan nesneler hem veri hem de veriyi işlemek için yöntemler bulundurur.
- Veriye erişim şekilleri ve yöntemleri tanımlanmalıdır.
- Nesne yönelimli tasarım sürecinde nesnelerin sınıflarının ve sınıflar arası ilişkileri tasarlanır.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Gerçek dünya nesneleri ile yazılımdaki soyut nesneler arasında bir bağlantı vardır.
- Bu bağlantıların açık anlaşılır olması bakım sürecinde bakım kolaylığı sağlayacaktır.
- Farklı gruplar tarafından geliştirilen büyük sistemler için tasarım modelleri önemli bir iletişim mekanizmasıdır.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Sistem tasarımını kavramdan tasarıma dönüştürmek için kullanılan çeşitli nesne yönelimli tasarım süreçleri olmakla birlikte genel olarak bu süreçlerdeki ortak işlemler:
  - Sistemin bağlamı, dış etkileşimleri tanımlanır,
  - Sistem mimarisi tasarlanır,
  - Sistemin temel nesneleri belirlenir,
  - Tasarım modeller geliştirilir,
  - Nesne arayüzleri ayrıntıları belirlenir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Tasarım tekrarlamalı ve yaratıcı süreçtir.
- Tasarımın üzerinde çalışabilmek için UML gibi gösterimler kullanılır.
- Şekil 7.1’ de örnek bir sistem bağlamı örneği verilmiştir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Sistem bağlamı ve etkileşimler

- Tasarlanan yazılım ile dış ortamı arasındaki ilişkileri anlamak, gerekli sistem işlevselliğinin nasıl sağlanacağına ve sistemin çevresiyle nasıl iletişim kuracak şekilde yapılandırılacağına karar vermek için gereklidir.
- Bağlamın anlaşılması, sistemin sınırlarını belirlenmesini de sağlar.
- Sistem sınırlarının ayarlanması, hangi özelliklerin sistemde hangi özelliklerin sistemle ilişkili diğer sitemlerde olduğuna karar verilmesine yardımcı olur.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Tasarım tekrarlamalı ve yaratıcı süreçtir.
- Tasarımın üzerinde çalışabilmek için UML gibi gösterimler kullanılır.
- Bir sistem bağlam modeli, geliştirilmekte olan sistemin çevresindeki diğer sistemleri gösteren yapısal bir modeldir.
- Etkileşim modeli, sistemin kullanıldığı ortamla nasıl etkileşime girdiğini gösteren dinamik bir modeldir.
- Şekil 7.1’ de örnek bir sistem bağlamı verilmiştir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Bir sistemin ortamı ile etkileşimleri modellenmesi için ayrıntıdan yoksun soyut bir yaklaşım kullanılmalıdır.

- Bunu yapmanın iyi bir yolu olarak ta örneği şekil 7.2' de verilen kullanım durumu modeli kullanmaktır.

- Aynı örneğin kullanım durumu tanımı şekil 7.3' te verilmiştir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Mimari tasarım

- Bir önceki aşamada sistem ve çevresi arasındaki etkileşimler anlaşıldıktan sonra, buradan yola çıkarak bu bilgiler ile sistem mimarisi tasarlanır.
- Sistemi oluşturan temel bileşenleri ve bunların etkileşimleri tanımlanır.
- Sonrasında sistemi oluşturan bileşenler katmanlı veya istemci-sunucu modeli gibi bir mimari desen kullanarak tasarlanır.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Bağımsız alt sistemlerden oluşan üst düzey mimari tasarım şekil 7.4’ te verilmiştir.

- Bu sistemin alt sistemlerinden birine ait mimari ise şekil 7.5’ te verilmiştir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Nesne sınıfı belirleme

- İyi bir tasarım için sistemi iyi anlamış olmak gerekir. Bu süreçte sistemdeki temel nesneler hakkında fikir sahibi olunmalıdır.
- Kullanım durumu tanımı, sistemin nesne ve fonksiyonlarının belirlenmesi için kullanılır.
- Nesne sınıflarını tanımlamak genellikle nesne yönelimli tasarımın zor bir aşamasıdır.

Tasarım ve gerçekleştirme

UML kullanarak nesne yönelimli tasarım

- Nesne tanımlama, sistem tasarımcılarının beceri, deneyim ve alan bilgisine dayanır.
- Nesne tanımlama tekrarlı bir süreçtir.
- Bir defada nesne tanımlamalarını doğru yapmak çok mümkün olmaz.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Nesne sınıflarının belirlenmesi için farklı yollar önerilmiştir bunlar:
  - Sistemin doğal dil ile yapılmış tanımının dil bilgisel bir analizinin kullanılması.
  - Tanımlamayı uygulama alanındaki somut şeylere dayandırılması.
  - Senaryo tabanlı bir analiz kullanılması. Her senaryodaki nesneler, öznitelikler ve yöntemler belirlenmelidir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Ders kitabınızda kullanılan hava durumu istasyonu örneğinde nesne belirlenmesi sitemdeki somut donanıma ve verilere dayanmaktadır. Örneğe ait nesne sınıfları şekil 7.6’ da verilmiştir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Burada:

Yer termometresi, *Rüzgarölçer*, ve *Basınçölçer* sistemdeki teçhizat ile ilgili olan uygulama alanı nesneleridir. Bu sebeple kullanım durumu modelinde tanımlanan etkileşimleri yansıtır.

HavaDurumuİstasyonu ise hava durumu istasyonunun çevresine esas arayüz iken *HavaDurumuVerisi* teçhizatlardan(*Yer termometresi*, *Rüzgarölçer* ve *Basınçölçer nesneleri*) özetlenmiş verileri kapsar.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Tasarım modelleri

- Tasarım modelleri nesneleri ve nesne sınıflarını ve aralarındaki ilişkileri gösterir.
- Tasarım modeli için UML kullanıldığında İki tip tasarım modeli geliştirilmelidir, bunlar:
  - *Yapısal modeller*, nesne sınıfları ve aralarındaki ilişki bakımından sistemin durağan yapısını tanımlar.
  - *Dinamik modeller*, nesneler arasındaki dinamik etkileşimleri tanımlar.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- Kullanım durumu ve mimari modellere ayrıntı eklemek için üç UML model türü kullanışlı olacaktır bunlar:
  - *Alt sistem modelleri*, nesnelerin uyumlu alt sistemler şeklinde mantıksal guruplarını gösterir. Alt sistem modelleri durağandır.
  - *Sıra modelleri*, nesne etkileşiminin sırasını gösterir. Sıra modelleri dinamik modellerdir.
  - *Durum makinesi modelleri*, nesnelerin tek tek olaylara yanıt olarak durumlarını nasıl değiştirdiğini gösteren modellerdir. Durum makinesi modelleri dinamik modellerdir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Alt sistem modelleri, tasarımın ilgili nesne grupları şeklinde nasıl düzenlendiğini gösterir. Şekil 7.4 alt sistemleri göstermek için verilmiştir. Alt sistem modellerine ilave olarak ayrıntı nesne modelleri de verilebilir.

- Kalıtım modelleri,
- Genelleştirme modelleri,
- Birleştirme modelleri gibi.

Yararlı link : https://www.omg.org/spec/UML

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Sıra modelleri, gerçekleşen nesne etkileşimlerinin sırasını gösterir. Sıra modellerinde;

- Nesneler üstte yatay olarak düzenlenmiştir;
- Zaman dikey olarak gösterilir, böylece modeller yukarıdan aşağıya okunur;
- Etkileşimler etiketli oklarla gösterilir, farklı ok stilleri farklı etkileşim türlerini gösterir;
- Nesne, sistemdeki kontrol edici nesne olduğunda nesne yaşam çizgisindeki ince dikdörtgen süreyi gösterir.

Şekil 7.7' de UML sıra diyagramı hava istasyonu veri toplanmasını göstermektedir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

- *Durum diyagramları*, nesnelerin aldığı isteklere nasıl tepki verdiğini ve bu isteklerin tetiklediği durum geçişlerini göstermek için kullanılır.
- *Durum diyagramları*, bir sistemin veya bir nesnenin çalışma zamanı davranışının üst düzey modelleridir.
- Sistemdeki basit nesneler için gereksiz durum diyagramı eklemek, tasarıma gereksiz ayrıntılar ekler.
- Şekil 7.8' de örneğe ait durum diyagramı verilmiştir.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Arayüz spesifikasyonları

- Nesneye yönelik tasarım sürecinde bileşenler arasındaki arayüzlerin tanımlanması gerekir.
- Nesnelerin ve diğer bileşenlerin paralel olarak tasarlanabilmesi için nesne arabirimlerinin belirtilmesi gerekir.
- Arayüz tasarım spesifikasyonunda öznitelikler tanımlanmadığı için veri gösterim detayı arayüz tasarımına katılmamalıdır.

Tasarım ve gerçekleştirme
UML kullanarak nesne yönelimli tasarım

Arayüz spesifikasyonları

- Veriye erişim ve güncelleme için gereken operasyonlar ise arayüz tasarımında yer almalıdır.
- Bir nesne, sağlanan yöntemlerin bakış açısı olan birkaç arayüze sahip olabilir.
- UML, arayüz özellikleri için sınıf diyagramları kullanır.

Şekil 7.9 örnek hava istasyonu için arayüzler gösterilmiştir.

Tasarım ve gerçekleştirme

Tasarım desenleri

- Tasarım desenleri, önceden projelerde kullanılmış ve başarılı olmuş çözümlerin ortak problemlerde yeniden kullanımı için problemin tanımı ve çözümünü içeren kalıplardır.
- Desenler nesne yönelimli yazılım tasarımına önemli katkı sağlamaktadır.
- Desenler bir tasarımı desen üzerinden açıklarken sözlük olarak da kullanılır.
- İlk desen tanımlamaları 1995 yılında «dörtler çetesi»(gang of four) tarafından yazılan desenler kitabında ve 1997 Siemens'te çalışanlar tarafından yayınlanmıştır.

Tasarım ve gerçekleştirme

Tasarım desenleri

- Tasarım deseni, tasarımcıların başarılı çözümlerden elde ettikleri bilgi ve deneyimlerini, yeniden kullanılması amacıyla paylaşmanın bir nevi yolu olmuştur.
- Bir desen, sorunun tanımı ve çözümünün özüdür.
- Farklı ortamlarda tekrar kullanılmak için yeterince soyut olmalıdır.
- Desen tanımları, sıklıkla kalıtım ve çok şekillilik gibi nesne yönelimli özelliklerden faydalanır.

Tasarım ve gerçekleştirme

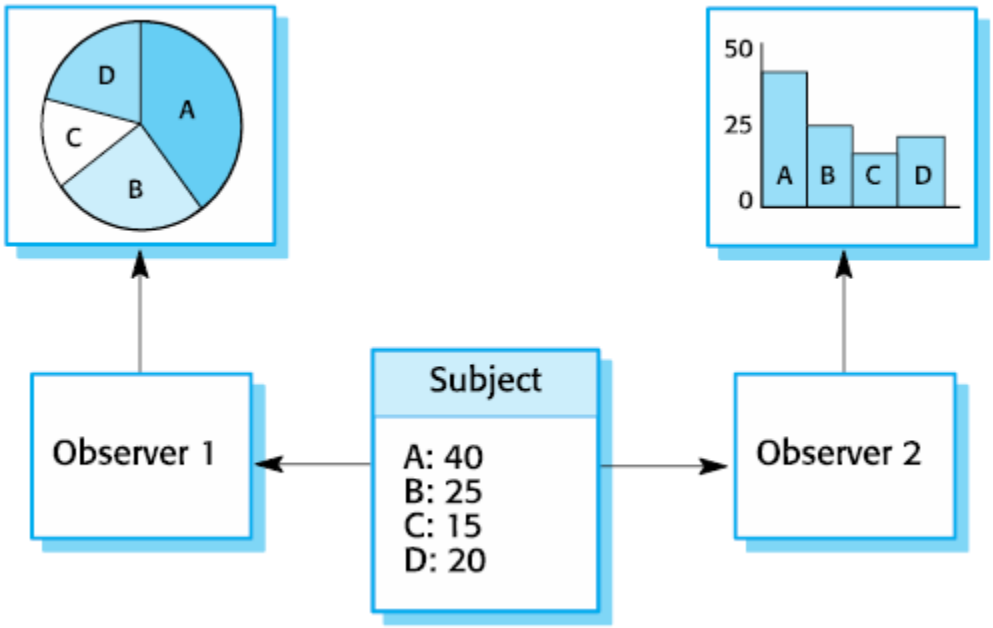
Tasarım desenleri

- Tasarım desen öğeleri,
  - İsim : deseni anlamlı olarak tanımlayacak bir isim,
  - Problem açıklaması: desenin ne zaman uygulanabileceğinin bir açıklaması,
  - Çözüm tanımı: çözümü oluşturan parçaların genellikle nesne ve nesneler arası ilişkilerini gösteren grafiksel gösterim.
  - Sonuçlar: Desenin uygulanmasının sonuçları ve avantaj ve dezavantajları.

Tasarım ve gerçekleştirme

Tasarım desenleri

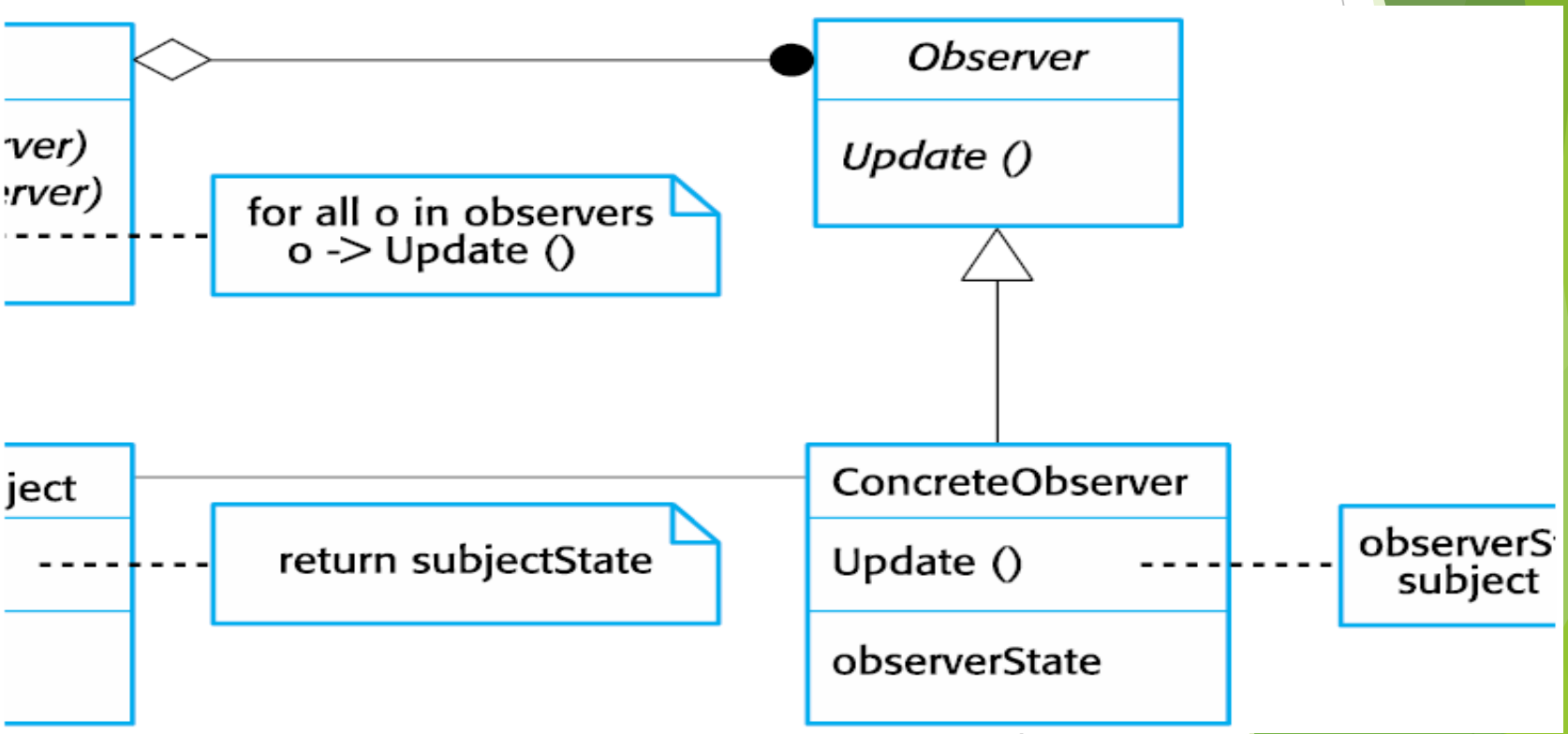
- Dörtler çetesinin desenler kitabından alınan şekil 7.1'de tanımı verilen (Observer) Gözlemci isimli deseninin grafiksel gösterimi aşağıda verilmiştir.



Tasarım ve gerçekleştirme

Tasarım desenleri

- (Observer) Gözlemci isimli desene ait UML modeli ise aşağıda yer almaktadır.



Yazılım Mühendisliği

Kaynakça

- *Ian Sommerville, Çeviri editörü: Prof Dr. N. Yasemin Topaloğlu, Yazılım Mühendisliği, Nobel akademik yayıncılık , 2018, 10. basımdan çeviri. (Ders kitabı)*

- https://iansommerville.com/software-engineering-book/slides/

Yazılım Mühendisliği Gerçekleştirme

Bölüm 7: Tasarım ve Gerçekleştirim

Genel Bakış

- Gerçekleştirim konuları
  - Yeniden kullanım
  - Konfigürasyon yönetimi
  - Konakçı hedef geliştirme
- Açık kaynak geliştirme
  - Açık kaynak lisanslama

Tasarım ve **gerçekleştirme**

- Bu anlatımda, gerçekleştirim aşamasında önemli olan kodlamaya yönelik ilkeler konusu ele alınmamıştır.

  Aşağıda belirtilen uygulama sorunları üzerinde durulacaktır.

Bunlar;

- Yeniden kullanım
- Yapılandırma yönetimi
- Konakçı hedef geliştirme
- Açık kaynak geliştirme

https://introcs.cs.princeton.edu/java/11style/

Tasarım ve **gerçekleştirme**

- Yeniden kullanım: Çoğu modern yazılım, mevcut tasarım, bileşenler veya sistemler yeniden kullanılarak oluşturulur. Yazılım geliştirirken, mevcut kaynaklardan mümkün olduğunca fazla yararlanmalısınız.

- Yazılım yeniden kullanımı düzeyleri
  - *Soyutlama düzeyi*: Bu seviyede, yazılım kodu, bileşeni vs. tekrar kullanılmaz, tasarımda başarılı olunmuş soyutlamalar bilgisi kullanılır.

  - *Nesne düzeyi*: Hazır kod yani doğrudan bir kütüphanedeki nesneler yeniden kullanılır.

Tasarım ve **gerçekleştirme**

- Yazılım yeniden kullanımı düzeyleri
  - *Bileşen düzeyi*: Bileşenler, yeniden kullanılan nesne ve nesne sınıfları koleksiyonlarıdır. Örneğin belli bir işlevi yerine getirmek için mevcut hazır bir ekran.
  - *Sistem düzeyi*: Bu düzeyde, tüm uygulama sistemi yeniden kullanılır.

Tasarım ve **gerçekleştirme**

- Yazılım yeniden kullanımının avantajları
  - Yazılımı hızlı bir şekilde geliştirme,
  - Düşük geliştirme maliyeti,
  - İlk defa geliştirilen yazılama göre daha güvenilir
  - Geliştirme riski azalması

Tasarım ve **gerçekleştirme**

- Yazılım yeniden kullanımının maliyetleri
  - Zaman maliyeti,
  - Satın alma maliyeti,
  - Yapılandırma maliyeti,
  - Bütünleştirme maliyeti.

Tasarım ve **gerçekleştirme**

- Konfigürasyon yönetimi: Yazılım geliştirme sürecinde değişen bir yazılım sisteminin değişimlerinin yönetimi sürecine konfigürasyon yönetimi adı verilir.

- Konfigürasyon yönetiminin amacı, tüm geliştiricilerin proje koduna ve belgelerine kontrollü şekilde erişebilmesi, hangi değişikliklerin yapıldığını öğrenebilmesi ve bir sistem oluşturmak için bileşenleri derleyip bağlayabilmesi için sistem bütünleştirme sürecini desteklemektir.

Tasarım ve **gerçekleştirme**

- Konfigürasyon yönetim sürecinde 4 faaliyet yapılır;
  - *1.Sürüm yönetimi*: Yazılım bileşenlerinin farklı sürümlerini takip etmek için destek sağlanır.
  - Sürüm yönetim sistemleri, farklı programcıların koordineli olarak geliştirim yapabilmesi için imkan sunar.
- Örnek sürüm kontrol yazılım araçları,
  - Concurrent Versions System (CVS)
  - GIT version

Tasarım ve **gerçekleştirme**

- 2. *Sistem bütünleştirme*: Bir sistemin her bir sürümü için bileşenlerin hangi sürümlerinin kullanıldığının tanımlanmasında geliştiriciler destek sağlanması faaliyetidir.

Bu tanımlama daha sonra gerekli bileşenleri derleyerek ve bağlayarak sistemi otomatik olarak oluşturmak için kullanılır.

Tasarım ve **gerçekleştirme**

- *3.Problem izleme*: Problem izleme faaliyetinde, kullanıcıların hataları ve diğer sorunları bildirmesine ve tüm geliştiricilerin, mevcut bu sorunlar üzerinde kimin çalıştığını ve ne zaman düzeltildiğini görmelerine olanak sağlamak için destek sağlanmasıdır.

- *4.Dağıtım yönetimi*: Yazılımın sürümlerinin kullanım için hazırlanması ve mevcut kullanılan yazılımların izlenmesi faaliyetidir.

Tasarım ve **gerçekleştirme**

- Konakçı hedef geliştirme:
  - Çoğu yazılım bir bilgisayarda (konakçı) geliştirilir, ancak ayrı bir makinede (hedef) çalışır.
  - Daha genel olarak, bir geliştirme platformu ve bir de işletim platformundan söz edebiliriz.
  - Burada bir platform ile ifade edilen donanım, işletim sistemi, veri tabanı yönetim sistemi, etkileşimli geliştirme ortamı ve diğer destekleyici yazılımlardır.
  - Geliştirme platformu genellikle işletme platformundan farklı kurulmuş yazılıma sahiptir; bu platformların mimarileri farklı olabilir

Tasarım ve **gerçekleştirme**

- Yazılım mühendisliği süreçlerini destekleyen yazılım geliştirme platformunda aşağıdaki araçlar bulunmalıdır.
  - Derleyici ile tümleşik kod düzenleme sistemi,
  - Dilin hata ayıklama sistemi,
  - Grafiksel modelleme araçları,
  - Sınama araçları,
  - Program görselleştirme araçları,
  - Sürüm işlemleri için yapılandırma yönetim araçları.

Tasarım ve **gerçekleştirme**

- Bileşenlerin konuşlandırılacağı platforma karar verirken aşağıdaki durumlara dikkat edilmelidir bunlar;
  - Bir bileşen belli bir donanım mimarisi için tasarlanmışsa veya başka bir yazılım sistemine bağlıysa, gerekli donanım ve yazılım desteğini sağlayan bir platformda konuşlanması gerekir,
  - Yararlanılabilirliği yüksek sistemler için, bileşenlerin birden fazla platforma konuşlanması gerektirebilir. Bir platform arızası durumunda, bileşenin alternatif bir uygulamasının mevcut olduğu anlamına gelir,
  - Bileşenler arasında yüksek düzeyde iletişim trafiği varsa, bunları aynı platformda veya fiziksel olarak birbirine yakın platformlarda konuşlandırmak genellikle mantıklıdır.

Tasarım ve **gerçekleştirme**

- Açık kaynak geliştirme: Açık kaynak geliştirme, bir yazılım sisteminin kaynak kodunun yayınlandığı ve gönüllülerin geliştirme süreçlerine katılmaya davet edildiği bir yazılım geliştirme yaklaşımıdır.

- Kökleri kaynak kodun tescilli olmaması gerektiğini, ancak kullanıcıların istedikleri gibi incelemesi ve değiştirmesi için her zaman hazır olması gerektiğini savunan Özgür Yazılım Vakfı'ndadır.
  - https://www.fsf.org/

Tasarım ve **gerçekleştirme**

- Açık kaynaklı yazılım, bu yaklaşımı daha geniş bir gönüllü kitleye yaymak için interneti kullanarak genişletmiştir.

- Bazı açık kaynak yazılımlar
  - Linux işletim sistemi çeşitli versiyonları
  - Java
  - Apache web sunucusu
  - mySQL veritabanı yönetim sistemi

https://en.wikipedia.org/wiki/Category:Software_using_the_GPL_license

Tasarım ve **gerçekleştirme**

- Bir yazılım firmanız varsa açık kaynak ile yazılım geliştirir misiniz ?

Bunun için üzerinde düşünmeniz gereken konular

1. Geliştirilmekte olduğunuz yazılım ürününüzde açık kaynak bileşenlerden faydalanmalı mıyım ?

2. Yazılımın geliştirme için açık kaynak bir yaklaşım kullanılabilir miyim?

Tasarım ve **gerçekleştirme**

- Açık kaynak lisanslama: Açık kaynak geliştirmenin temel ilkesi, kaynak kodun serbestçe kullanılabilir olması gerektiğidir, ancak bu herkesin kodla istediği her şeyi yapabileceği anlamına gelmez.
- Kodu geliştiren kodun sahibidir ve kullanım için bazı şartlar koyabilir.
- Bazı açık kaynak geliştiricileri, yeni bir sistem geliştirirken açık kaynak bileşen kullanılıyorsa, onunda açık kaynak olması gerektiğine inanmaktadır.
- Diğerleri, bu kısıtlama olmadan kodlarının kullanılmasına izin vermeye isteklidir.

Tasarım ve **gerçekleştirme**

- Açık kaynak lisans modelleri
- GNU Genel Kamu Lisansı(GPL),
  - Bu, "karşılıklı" bir lisanstır, yani GPL lisansı altında lisanslanmış açık kaynaklı bir yazılım kullanıyorsanız, o yazılımı açık kaynak yapmalısınız,
- GNU Kısıtlı Genel Kamu Lisansı(LGPL),
  - bu lisans, açık kaynak koduna bağlanan bileşenleri kullanarak geliştirileceğimiz bileşenlerin kaynağını yayınlamak zorunda olmadığımız GPL lisansının bir çeşididir.

Tasarım ve **gerçekleştirme**

- Berkley Standart Dağıtım Lisansı(BSD),
  - Bu, karşılıklı olmayan bir lisanstır, açık kaynak kodunda yapılan herhangi bir değişikliği yeniden yayınlamak zorunda olmadığınız anlamına gelir. Bu lisans modelinde kodu satılan tescilli sistemlere dahil edebilirsiniz.

Tasarım ve **gerçekleştirme**

- Açık kaynak kullanan firmalar nelere dikkat etmeli
  - İndirilen ve kullanılan açık kaynaklı bileşenler hakkında bilgi sağlamak için bir sistem oluşturun,
  - Farklı lisans türlerinin farkında olun ve bir bileşenin kullanılmadan önce nasıl lisanslandığını anlayın,
  - Bileşenler için değişim farkında olun,
  - İnsanları açık kaynak konusunda eğitin,
  - Denetim sistemlerini hazır bulundurun,
  - Açık kaynak topluluğuna katılın.

Yazılım Mühendisliği

Kaynakça

- *Ian Sommerville, Çeviri editörü: Prof. Dr. N. Yasemin Topaloğlu, Yazılım Mühendisliği, Nobel akademik yayıncılık , 2018, 10. basımdan çeviri. (Ders kitabı)*

- https://iansommerville.com/software-engineering-book/slides/

Yazılım Mühendisliği
Yazılım Testi

Bölüm 8: Yazılım Testi

Genel Bakış

- Yazılım testi
  - Doğrulama ve geçerleme
  - Geliştirme testi
    - Birim test

Yazılım testi

- Yazılım testi nedir?
  - Testler, bir programın gereksinimlerde belirtilenleri karşıladığını göstermeye ve kullanılmadan önce program hatalarını ortaya çıkarmaya yönelik yapılan çalışmadır.
- Yazılım testinin amacı nedir?
  - Yazılımın hatalı, olmaması gereken, gereksinimler ile uyuşmayan yönlerini ortaya çıkarmak,
  - Yazılımın gereksinimleri karşıladığını paydaşlara göstermek.

Yazılım testi

- Geçerleme testi

  Geliştiriciye ve müşteriye yazılımın gereksinimlerini karşıladığını gösterir.

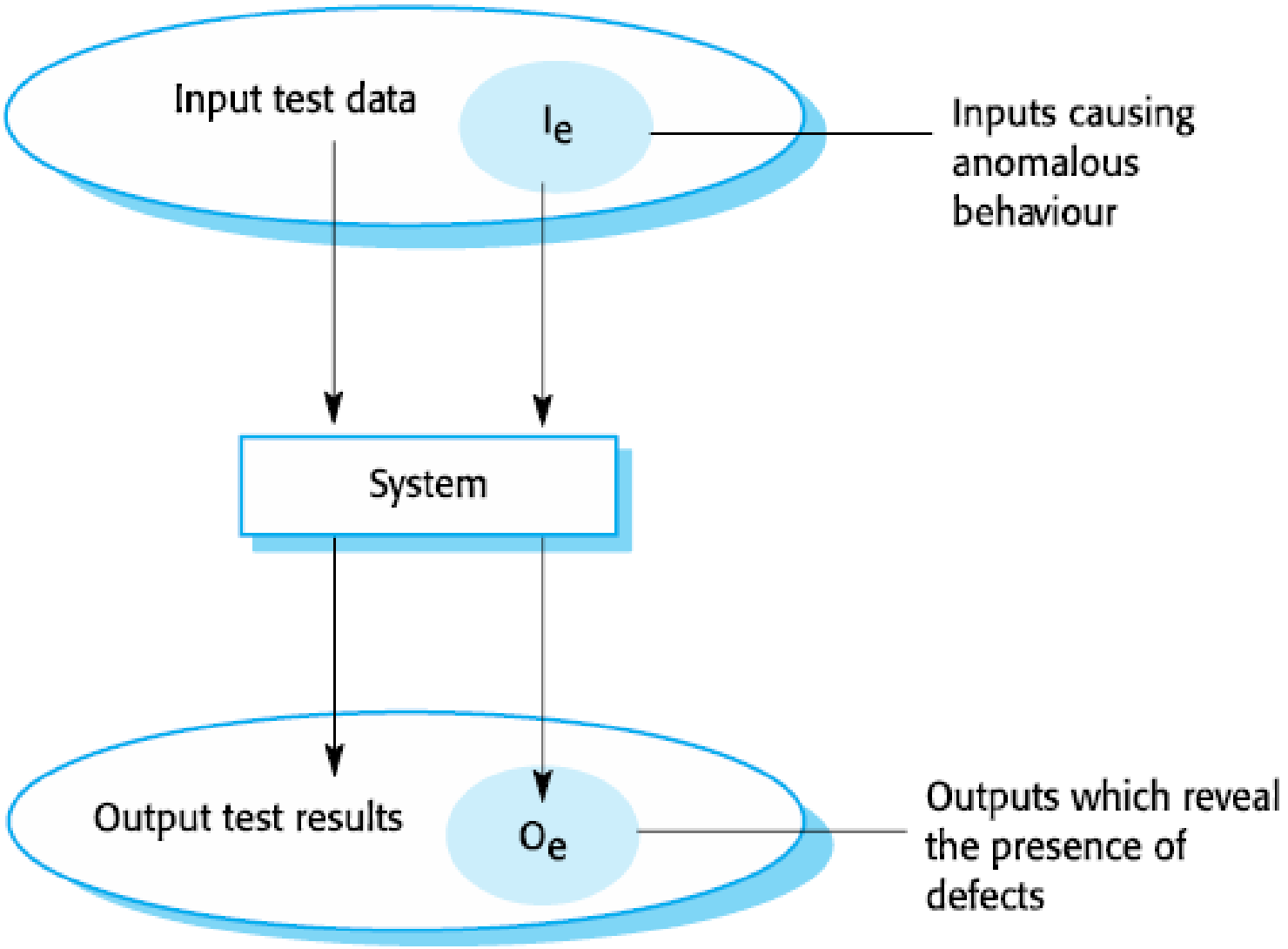
- Kusur testi

  Yazılımın, çalışmasının yanlış olduğu veya belirtimine uygun olmayan hata veya kusurları ortaya çıkarma ile ilgilenilir.

- Test işlemi için yapay veriler kullanılır.
- Programın bu yapay veriler ile çalıştırılmasından elde edilen test sonuçlar hatalar, anormallikler ve programın işlevsel olmayan özellikleri açısından kontrol için kullanılır.

Yazılım testi

Program testi girdi ve çıktı modeli



Yazılım testi

- Test etme, doğrulama ve geçerleme sürecinin bir parçasıdır.

Are we building the right product?

- **Doğrulama(Verification):** Doğru(çalışan) ürünü geliştiriyor muyuz ?
  - Yazılımın fonksiyonel ve fonksiyonel olmayan gereksinimleri karşılayıp karşılamadığını test etme sürecidir.

Are we building the product right ?

- **Geçerleme(Validation):** Ürünü doğru(şekilde) geliştiriyor muyuz ?
  - Yazılım geliştirme sürecinin her aşamasına ait çıktının, o aşamanın başında tanımlanmış koşulları karşılayıp karşılamadığını test etme sürecidir.

Yazılım testi

- Doğrulama ve Geçerlemenin amacı, sistemin amaca uygun olduğuna güven oluşturmaktır.
- Güven seviyesi sistemin amacına, kullanıcı beklentilerine ve pazarlama ortamına bağlıdır.
  - Yazılım amacı

    Güven düzeyi, yazılımın ne kadar kritik olduğuna bağlıdır.
  - Kullanıcı beklentileri

    Kullanıcıların, önceden güvenilir olmayan yazılım tecrübeleri nedeni ile belli yazılım türlerinden beklentileri düşük olabilir.
  - Pazarlama çevresi

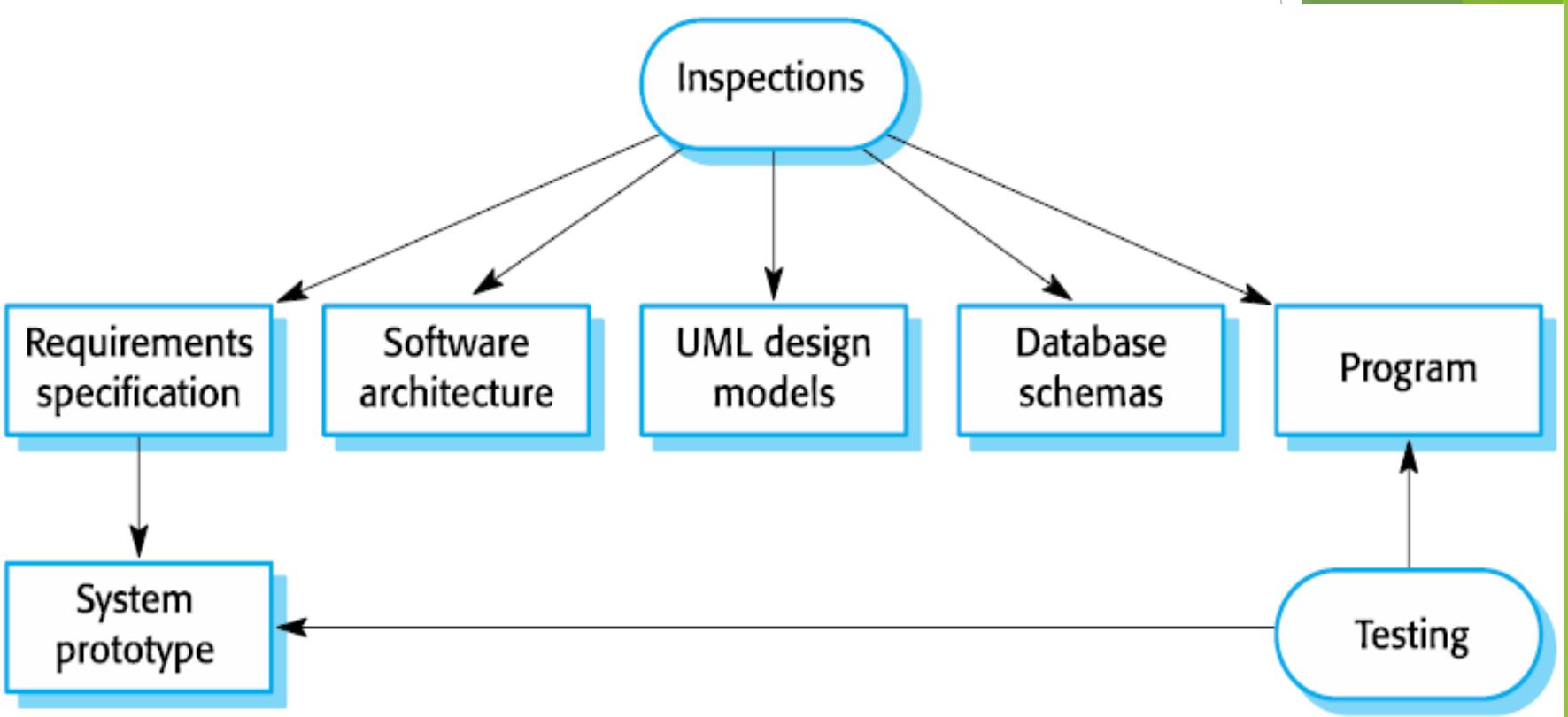
    Bir ürünü erkenden pazara sunmak, programdaki hataları bulmaktan daha önemli olabilir.

Yazılım testi

- Yazılım denetleme ve yazılım testi karşılaştırma:
  - Yazılım testinin yanı sıra, doğrulama ve geçerleme süreci yazılım denetimlerini ve incelemelerini de içerebilir.
  - Yazılım denetimleri problemleri ortaya çıkarmak için sabit sistem gösteriminin incelemesi ile ilgilidir(statik doğrulama - static verification). Denetlemeler sistemin herhangi bir gösterimine (gereksinimler, tasarım, yapılandırma verileri, test verileri vb.) uygulanabilir.
  - Denetlemeler bir sistemin çalıştırılmasını gerektirmez.
  - Program hatalarını bulmak için etkilidir ancak yazılım testi yerine geçemez.
  - Yazılım testi yazılımın çalıştırılması ve davranışlarının incelenmesi ile ilgilidir.(dinamik doğrulama - dynamic verification)

Yazılım testi

Denetlemeler ve yazılım test etme



Yazılım testi

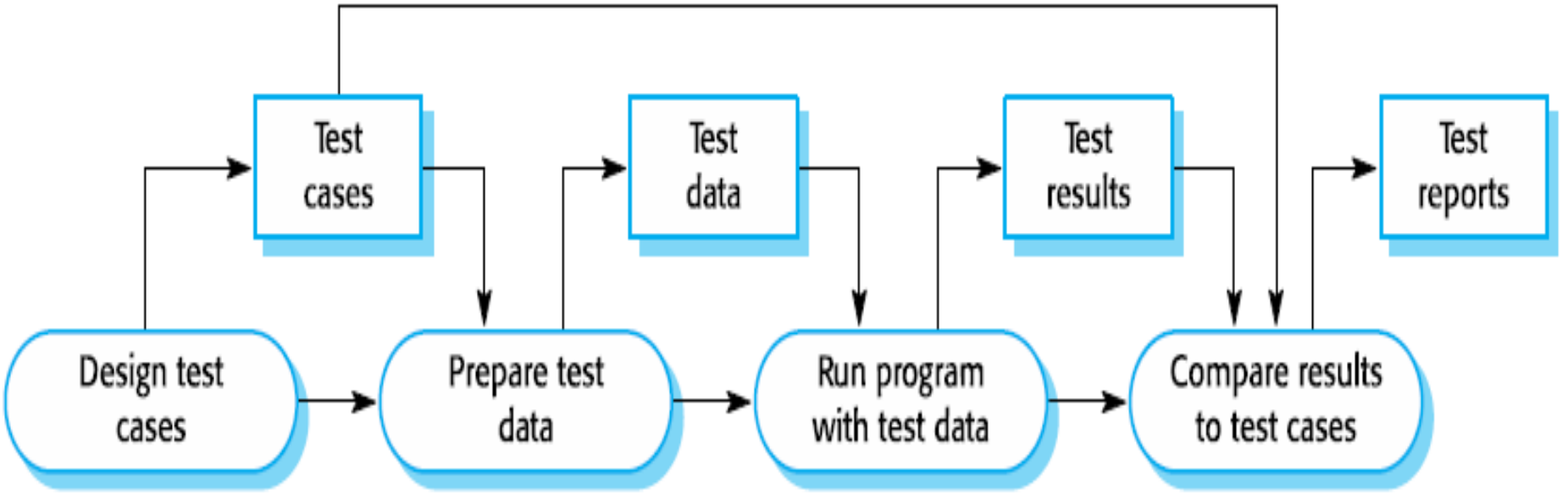
- Aşağıdakiler yazılım denetlemenin faydalarıdır:
  - Yazılım testi sırasında bir hata diğer hataları kamufle edebilirken, denetlemeler statik bir süreç olduğundan hatalar arasındaki etkileşimlerle ilgilenilmesi gerekmez.
  - Denetimler ile yazılımın tamamlanmamış sürümleri, ek maliyetler olmadan kontrol edilebilir. Bir program tamamlanmamışsa, geliştirilen kısmını test etmek için ona özel yazılım çalışma test ortamı hazırlamanız gerekir.
  - Yazılım denetimi, program kusurlarını aramanın yanı sıra, bir programın kalite özelliklerini(standartlara uygunluk, taşınabilirlik vs.) de değerlendirmeye alabilir.

Yazılım testi

- Yazılım denetleme ve yazılım testi
  - Denetlemeler ve testler tamamlayıcıdır ve doğrulama tekniklerine karşı değildir.
  - Her ikisi de Doğrulama ve Geçerleme işlemi sırasında kullanılmalıdır.
  - Denetlemeler belirtimlere uygunluğu kontrol edebilir, ancak müşterinin gerçek gereksinimlerine uygunluğu kontrol edemez.
  - Denetlemeler performans, kullanılabilirlik vb. gibi işlevsel olmayan özellikleri kontrol edemez.

Yazılım testi

Yazılım testi sürecinin modeli



Yazılım testi

- Ticari bir yazılım sistemi aşağıdaki testleri geçmelidir:
  - Geliştirme testi: Yazılım geliştirme sürecinde kusurları ortaya çıkarılmak için test edilir.
  - Dağıtım testi: Ayrı bir test ekibi tarafından, piyasaya çıkmadan önce yazılımın tam bir sürümü test edilir.
  - Kullanıcı testi: Bir yazılım kullanıcıları veya potansiyel kullanıcıları tarafından kendi ortamlarında test edilir.

Yazılım testi

- Yazılım testi üzerine
  - Yazılım test ekipleri tarafından tespit edilen hata yada kusurlar geliştiricilere rapor edilir ve geliştiriciler tarafından bir süreç dahilinde düzeltmeler yapılır.
  - Bir yazılımda herhangi bir nedenle bir değişiklik yapıldıktan sonra yazılımda değişiklik yapılan ve değişiklik yapılmayan alanlarda, varsa yeni hataları bulmak için regresyon testi ismi verilen testler yapılır.
  - Yazılım testi için test araçları kullanımı yaygınlaşmıştır.

Yazılım testi

- Geliştirme testi
- Geliştirme testi, yazılım sistemini geliştiren ekip tarafından yürütülen tüm test faaliyetlerini kapsar.
  - Birim Test : Bağımsız program birimlerinin veya nesne sınıflarının fonksiyonelliği test edilir.
  - Bileşen testi : Bağımsız birimler birleştirilerek bileşenler oluşturulur. Bu test ile bileşen fonksiyonlarına erişim sağlayan arayüzler test edilir.
  - Sistem testi : Bileşenlerin birleştirilmesiyle sistemler oluşturulur. Bu test ile daha çok bileşen etkileşimleri test edilir.

Yazılım testi

- Birim test
  - Birim test ile bileşenler ayrı ayrı kendi başına test edilir.
  - Birim test bir kusur test sürecidir.
  - Birimler şunlar olabilir:
    - Bir nesne içindeki metotlar veya bağımsız fonksiyonlar.
    - Çeşitli özniteliklere ve yöntemlere sahip nesne sınıfları.
    - Arayüzü olan bileşik bileşenler.

Yazılım testi

- Nesne sınıflarının testi
  - Bir nesne sınıfını tam test etmek için :
    - Bir nesneyle ilişkili tüm işlemleri test etme.
    - Tüm nesne özniteliklerine değer atama ve okuma.
    - Nesneyi olası tüm durumlarda çalıştırma.

Yazılım testi

- Otomatik birim test
  - Mümkün ise birim test otomatikleştirilmelidir.
  - Otomatik birim testi için JUnit gibi test çerçeveleri(framework) kullanılabilir.
  - Birim test çerçevelerinde sunulan genel test sınıfları ile yazılımınıza uygun testler oluşturabilir, çalıştırabilir ve sonuç raporlarını görebilirsiniz.

Yazılım testi

- Birim test takımlarının seçimi
  - Test takımları tanımlandığı gibi kullanıldığında, test edilen bileşenin yapması gereken işlevi yerine getirdiğini göstermelidir.
  - Test edilen bileşende kusurlar varsa, bunlar test takımları tarafından ortaya çıkarılmalıdır.
  - Bunun için iki farklı test takımı tasarlanmalı:
    - Programın normal çalışmasını yansıtmalı ve bileşenin beklendiği gibi çalıştığını göstermek için tasarlanmalıdır.
    - Kusur çıkabilecek durumları test etmeye yönelik tasarlanmalıdır.

Yazılım testi

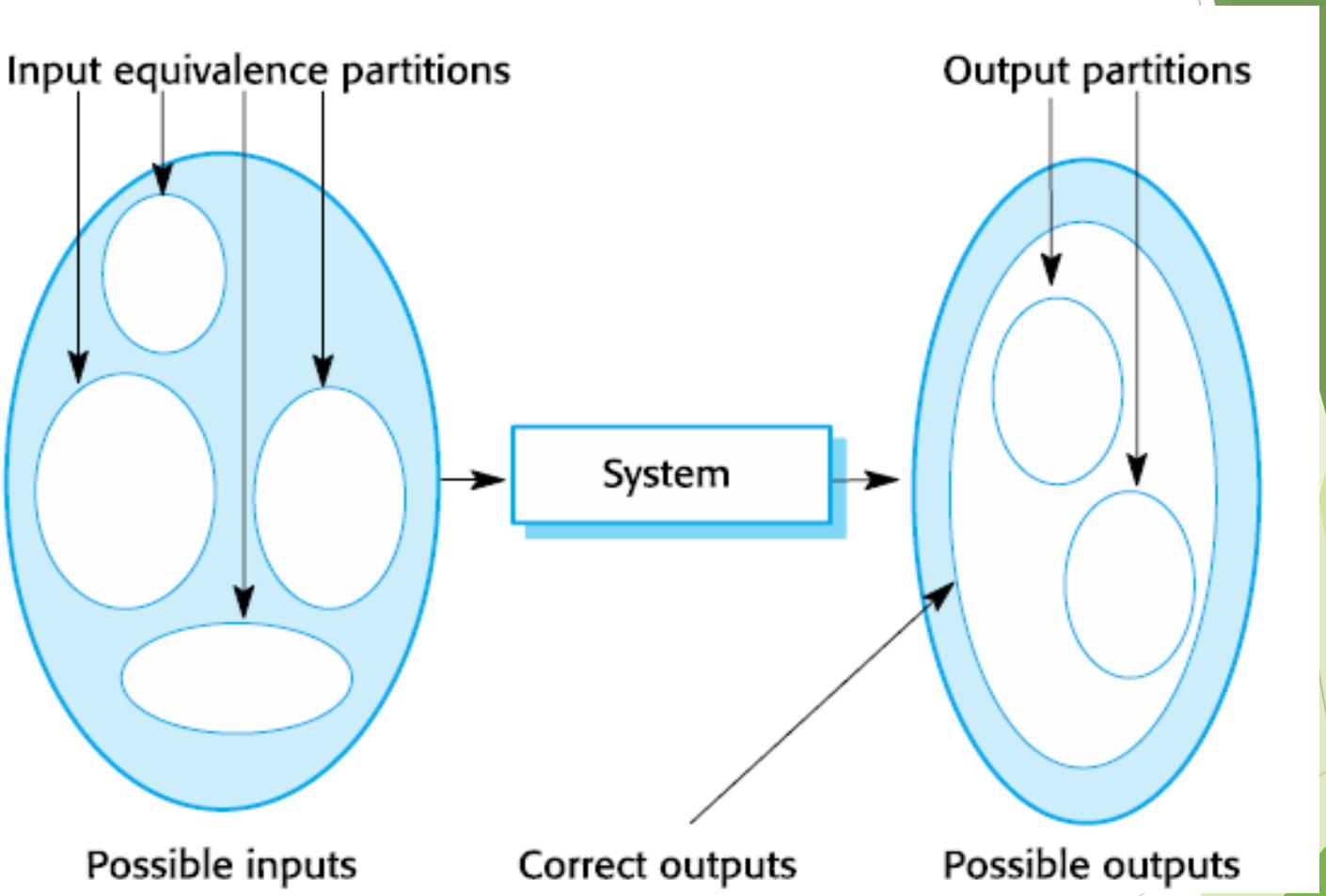
	A	B	C	D	E	F	G	H	I	J	K
1	Test Case ID		BU_001	Test Case Description		Test the Login Functionality in Banking					
2	Created By		Mark	Reviewed By		Bill		Version		2.1	
3											
4	QA Tester's Log		Review comments from Bill incorprate in version 2.1								
5											
6	Tester's Name		Mark	Date Tested		1-Jan-2017		Test Case (Pass/Fail/Not		Pass	
7											
8	S #	Prerequisites:				S #	Test Data				
9	1	Access to Chrome Browser				1	Userid = mg12345				
10	2					2	Pass = df12@434c				
11	3					3					
12	4					4					
13											
14	Test Scenario	Verify on entering valid userid and password, the customer can login									
15											
16	Step #	Step Details		Expected Results		Actual Results			Pass / Fail / Not executed / Suspended		
17											
18	1	Navigate to http://demo.guru99.com		Site should open		As Expected			Pass		
19	2	Enter Userid & Password		Credential can be entered		As Expected			Pass		
20	3	Click Submit		Cutomer is logged in		As Expected			Pass		
21	4										
22											
23											

https://www.guru99.com/test-case-vs-test-scenario.html

Yazılım testi

- Birim test takımı seçimi stratejileri
  - Bölünme testi, ortak özelliklere sahip olan ve aynı şekilde işlenmesi gereken girdi gruplarını tanımlar her birinden temsilci test alınır.
  - Kılavuzlara dayalı testte, test takımlarını seçmek için test etme kuralları kullanılır.

Yazılım testi



Yazılım testi

- Kılavuzlara dayalı test
  - Bu kılavuzlar test takımları seçimi için bilgiler içerir. Örneğin sıralar(squences), diziler, listeler içeren programlar test edilirken;
  - Yazılımı tek bir değeri olan sıralar ile test edin.
  - Farklı testlerde farklı eleman sayısı olan sıralar kullanın.
  - Testleri, sıraların ilk, orta ve son elemanlarına erişilecek şekilde tasarlayın.
  - Sıfır uzunlukta sıralar ile test edin.

Yazılım Mühendisliği

Kaynakça

- *Ian Sommerville, Çeviri editörü: Prof. Dr. N. Yasemin Topaloğlu, Yazılım Mühendisliği, Nobel akademik yayıncılık , 2018, 10. basımdan çeviri. (Ders kitabı)*

- https://iansommerville.com/software-engineering-book/slides/

Yazılım Mühendisliği
Yazılım Testi

Bölüm 8: Yazılım Testi

Genel Bakış

- Yazılım testi
  - Geliştirme testleri
    - Bileşen testi
    - Sistem testi
  - Test güdümlü geliştirme
  - Dağıtım testi
  - Kullanıcı testi

Yazılım testi

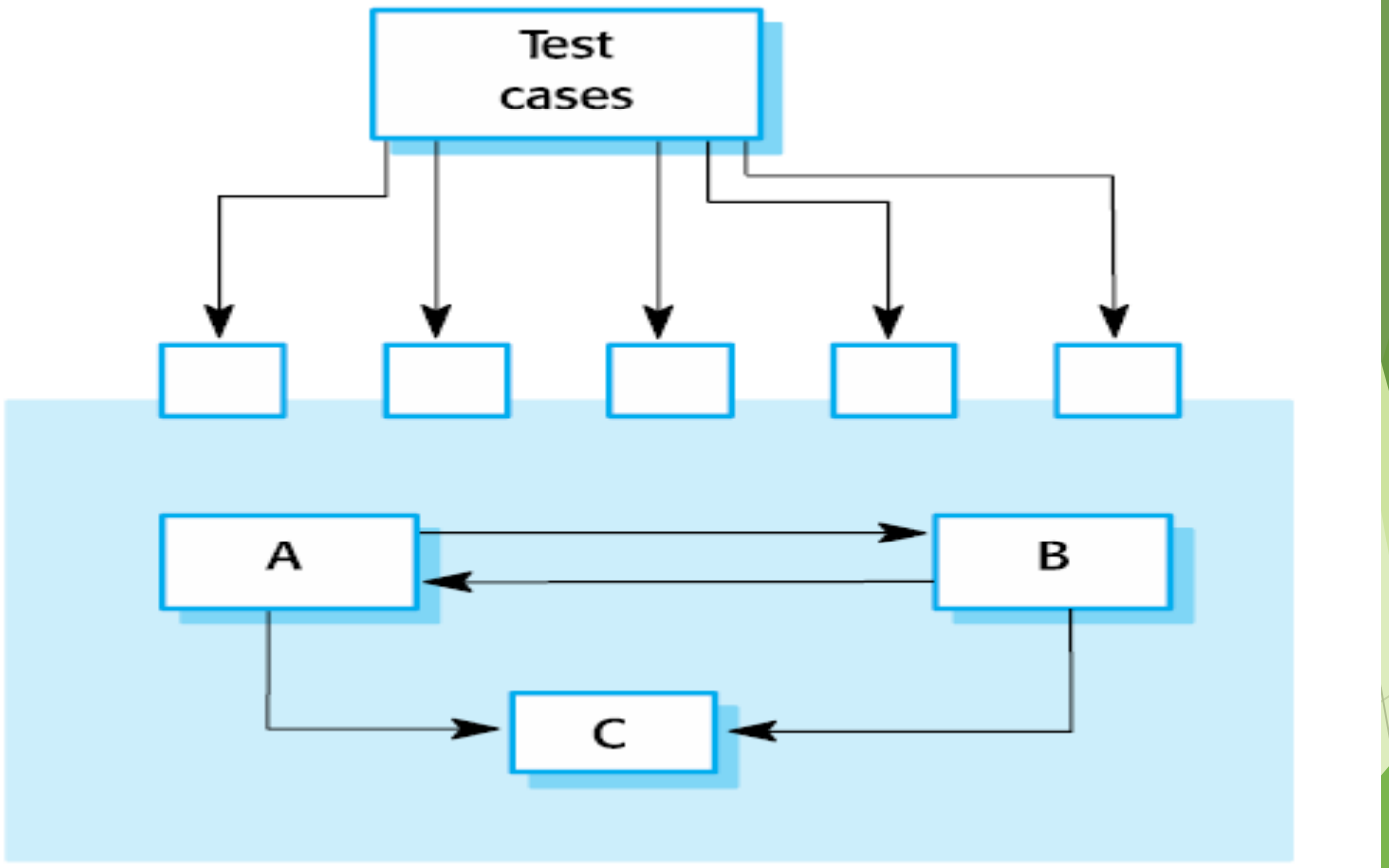
- Bileşen Testi
  - Yazılım bileşenleri genellikle birbiriyle etkileşen birkaç nesneden oluşan bileşik bileşenlerdir.
  - Bu nesnelerin fonksiyonelliğine tanımlanan bileşen arayüzü üzerinden erişilir.
  - Bileşik bileşenlerin test edilmesinde bileşen arayüzlerinin veya bileşenlerin spesifikasyonlarına göre davrandığını göstermeye odaklanılır.

Yazılım testi

- Bileşen Testi
  - Amaç, arayüz hataları veya arayüzler ile ilgili geçersiz varsayımlar nedeniyle oluşan kusurları tespit etmektir.
  - Arayüz tipleri
  - Parametre arayüzleri: veri bir metot veya yordamdan diğerine geçerilir.
  - Paylaşılan bellek arayüzleri: Bellek bloğu prosedürler veya fonksiyonlar arasında paylaşılır.
  - Prosedürel arabirimler: Alt sistem, diğer alt sistemler tarafından çağrılacak bir dizi yordamı kapsar.
  - Mesaj geçiş arayüzleri: Bir bileşenin diğer bir bileşene bir mesaj göndererek hizmet talep ettiği arabirimlerdir.

Yazılım testi

- Bileşen Testi



Yazılım testi

- Bileşen Testi arayüz hataları
  - Arayüz yanlış kullanımı: Bir bileşen başka bir bileşeni çağırır ve arayüzünü kullanırken bir hata yapar.
  - Arayüzün yanlış anlaşılması: Çağıran bir bileşen, çağrılan bileşenin arabiriminin özelliklerini yanlış anlar ve davranışı hakkında varsayımlar yapar.
  - Zamanlama hataları: Çağrılan ve çağıran bileşen farklı hızlarda çalışır ve böylece güncel olmayan bilgilere erişilir.

Yazılım testi

- Bileşen Testi arayüz kılavuz bilgi
- Kodu inceleyerek dışsal bileşenlere olan çağrıları tespit et ve dış bileşene yapılan çağrı için parametre aralığının en uç noktalarında olmasını sağlayacak şekilde bir test kümesi tasarlayın.
- İşaretçi parametrelerini her zaman null işaretçilerle test edin.
- Bileşenin arızalanmasına neden olan testler tasarlayın.
- Mesaj geçirme sistemlerinde stres testini kullanın.
- Paylaşılan bellek sistemlerinde, bileşenlerin etkinleştirilme sırasını değiştirin.

Yazılım testi

- Sistem Testi
- Sistemin bileşen ve nesnelerinin etkileşimini, bütünleştirilmesi ve daha sonra entegre sistemin test edilmesini içerir.
- Bileşenler arasındaki etkileşimleri test etmeye odaklanır.
- Sistem testi, bir sistemin ortaya çıkan davranışını test eder.
- Sistem testinde bileşenlerin uyumluluğu, sorunsuz etkileştiği ve arayüzlerindeki aktarımların doğruluğu kontrol edilir.

Yazılım testi

- Sistem Testi
- Sistem testi ile etkileşim sorunları ortaya çıkarılmaya çalışıldığı için, kullanım durumu bazlı test sistem testi için uygundur.

- Kullanım durumu sıra diyagram olarak modellenmiş ise bu diayagramlardan etkileşen bileşenler ve nesneler görülebilir.

Yazılım testi

- Sistem testi ile bileşen testi farklılıkları;
- Sistem testinde, yeniden kullanılabilir bileşenler ve hazır sistemler yeni geliştirilen bileşenlerle entegre edilebilir.
- Sistem testinde farklı üyeler veya alt ekiplerin geliştirdiği bileşenler bir araya getirilir. Bileşen testine göre sistem testi daha fazla geliştirici ve yazılım geliştirme takımını kapsar.

Yazılım testi

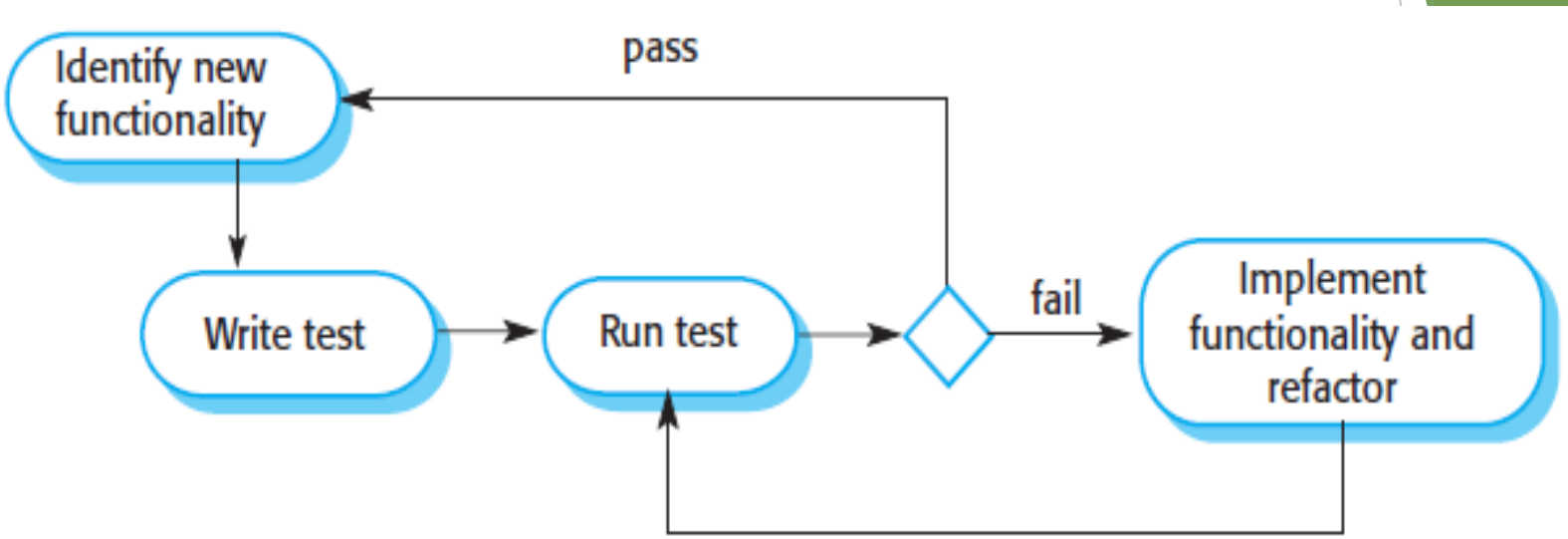
- Sistem testi
- Tüm sistemin olası her program yürütme sırasının test edildiği kapsamlı testler mümkün değildir.

Yazılım testi

- Test güdümlü geliştirme
- Test güdümlü geliştirme (TGG), test etme ve kod geliştirmenin dönüşümlü olduğu bir yaklaşımdır.
- Kod o artış için bir dizi testle birlikte artımsal olarak geliştirilir.
- TGG, Extreme Programming(XP) gibi çevik yöntemlerin bir parçası olarak ortaya çıktı ancak plana dayalı geliştirme süreçlerinde de kullanıyor.

Yazılım testi

- Test güdümlü geliştirme



Yazılım testi

- Test güdümlü geliştirme yararları
- Kod kapsamı
  - Yazılan her kod bölümünün en az bir ilişkilendirilmiş testi olduğundan yazılan tüm kodlarda en az bir test bulunur.
- Gerileme(regresyon) testi
  - Bir program geliştirildikçe bir gerileme test takımı aşamalı olarak geliştirilir.
- Basitleştirilmiş hata ayıklama
  - Bir test başarısız olduğunda, sorunun nerede olduğu açıktır.
- Sistem belgeleri
  - Testler , kodun ne yapması gerektiğini tanımlayan bir belge türüdür.

Yazılım testi

- Dağıtım testi
- Geliştirme ekibinin dışındaki kişiler tarafından bir sistemin bir dağıtımını kullanım ile test etme işlemidir.
- Testin birincil amacı, satıcıları sistemin kullanım için yeterince iyi olduğuna ikna etmektir.
- Dağıtım testi genellikle, testlerin yalnızca sistem spesifikasyonundan türetildiği bir kara kutu testidir.

Yazılım testi

- Dağıtım testi
- Dağıtım testi, bir sistem testi şeklidir.
- Sistem testi ile farklar:
  - Sistem geliştirmeye katılmamış ayrı bir ekip dağıtım testinden sorumlu olmalıdır.
  - Geliştirme ekibi tarafından yapılan sistem testi, sistemdeki kusurları ortaya çıkarmaya yöneliktir. Dağıtım testi sistemin gereksinimlerini karşılayıp karşılamadığını kontrol etmek ve müşteriler için yeterli olduğundan emin olmak için yapılır yani bir doğrulama testidir.

Yazılım testi

- Dağıtım testi – gereksinimlere dayalı test
- Gereksinimlere dayalı test, her gereksinimin incelenmesini ve bunun için bir test veya testler geliştirilmesini kapsar.
- Gereksinimlere dayalı test bir doğrulama testidir.

Yazılım testi

- Dağıtım testi – senaryo testi
- Senaryo testi, tipik senaryolar tasarladığınız ve bu senaryoları sisteme göre test takımları geliştirmek için kullandığınız bir dağıtım testi yaklaşımıdır.
- Bir senaryo, sistemin kullanılabileceği bir yolu tanımlayan bir hikayedir.
- Gereksinim mühendisliği sürecinin bir parçası olarak senaryolar veya kullanıcı hikayeleri kullandıysanız, bunları test senaryoları olarak yeniden kullanabilirsiniz.

Yazılım testi

- Dağıtım testi – performans testi
- Dağıtım testi, bir sistemin performans ve güvenilirlik gibi özelliklerinin test edilmesini içerebilir.
- Testler, sistemin kullanım profilini yansıtmalıdır.
- Performans testleri genellikle, sistem performansı kabul edilemez noktaya gelinceye kadar yükün sürekli artırıldığı bir dizi testin planlanmasını içerir.

Yazılım testi

- Dağıtım testi – performans testi
- Stres testi, sistemin yetmezlik davranışını test etmek için kasıtlı olarak aşırı yüklendiği bir performans testi şeklidir.
- Stres testinin yararları:
  - Sistemin başarısızlık hali davranışı belirlenir.
  - Sistem tamamen yüklendiğinde oluşan hatalar ortaya çıkar.

Yazılım testi

- Kullanıcı testi
- Kullanıcı testi ile test sürecinde kullanıcılar sistem testi hakkında girdi ve tavsiyelerde bulunur.
- Kapsamlı sistem ve dağıtım testi yapıldığında bile kullanıcı testi şarttır.

Yazılım testi

- Kullanıcı testi
- Kullanıcı testinin 3 farlı şekli vardır, bunlar:
  - Alfa testi
  - Beta testi
  - Kabul testi

Yazılım testi

- Kullanıcı testi
  - Alfa testi: Yazılım kullanıcıları, geliştirme ortamında geliştirme ekibiyle birlikte yazılımı test etmek için çalışır.

Yazılım testi

- Kullanıcı testi
  - Beta testi: Yazılımın bir dağıtımı, daha geniş bir kullanıcı gurubuna, deneme yapmaları ve keşfettikleri sorunları sistem geliştiriciler ile paylaşmaları için sunulur.

Yazılım testi

- Kullanıcı testi
  - Kabul testi: Çoğunlukla özel sistemler için müşteriler, yazılım sisteminin geliştiricilerden teslim alınmaya ve dağıtıma hazır olup olmadığına karar vermek için bir test yapar.

Yazılım testi

- Kullanıcı testi
  - Kabul testi 6 aşaması olan bir süreçtir. Bunlar:
    - Kabul kriterlerini tanımla,
    - Kabul testini planlama,
    - Kabul testlerini türet,
    - Kabul testlerini çalıştır,
    - Test sonuçlarını tartış,
    - Sistemi reddet yada kabul et.

Yazılım Mühendisliği

Kaynakça

- *Ian Sommerville, Çeviri editörü: Prof. Dr. N. Yasemin Topaloğlu, Yazılım Mühendisliği, Nobel akademik yayıncılık , 2018, 10. basımdan çeviri. (Ders kitabı)*

- https://iansommerville.com/software-engineering-book/slides/

Yazılım Mühendisliği
Yazılım Evrimi

Bölüm 9: Yazılım Evrimi

Genel Bakış

- Yazılım evrimi
  - Yazılım evrimi
  - Kalıt sistemler

Yazılım Evrimi

- Yazılımlarda değişiklik önemli ve gereklidir.
  - Yazılım kullanıldıkça yeni gereksinimler ortaya çıkar,
  - Hatalar onarılmalıdır,
  - İş ortamı değişir,
  - Sisteme yeni donanımlar eklenir,
  - Sistem için güvenilirlik ve performans beklentileri olur.

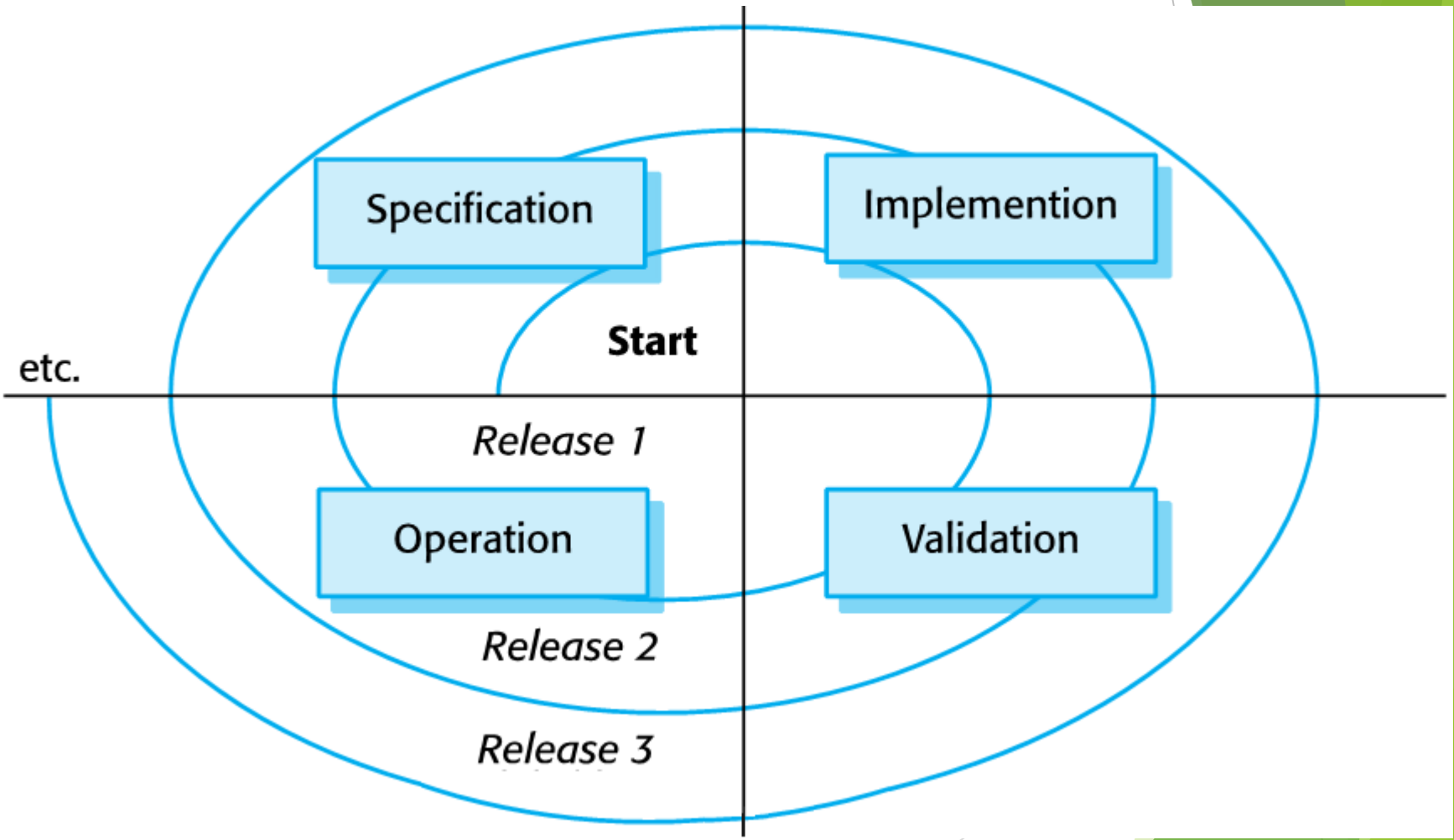
- Yazılım kullanan kurumların sistemlerinde değişikliği yapmak ve yönetmek önemlidir.

Yazılım Evrimi

- Kuruluşlar, kendileri için kritik ticari bir varlık olan sistemlerine büyük yatırım yaparlar.
- Bu varlıkların değerini korumak için, değiştirilmesi ve güncellenmesi gerekir.
- Büyük şirketler yazılım bütçesinin büyük bir kısmını, yeni yazılım geliştirmeye değil mevcut yazılımı değiştirme ve evrimleştirmeye harcarlar.

Yazılım Evrimi

Yazılım geliştirme ve evrim için spiral model

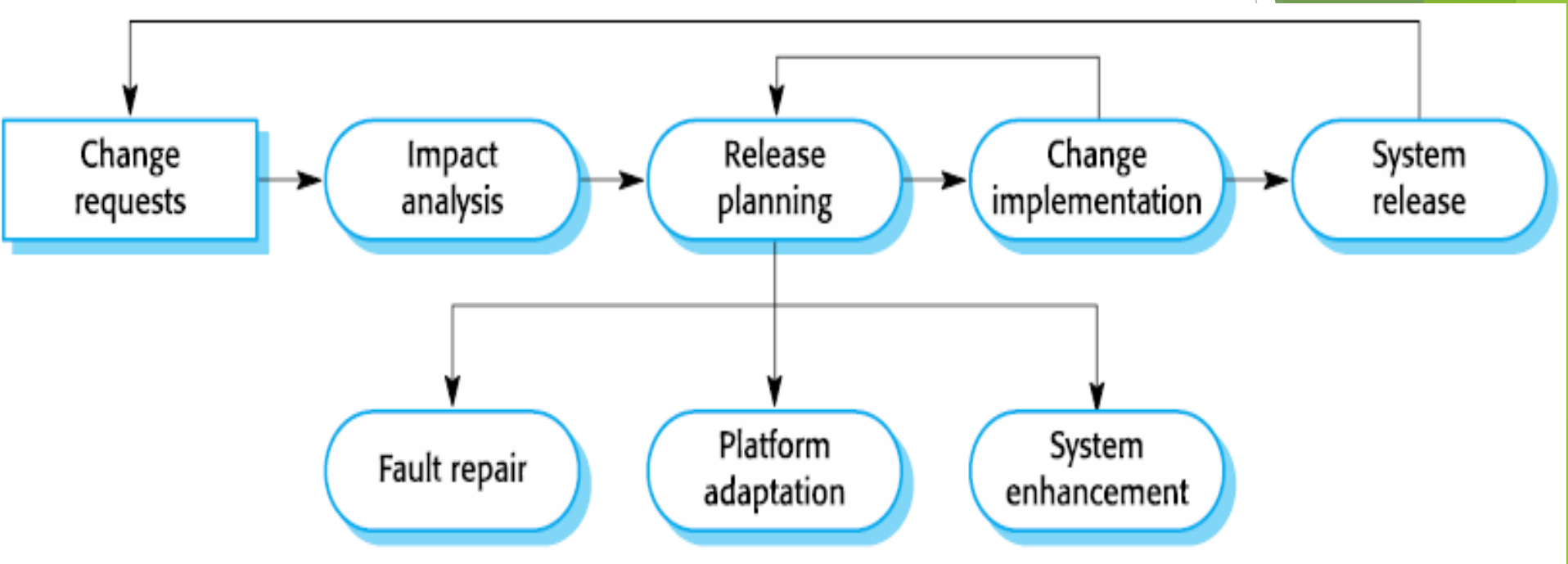


Yazılım Evrimi

- Yazılım evrimi:
- Yazılım evrim süreçleri
  - Yazılımın türü,
  - Kullanılan geliştirme süreçleri,
  - Süreçte görev alan kişilerin beceri ve deneyimleri.
- Değişim önerileri sistem gelişiminin itici gücüdür.
  - Değişiklikten etkilenen bileşenlerle ilişkilendirilmeli, böylece değişikliğin maliyeti ve etkisi tahmin edilmelidir.
- Yazılım değişimi ve evrimi sistem ömrü boyunca devam eder.

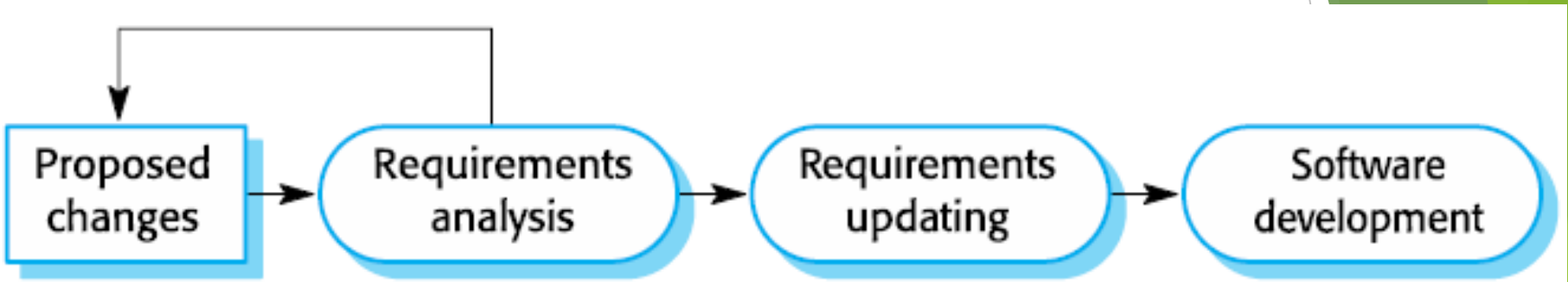
Yazılım Evrimi

Yazılım evrim süreci için genel bir model



Yazılım Evrimi

Değişiklik gerçekleştirimi



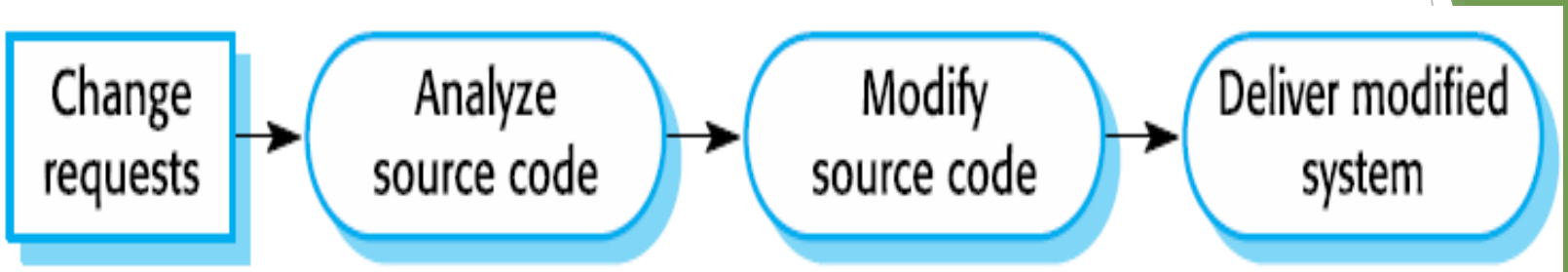
Yazılım Evrimi

Bazen yazılım değişikliklerinin acil olarak ele alınmasını gerektirebilir bunun sebepleri:

- Normal çalışmaya dönmek için ciddi bir sistem arızasının onarılması gerekebilir,
- Sistem ortamındaki değişikliklerden ortaya çıkmış, beklenmedik durumların ortadan kaldırılması gerekebilir,
- İş ortamında meydana gelen iş yapılış değişiklikleri olduğu durumda acilen yazılımın yeni düzenlemeye uyarlanması gerekebilir.

Yazılım Evrimi

Acil onarım süreci



Yazılım Evrimi

Çevik yöntemlerde yazılım evrimi

- Çevik yöntemler artımlı yazılım geliştirmeye dayandığı için geliştirme sürecinden, evrim sürecine geçişin sorunsuz olduğu yöntemlerdir,
  - Bu anlamda evrim, sık sistem dağıtımlarına dayanan geliştirme sürecinin devamıdır,
- Değişiklerin sonrasında yapılan otomatik gerileme testleri özellikle faydalıdır.
- Değişiklikler ek kullanıcı hikayeleri olarak ifade edilebilir.

Yazılım Evrimi

Geliştirme takımı ve evrim takımı farklı ise ortaya çıkabilecek sorunlar:

- Geliştirme takımının çevik bir yaklaşım kullandığı, ancak evrim takımın çevik yöntemlere aşina olmadığı ve plan güdümlü bir yaklaşımı tercih ettiği durum,
  - Evrim takımı, evrimi sürdürmek için ayrıntılı belgeler bekler ancak bu belgeler çevik süreçlerde üretilmez.
- Geliştirme için plan güdümlü bir yaklaşımın kullanıldığı, ancak evrim takımının çevik yöntemleri kullanmayı tercih ettiği durum,
  - Evrim takımının baştan otomatik testleri hazırlaması gerekebilir ve sistemdeki kod, çevik geliştirme takımının beklediği özellikte olmayabilir.

Kalıt Sistemler

Kalıt sistemler

Kalıt sistem nedir?

- Kuruluşlar uzun yıllar önce yazılım sistemlerine sahip olmuş ve dönem içinde de birkaç kez bu sistemlerini yenilemiştir, ancak halen bu sistemleri kullanan kuruluşlar vardır. İşte eskiden gelen mevcut bu yazılım sistemlerine kalıt sistem adı verilir.

Kalıt Sistemler

Kalıt sistemler

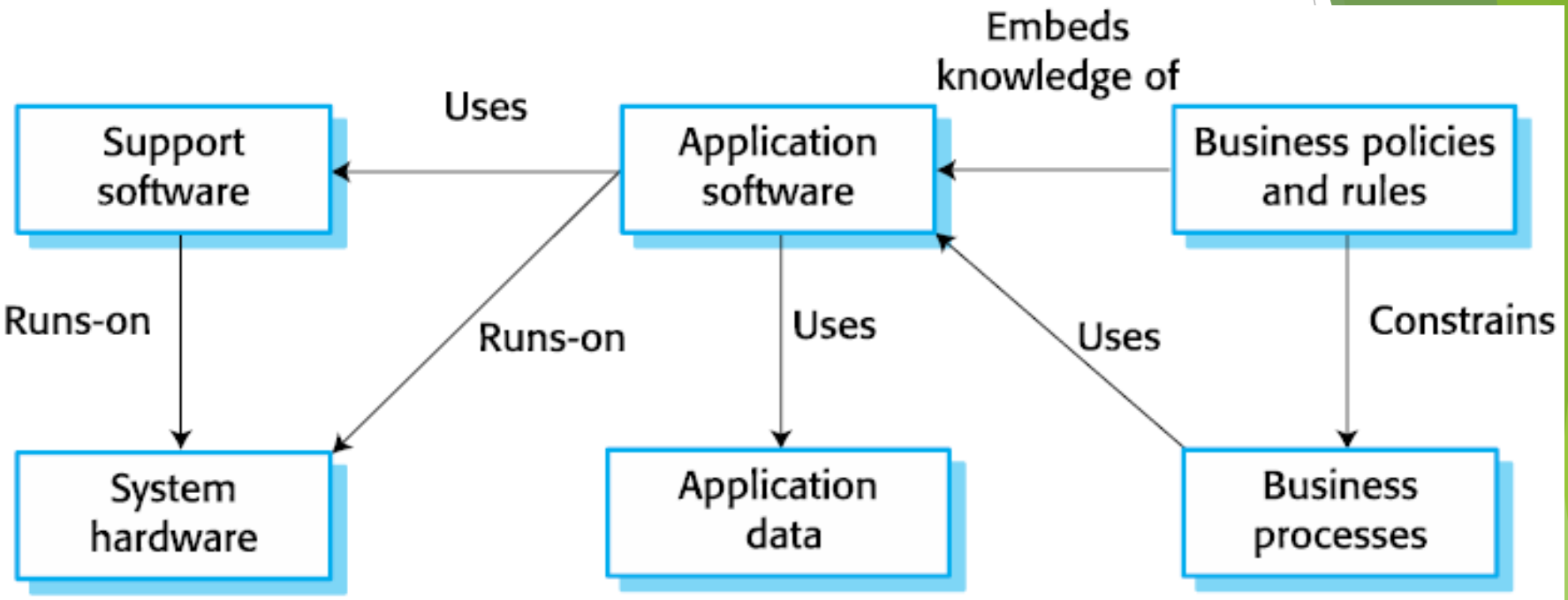
- Kalıt sistemler, artık yeni sistem geliştirme için kullanılmayan programlama dillerine ve teknolojiye dayanan nispeten eski sistemlerdir.

- Kalıt yazılım, ana bilgisayar gibi daha eski donanıma bağlı olabilir ve süreç ve yordamlara sahip olabilir.

- Kalıt sistemler sadece yazılım sistemleri değil, donanım, yazılım, kütüphaneler ve diğer destekleyici yazılım ve iş süreçlerini içeren daha geniş sosyo-teknik sistemlerdir.

Kalıt Sistemler

Bir kalıt sistemin öğeleri



Kalıt Sistemler

Kalıt sistem bileşenleri

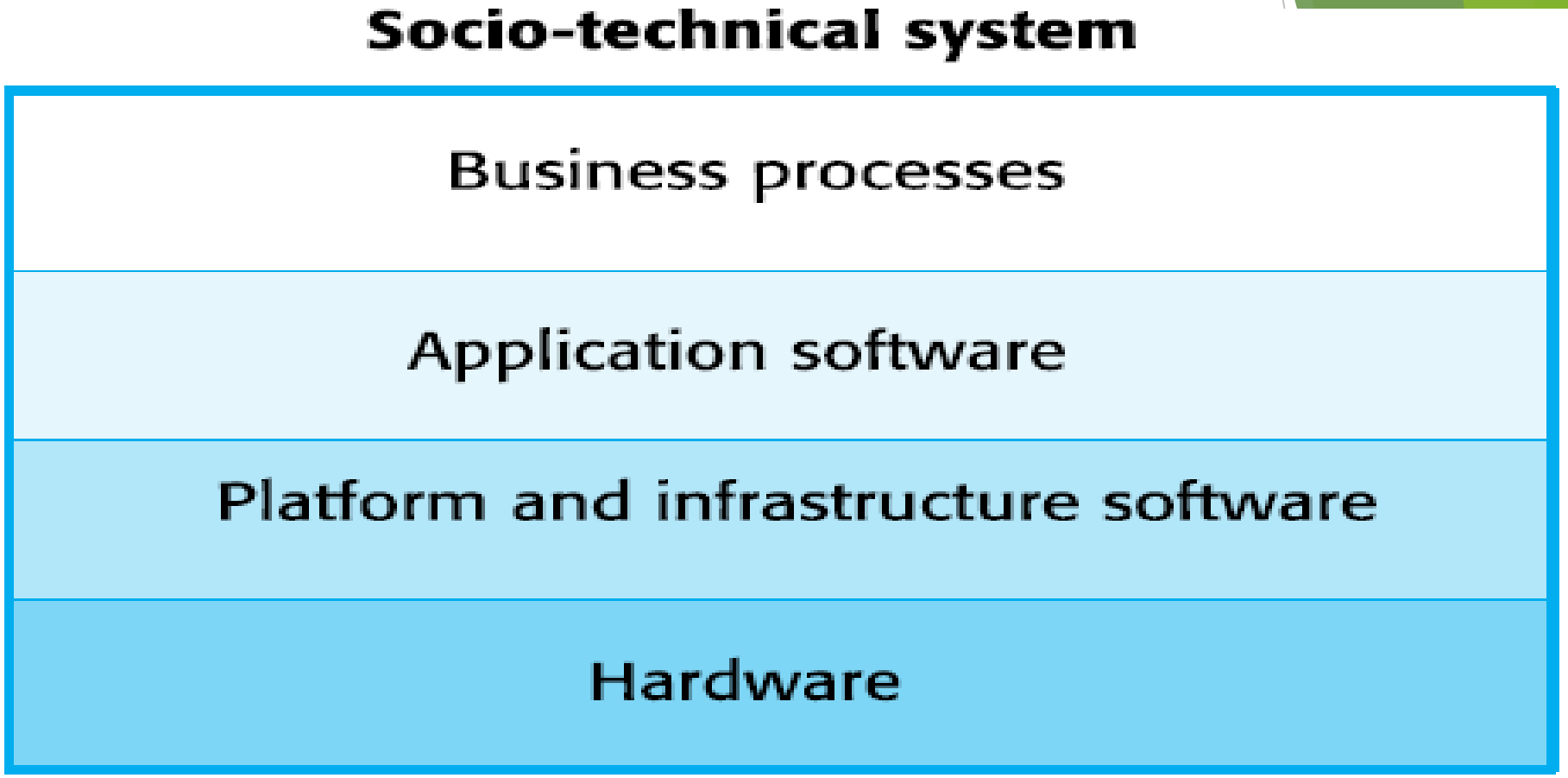
- Sistem donanımı, kalıt sistemler artık mevcut olmayan donanım için yazılmış olabilir.
- Destek yazılımı, kalıt sistem kullanılmayan veya desteklenmeyen çeşitli destek yazılımlarına bağlı olabilir.
- Uygulama yazılımı, kalıt sistemlerin uygulama yazılım sistemi genellikle bir dizi uygulama programından oluşur.
- Uygulama verileri, bunlar uygulama sistemi tarafından işlenen verilerdir.

Kalıt Sistemler

Kalıt sistem bileşenleri

- İş süreçleri, bunlar, bazı iş hedeflerine ulaşmak için işte kullanılan süreçlerdir.
- İşletme politikaları ve kuralları, bunlar, işletmenin nasıl yürütülmesi gerektiğinin tanımlar ve işletme üzerindeki kısıtlamalardır.

Kalıt Sistemler



Kalıt Sistemler

Kalıt sistem değişikliği risk nedenleri;

Eski sistemin değiştirilmesi riskli ve pahalıdır, bu sebeple işletmeler bu sistemleri değiştirmeye sıcak bakmaz ve kullanır.

Sistem değişikliği risk nedenleri:

- Sistem spesifikasyonun tam olarak bulunmaması,
- Sistem ve iş süreçlerinin iç içe geçmiş olması,
- Sisteme gömülmüş ve belgelendirilmemiş iş kuralları olması,
- Yeni yazılım geliştirme sürecinde süre ve bütçe aşılabilme riski olması.

Kalıt Sistemler

Kalıt sistem üzerinde değişiklik maliyet nedenleri;

Kalıt sistemler üzerinde çalışıp değişiklik ve düzeltme yapmak maliyetlidir.

Bunun nedenleri:

- Tutarlı programlama stili yoktur,
- Eski programlama dillerinin kullanılması ve bu dili bilen çalışan yokluğu,
- Yetersiz yada hiç olmayan yazılım dokümantasyonu,
- Yazılımın yapısal bozulmuşluğu,
- Program optimizasyonlarının yeni nesil çalışanlar tarafından anlaşılma zorluğu,
- Tutarsız, tekrarlı ve güncel olmayan veriler, uyumsuz dosya yapıları ve çeşitli veri tabanları bulunması.

Kalıt Sistemler

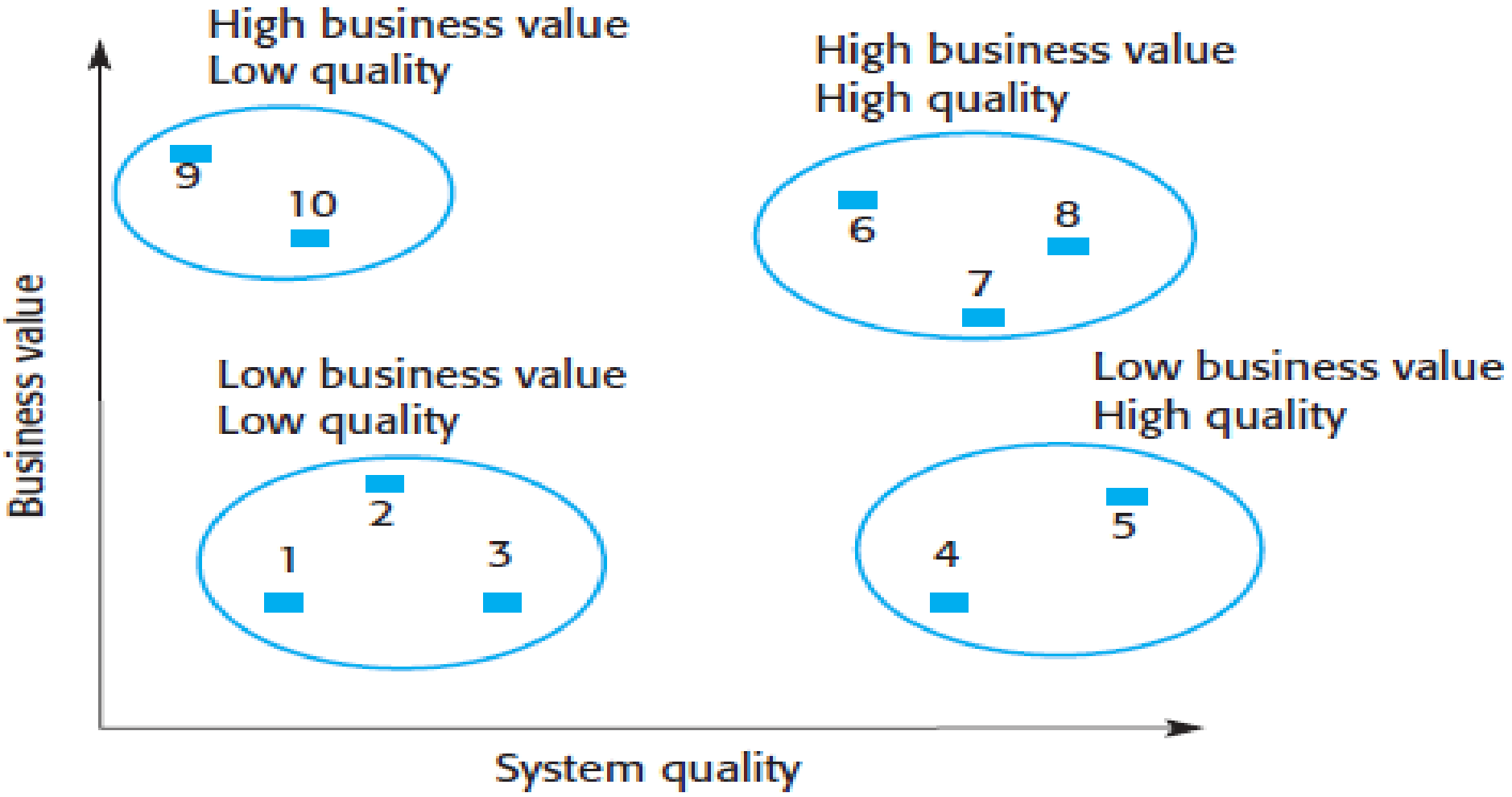
Kalıt sistem evrimi için stratejiler;

Kalıt sistemlere bel bağlayan kuruluşlar, bu sistemlerin evrimi için bir strateji seçmelidir. Bu stratejiler:

- Sistemi tamamen kullanımdan kaldırıp ve artık gerekli olmayacak iş süreçlerini değiştirmek,
- Sistemi değiştirmeden, düzenli bakım ile devam etmek,
- Sistemin bakım yapılabilirliğini kolaylaştırmak için yeniden yapılandırmak,
- Sistemi yeni bir sistemle değiştirmek.

- Seçilen strateji sistem kalitesine ve iş değerine(iş hayatına katkı değeri) bağlı olmalıdır.

Kalıt Sistemler



Kalıt Sistemler

Kalıt sistem değerlendirme kategorileri;

- Düşük kalite, düşük iş değeri,
  - Bu sistemler kullanımdan kaldırılmalıdır.
- Düşük kalite, yüksek iş değeri,
  - Önemli ticari katkı sağlar, ancak bakımı pahalıdır. Yeniden mühendislik yapılmalı yada uygun bir sistem varsa değiştirilmelidir.
- Yüksek kalite, düşük işletme değeri,
  - Hazır bir sistem ile değiştirilmeli, kullanımdan kaldırılmalı veya bakım ile devam edilmeli.
- Yüksek kalite, yüksek iş değeri,
  - Normal sistem bakımı kullanarak çalışmaya devam edilmeli.

Kalıt Sistemler

Kalıt sistem değerlendirmede paydaşlar;

- Değerlendirme farklı bakış açılarını dikkate almalıdır;
  - Sistem son kullanıcıları,
  - Ticari Müşteriler,
  - Bölüm yöneticileri,
  - BT yöneticileri,
  - Üst düzey yöneticiler.

Kalıt Sistemler

İş değeri belirlemede dikkate alınan temel konular;

- Sistemin kullanımı,
  - Sistemler kullanım nedeni önemsiz olarak ara sıra veya az sayıda kullanıcı tarafından kullanılıyor ise düşük iş değerine sahip olabilirler.
- Desteklenen iş süreçleri,
  - Bir sistem verimsiz iş süreçlerinin kullanımını zorlarsa, düşük bir iş değerine sahip olabilir.
- Sistem güvenilirliği,
  - Bir sistem güvenilebilir değilse ve sorunlardan müşteriler olumsuz etkileniyorsa, sistemin iş değeri düşüktür.
- Sistem çıktıları,
  - İşletme, sistem çıktılarına bağlıysa, sistemin iş değeri yüksektir.

Kalıt Sistemler

Kalıt sistem değerlendirilmesi

- İş süreci açısından değerlendirme,
  - İş süreci, işletmenin mevcut hedeflerini ne kadar iyi destekliyor?
- Teknik açıdan değerlendirme,
  - Ortam değerlendirmesi,
    - Sistem ortamı ne kadar etkilidir ve bakımı ne kadar maliyetlidir?

Kalıt Sistemler

Kalıt sistem değerlendirilmesi

- Teknik açıdan değerlendirme,
  - Uygulamanın değerlendirilmesi,
    - Uygulama yazılım sisteminin kalitesi nedir?
    - Anlaşılabilirlik, belgeleme, performans, programlama dili vb bakımından değerlendirme.

Yazılım Mühendisliği

Kaynakça

- *Ian Sommerville, Çeviri editörü: Prof. Dr. N. Yasemin Topaloğlu, Yazılım Mühendisliği, Nobel akademik yayıncılık , 2018, 10. basımdan çeviri. (Ders kitabı)*

- https://iansommerville.com/software-engineering-book/slides/

Yazılım Mühendisliği
Yazılım Bakımı

Bölüm 9: Yazılım Bakımı

Genel Bakış

Yazılım bakımı

- Bakım tahminleme
- Yeniden yazılım mühendisliği
- Yeniden üretim

Yazılım Bakımı

- Bir programı kullanıma sunduktan sonra değiştirme sürecidir.
- Müşteriye özel yazılımları değiştirmek, iyileştirmek için yapılan çalışmalar için yazılım bakımı terimi kullanılır.
- Genel yazılım ürünlerinin yeni dağıtımlarını oluşturmak için ise evrim terimi kullanılır.
- Bakım kapsamı içinde sistemin mimarisinde yapılan büyük değişiklikler yoktur.
- Değişiklikler mevcut yazılım değiştirilerek uygulanır.
- Değişiklikler sistemde mevcut bileşenleri değiştirerek ve yeni bileşenler ekleyerek gerçekleştirilir.

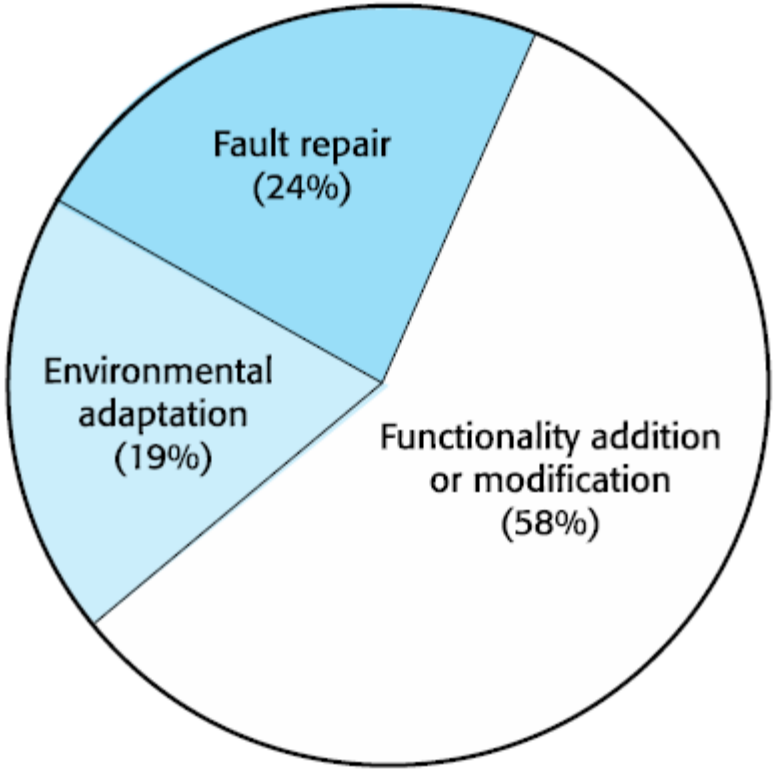
Yazılım Bakımı

Yazılım bakım türleri

- Hata onarımları,
  - Hataları ve güvenlik açıklarını gidermek için sistemi değiştirme,
  - Gereksinimlerini karşılayacak şekilde var olan eksiklikleri gidermek için sistemi değiştirme.
- Ortama uyarlama(Ortama dair uyarlama),
  - Yazılımı farklı bir işletim ortamına uyarlamak için bakım,
  - Bir sistemi farklı bir ortamda çalışacak şekilde değiştirme(donanım, işletim sistemi vb.).
- İşlevsellik ekleme,
  - Yeni gereksinimleri karşılamak için sistemi değiştirme.

Yazılım Bakımı

Bakım maliyetinin bakım türüne göre dağılım grafiği



Yazılım Bakımı

Yazılım bakım maliyetleri:

- Bakım maliyeti genellikle geliştirme maliyetinden daha fazladır (uygulamaya bağlı olarak 2 ila 100 katı).
- Hem teknik hem de teknik olmayan faktörlerden etkilenir.
  - Teknik faktör: programlama dili, stili, dokümantasyon vd.
  - Teknik olmayan faktör: donanımın sorunsuz çalışmaması, dış sisteme olan bağlantı vd.
- Bakım yapıldıkça maliyet artar.
  - Yapısal bozulmalar
- Kullanımda kalma süresi artmış(eskimiş) yazılımın bakımı maliyeti artar.
  - işletim sistemi, programlama dili, derleyiciler için destek bulmak vd.

Yazılım Bakımı

Yazılım bakım maliyetleri

- Yazılıma geliştirme süreci sonrasında yeni özellikler eklenmesi eklemek daha maliyetlidir. Bunun nedenleri;
  - Yeni takımın bakım yapabilmek için programları anlama gereği.
  - Bakım ve geliştirme takımlarının ayrı olması, geliştirme takımının, kolay bakım yapılabilir bir yazılım geliştirme gayretlerini azaltabilir.
  - Program bakım işi popüler değildir. Bu sürece deneyimsiz çalışanlar atanır.
  - Programların kullanım süresi uzadıkça, sürekli yapılan değişiklikler nedeniyle yapıları bozulur ve değişiklik yapılması zorlaşır.

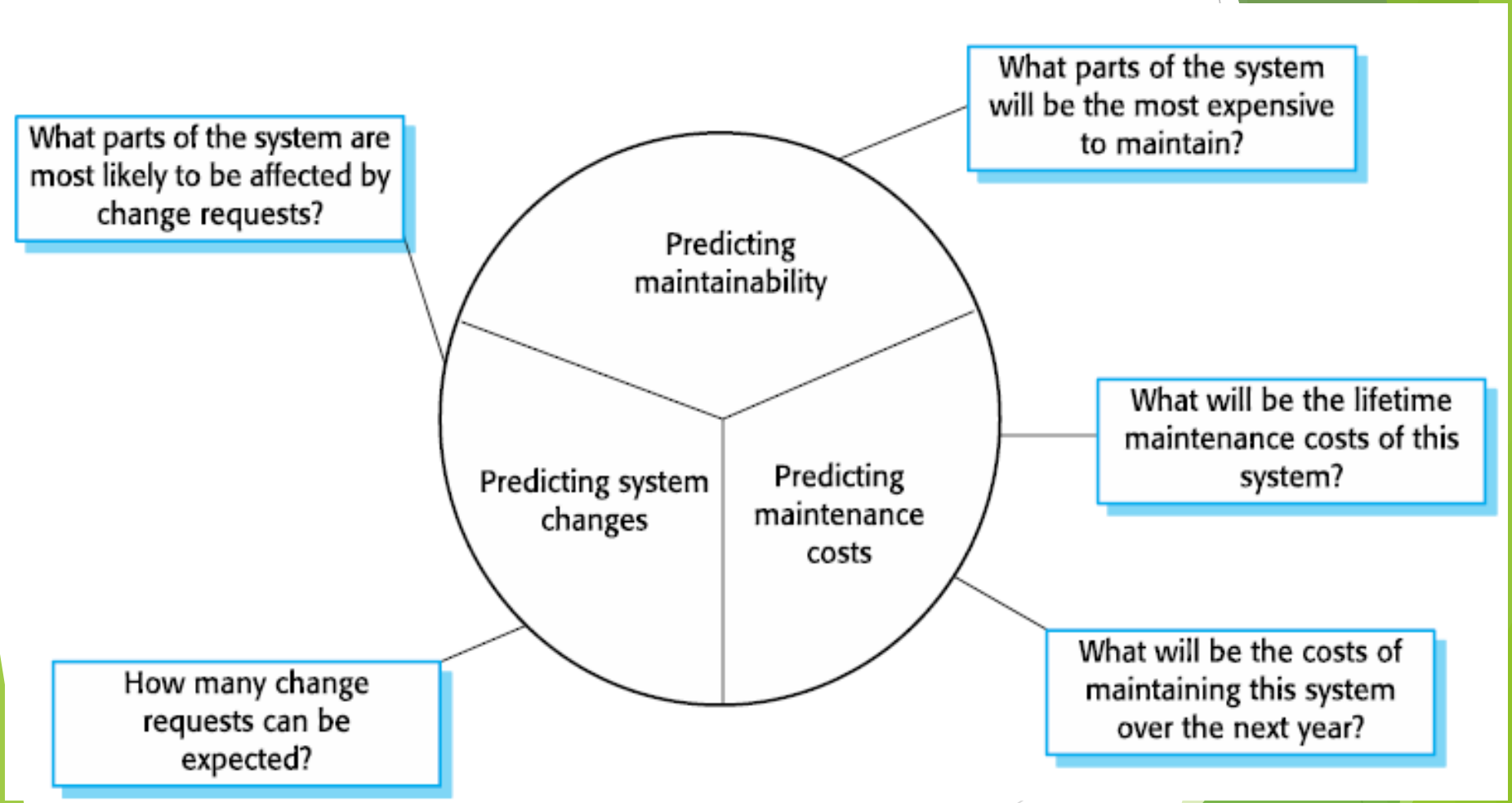
Yazılım Bakımı

Yazılım bakım tahminleme

- Bakım tahmini, sistemin hangi bölümlerinin sorunlara neden olabileceğini ve yüksek bakım maliyetlerine sahip olduğunu değerlendirmeyle ilgilidir. Bu yapılırsa;
  - Değişme olasılığı olan bileşenler daha uyarlanabilir tasarlanabilir,
  - Bu bileşenlerin bakım sürecinde bakım maliyetlerini düşürmek için bu bileşenler iyileştirilebilir,
  - Bir zaman aralığında bakım maliyetleri değerlendirilerek bakım bütçesi oluşturulabilir.

Yazılım Bakımı

Yazılım bakım tahminleme



Yazılım Bakımı

Değişiklik tahminleme

- Bir sistemdeki değişiklik sayısını tahmin etmek için bir sistem ve çevresi arasındaki ilişkilerin anlaşılmasını gerektirir. Bu ilişkiyi etkileyen faktörler;
  - Sistem arayüzleri sayısı ve karmaşıklığı,
  - Geçici sistem gereksinimlerinin sayısı,
  - Sistemin kullanıldığı iş süreçleri değişim durumu.

Yazılım Bakımı

Yazılım bakım tahminleme

- Program karmaşıklığı ile bakım kolaylığı ilişkilidir.
  - Karmaşık programların bakımı daha maliyetlidir.

- Karmaşık bileşenler basitleriyle değiştirilmelidir.

Yazılım Bakımı

Bakım tahminleme

- Bakım kolaylığını tahminlemek için kullanımda olan sistemin bazı verilerine bakılabilir, bunlar ölçüt olarak kullanılabilir, Bunlar:
  - Düzeltici bakım isteği sayısı,
  - Etki analizi için gerekli ortalama süre,
  - Bir değişiklik talebini gerçekleştirmek için harcanan ortalama süre,
  - Dikkat çeken değişiklik isteklerinin sayısı.
- Bunlardan herhangi biri veya tümü artıyorsa, bu bakım kolaylığında azalma olarak değerlendirilir.

Yazılım Bakımı

Yeniden yazılım mühendisliği

- Yeniden yazılım mühendisliği: Bir kalıt sistemin gerektiği kadarının, işlevselliğini değiştirmeden yeniden yapılandırma veya modern bir yapıya dönüştürülmesi çalışmasıdır.
- Bakım kolaylığını kaybetmiş sistem veya alt sistemlere uygulanan bir çalışmadır.
- Sistem yeniden yapılandırılabilir veya yeniden belgelendirilebilir veya yeniden programlanabilir.
- Mevcut sistemi nispeten az maliyetle hayata döndürme çalışmasıdır.

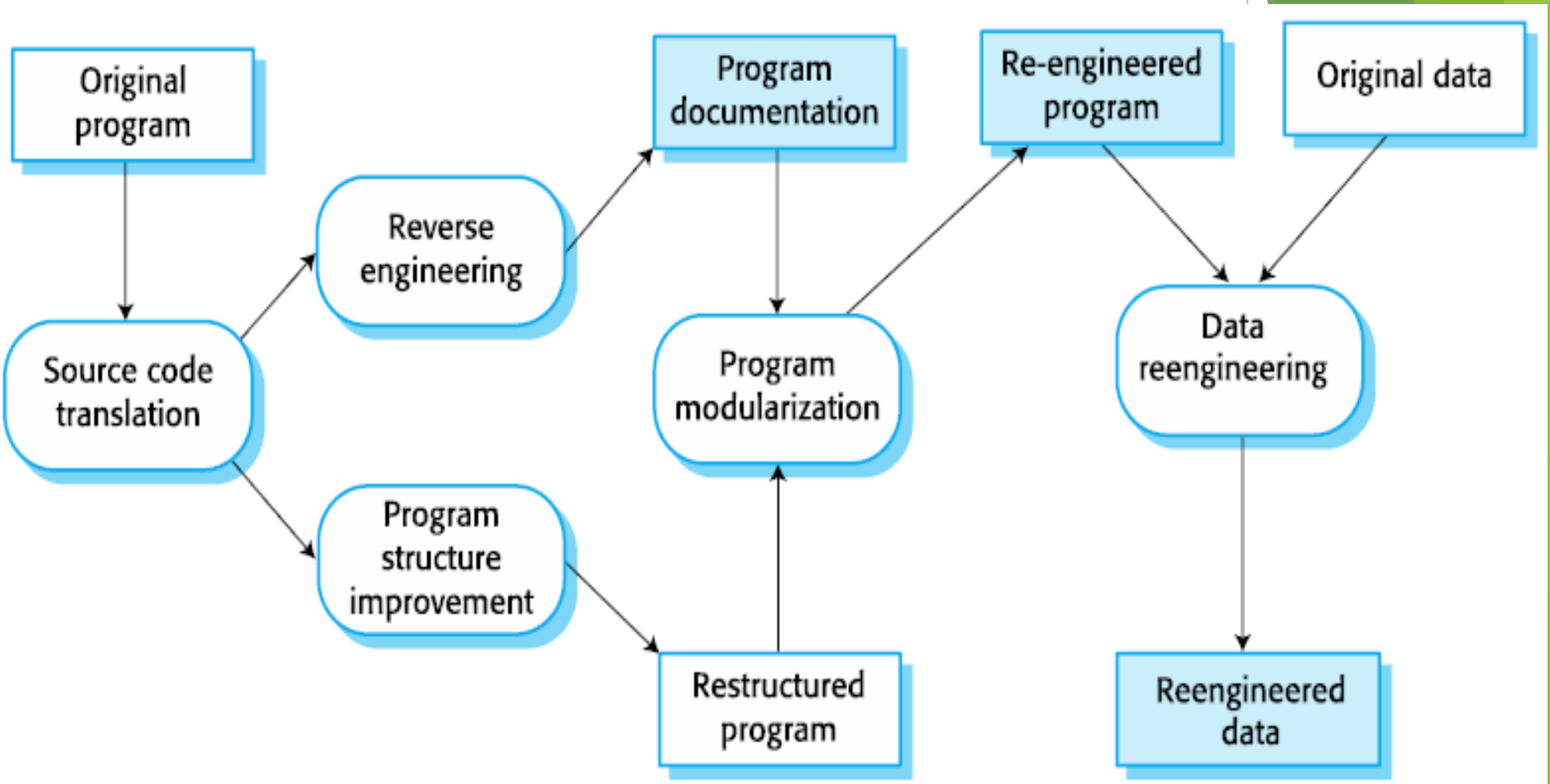
Yazılım Bakımı

Yeniden yazılım mühendisliği avantajları

- Riski azaltmak
  - Yeni yazılım geliştirmenin yeniden yazılım mühendisliğine göre riski yüksektir. Yazılım spesifikasyonunda hata yapma riski, bütçe ve proje süresinin aşılması riskleri gibi.
- Maliyeti azaltmak
  - Genellikle, yeni yazılım geliştirme maliyeti, yeniden mühendislik maliyetinden büyük oranda daha yüksektir.

Yazılım Bakımı

Yeniden yazılım mühendisliği sürecinin genel bir modeli



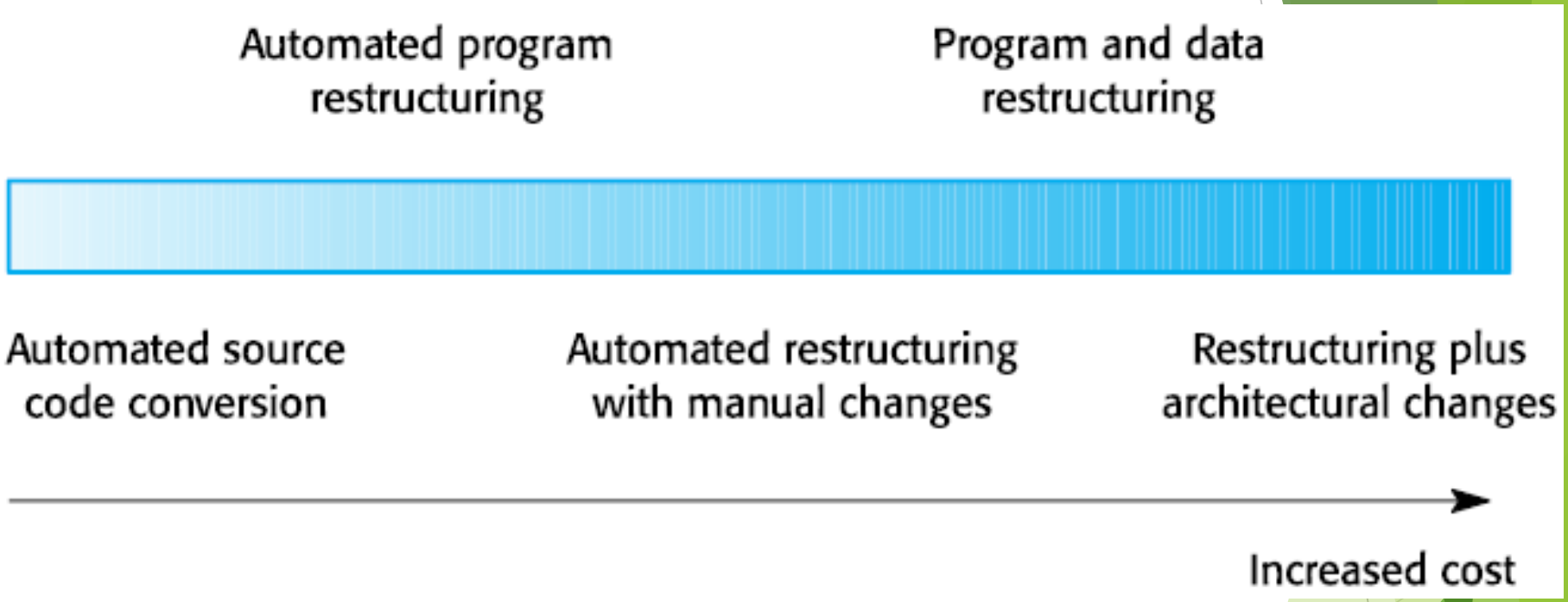
Yazılım Bakımı

Yeniden yazılım mühendisliği süreci etkinlikleri

- Kaynak kod dönüştürme
  - Kaynak kod yeni bir dile yada aynı dilin yeni sürümüne dönüştürülür,
- Tersine mühendislik
  - Program anlamak için incelenir bilgi çıkarılır ve belgeleme yapılır,
- Program yapısının iyileştirilmesi
  - için otomatik olarak yeniden yapılandırma,
- Program modülerleştirme
  - Programın ilişkili bölümleri bir araya toplanır ve sadeleştirilir,
- Veri mühendisliği
  - Sistem verilerini temizleme ve yeniden yapılandırma.

Yazılım Bakımı

Yeniden yazılım mühendisliği süreci maliyeti



Yazılım Bakımı

Yeniden üretim

- Yeniden üretme, değiştirmeler ile oluşan bakım kolaylığı azalmasının nispeten önüne geçmek için bir programda iyileştirmeler yapma sürecidir.
- Yeniden üretme, sonraki değiştirmelerdeki sorunları azaltan 'önleyici bakım' olarak değerlendirilebilir.
- Yeniden üretim, programın yapısını iyileştirmek, karmaşıklığını azaltmak veya anlaşılmasını kolaylaştırmak için değiştirilmesi için yapılır.
- Yeniden üretim ile işlevsellik eklenmemeli, program iyileştirilmeye yoğunlaşılmadır.

Yazılım Bakımı

Yeniden mühendislik ve yeniden üretim

- Yeniden mühendislik, sistem bakımı ile bakım maliyetleri arttıktan sonra gerçekleştirilir.
- Yeniden üretim, geliştirme ve evrim süreci boyunca sürekli bir iyileştirme sürecidir.
- Yeniden üretim bakım maliyetlerini arttıran ve zorlaştıran program yapısı ve kod bozulmasından kaçınılması amacıyla yapılır.

Yazılım Bakımı

Yeniden üretim ile yapılabilecek iyileştirmeler

- Yinelenen kod
  - Aynı veya çok benzer veri kümeleri tek bir yöntem veya işlev olarak tanımlanıp ve çağrılabilir.
- Uzun yöntemler
  - Bir yöntem çok uzunsa, bir kısa yöntem haline getirilmelidir.
- Switch-case deyimleri
  - Farklı tip verilere bağlı olarak programın bir çok yerinde switch-case bulunabilir. Bunlar için çok biçimli yapı kullanılarak iyileştirme sağlanabilir.

Yazılım Bakımı

Yeniden üretim ile yapılabilecek iyileştirmeler

- Veri kümeleme
  - Aynı veri öğesi grubu bir programın çeşitli yerlerinde oluştuğunda ortaya çıkar. Bunlar genellikle tüm verileri kapsayan bir fonksiyon yada nesne ile değiştirilebilir.
- Spekülatif genellik
  - Bu geliştiriciler gelecekte gerekli olabilir diye programa genelliği dahil ettiklerinde olur. Sadece nümerik veri sıralamaya ihtiyaç varken ileride gerekir diye her tipte veriyi sıralayacak fonksiyon tasarlayıp kodlanmışsa, bunlar kaldırılarak iyileşme sağlanabilir.

Yazılım Mühendisliği

Kaynakça

- *Ian Sommerville, Çeviri editörü: Prof. Dr. N. Yasemin Topaloğlu, Yazılım Mühendisliği, Nobel akademik yayıncılık , 2018, 10. basımdan çeviri. (Ders kitabı)*

- https://iansommerville.com/software-engineering-book/slides/

Yazılım Mühendisliği
Yazılım Kalite Yönetimi

Bölüm 24: Yazılım Kalite Yönetimi

Genel Bakış

Yazılım kalite yönetimi

- Yazılım kalitesi
- Yazılım standartları

Yazılım Kalite Yönetimi

- Yazılım kalite yönetimi; yazılım ürünü için yüksek kalite seviyesi elde edilmesini sağlamakla ilgilenir. Bunu sağlamak için hem kurumsal hem de proje düzeyi yönünden çalışılır,
- Kurumsal düzeyde kalite yönetimi, yüksek kaliteli yazılımlar geliştirmek için kurumsal süreçler ve standartlar çerçevesi geliştirmekle ilgilenir.
- Proje düzeyinde kalite yönetimi, tanımlı kalite süreçlerinin uygulanmasını ve bu planlanan süreçlere uyulduğunu kontrol etmeyi içerir.
- Proje düzeyi kalite yönetiminde proje için bir kalite planı oluşturulur.

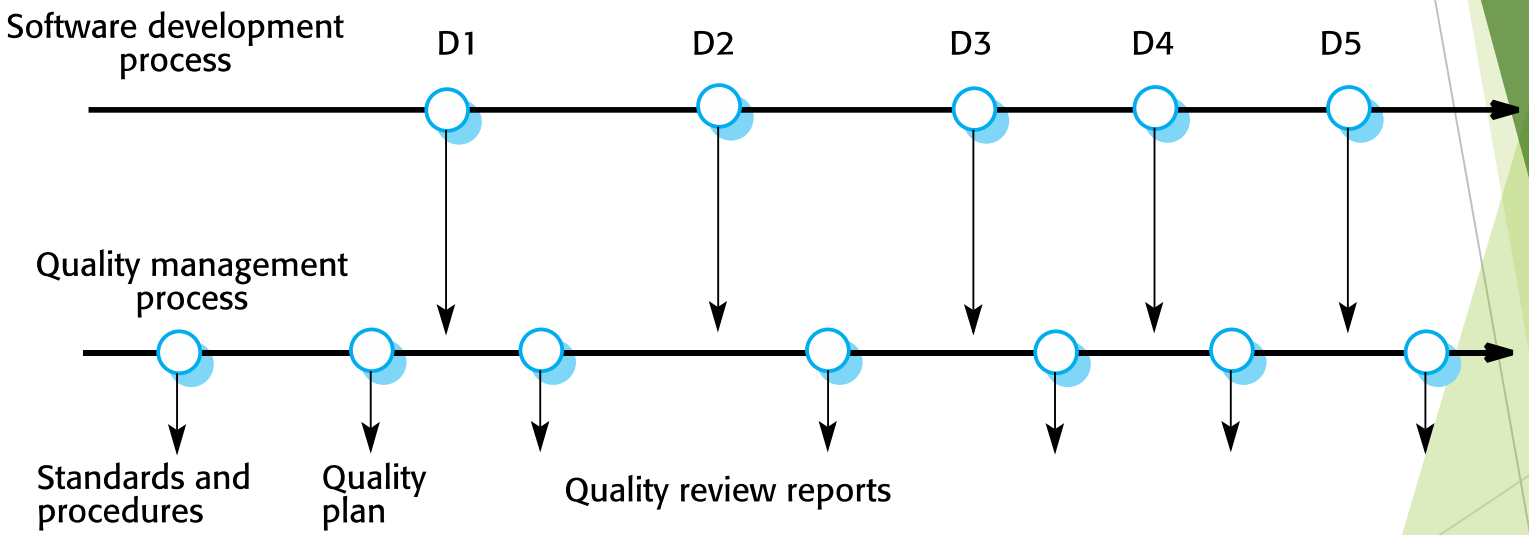
Yazılım Kalite Yönetimi

- Yazılım kalite yönetim takımı, geliştirme takımından bağımsız bir grup olmalıdır.

- Kalite yönetimi, yazılım geliştirme süreci üzerinde bağımsız bir kontrol sağlar.

Yazılım Kalite Yönetimi

- Kalite yönetim süreci, proje çıktılarını, kurumsal standartlar ve amaçlar ile uyumlu olup olmadıkları amacıyla kontrol eder.



Yazılım Kalite Yönetimi

- Yazılım kalite planı, istenen ürün niteliklerini ve bunların nasıl değerlendirileceğini ve en önemli kalite özelliklerini tanımlar.
- Kalite planı kalite değerlendirme sürecini de tanımlamalıdır.
- Hangi kurumsal standartların uygulanması gerektiğini tanımlamalı ve gerektiğinde kullanılacak yeni standartları tanımlamalıdır.

Yazılım Kalite Yönetimi

- Humprey’ e göre yazılım kalite planı içeriği:
  - Ürün tanıtımı,
  - Ürün planları,
  - Süreç tanımları,
  - Kalite hedefleri,
  - Riskler ve risk yönetimi.

Yazılım Kalite Yönetimi

- Yazılım kalitesi
- İmalat sanayi için basit bir ifadeyle kalite, bir ürünün spesifikasyonlarını karşılaması gerektiği olarak düşünülür. Ancak bu ifade yazılım ürünü için problemlidir,
  - Gereksinimleri tam olarak yazmak başarılamayabilir. Bu da müşteri ile geliştiricinin farklı değerlendirmesine neden olur.
  - Paydaşların tamamının talebi spesifikasyona yansıtılamaz bunun sonucu olarak o paydaşlar yazılımı düşük kalite olarak değerlendirir.
  - Yazılım kalite niteliklerini ölçülebilir olarak yazmak zordur.

Yazılım Kalite Yönetimi

- Yazılım kalitesi
- Kalite yönetim takımı aşağıdaki sorular çerçevesinde değerlendirip karar verir:
  - Yazılım uygun şekilde test edildi mi?
  - Yazılım yeterince güvenilir mi?
  - Yazılım performansı normal kullanım için yeterli mi?
  - Yazılım kullanılabilir mi?
  - Yazılım yapılandırılması iyi ve anlaşılabilir mi?
  - Kodlama ve belgeleme standartları takip edildi mi?

Yazılım Kalite Yönetimi

- Yazılım kalitesinde fonksiyonel olmayan özelliklerin önemi:
  - Yazılımın sübjektif kalitesi büyük ölçüde işlevsel olmayan özelliklerine bağlıdır.
  - Yazılım işlevselliği tam olarak beklendiği gibi değilse, kullanıcılar bu soruna çözüm bularak yapmak istediklerini yapmanın başka yollarını bulurlar.
  - Ancak, fonksiyonel olmayan özelliklerde sorun varsa örneğin yazılım yavaş veya güvenilir değil ise kullanıcıların amaçlarına ulaşmaları pek mümkün olmayacaktır.

Yazılım Kalite Yönetimi

- Yazılım kalite özellikleri:

Emniyet(Safety)

Taşınabilirlik(Portability)

Test edilebilirlik(Testability)

Güvenilirlik(Reliability)

Yeniden kullanılabilirlik(Reusability)

Modülerlik(Modularity)

Sağlamlık(Robustness)

Öğrenilebilirlik(Learnability)

Anlaşılabilirlik(Understandability)

Güvenlik(Security)

Kullanılabilirlik(Usability)

Uyarlanabilirlik(Adaptability)

Dayanıklılık(Resilience)

Verimlilik(Efficiency)

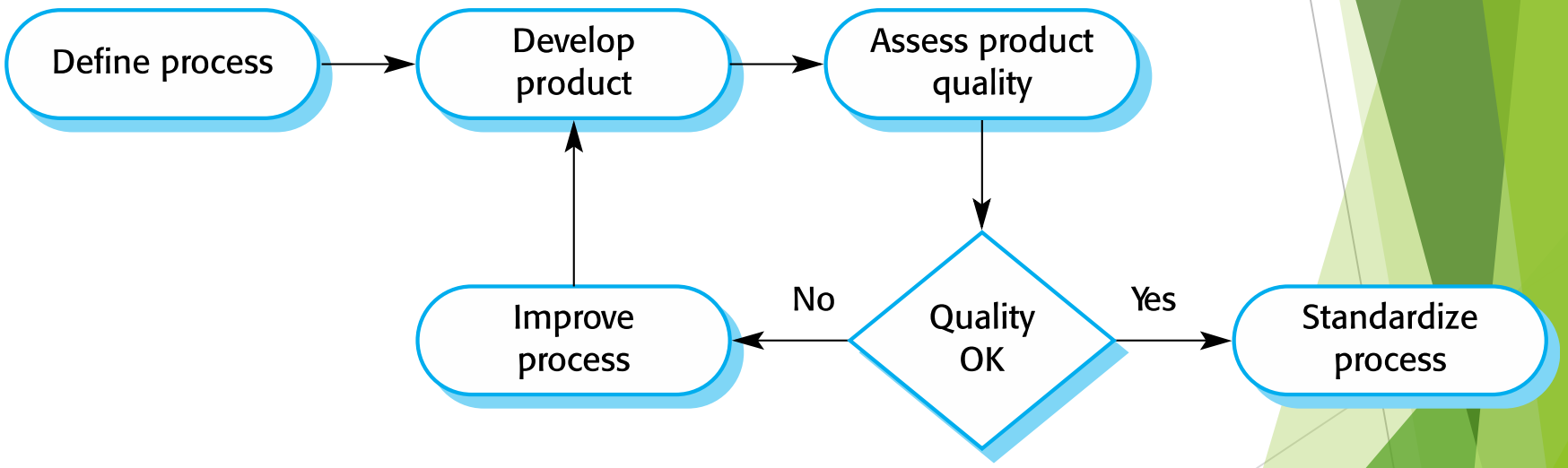
Karmaşıklık(Complexity)

Yazılım Kalite Yönetimi

- Yazılım kalite özelliklerinin tümünün sağlanması zor olabilir:
  - Yazılım sisteminde bu kalite özelliklerinin tümü için en iyiyi sağlamak mümkün olmaz. Bunlardan birini iyileştirmek diğerinden feragat etmeyi gerektirebilir.
  - Bu nedenle kalite planında, geliştirilmekte olan yazılım için en önemli kalite özellikleri vurgulanmalıdır.
  - Kalite planında, bir kalite özelliğinin sağlanıp sağlanmadığını değerlendirmek için kalite değerlendirme sürecinin bir tanımı da bulunmalıdır.

Yazılım Kalite Yönetimi

- Süreç tabanlı kalite yaklaşımı
  - Geliştirilen bir ürünün kalitesi, üretim sürecinin kalitesinden etkilenir.



Yazılım Kalite Yönetimi

- Yazılım geliştirme süreci ve yazılımın kalitesi:
  - Yazılım süreçleri ile ürün kalitesi arasında çok karmaşık ve az anlaşılmış bir ilişki vardır.
    - Bireysel yetenek ve deneyimler özellikle yazılım tasarım süreci olmak üzere yazılım geliştirmede önemlidir,
    - Bir uygulamanın yeniliği/farklılığı veya hızlandırılmış bir yazılım geliştirme talebi gibi dış faktörler ürün kalitesini düşürebilir.

Yazılım Kalite Yönetimi

- Yazılım kalite standartları

- Kalite yönetiminde önemli bir role sahip olan standartlar, bir ürün veya sürecin gerekli özelliklerini tanımlar.

- Standartlar uluslararası, ulusal, kurumsal veya proje standartları olabilir.

Yazılım Kalite Yönetimi

- Yazılım standartları neden önemlidir
- En iyi deneyimlerin standart olarak tanımlanması, kurumun bu deneyimi yeniden kullanmasına ve önceki hatalardan kaçınmasını sağlar.
- Standartlar kurumun kaliteye bakışını ifade eder, yazılım için kalitenin sağlanıp sağlanmadığı kararı için temel olur.
- Kurumda sürekliliği sağlar. Yeni işe başlayan personel kullanılan standartları anlayarak kuruluşu anlayabilir.

Yazılım Kalite Yönetimi

- Yazılım mühendisliği için standart tipleri
- Ürün standartları
  - Geliştirilen yazılım ürününe uygulanan standartlar. Bunlar, kodlama, gereksinim belgelerinin yapısı, nesne sınıfı tanımı vb. için standartları içerir.
- Süreç standartları
  - Bunlar, yazılım geliştirme sırasında izlenmesi gereken süreçleri tanımlar. Bunlar, spesifikasyon, tasarım ve doğrulama süreçlerinin tanımlarını, her bir süreçte hazırlanması gereken belgelerin bir tanımını içerebilir.

Yazılım Kalite Yönetimi

- Standartlar ile ilgili sorunlar

- Yazılım mühendisleri tarafından teknik işleriyle ilgili ve güncel olarak görülmeyebilirler.
- Genellikle çok fazla bürokratik form doldurmayı içerirler.
- Yazılım araçları ile yapılmıyorsa, standartlarla yönelik belgelemeyi sağlamak için sıkıcı form doldurma çalışmaları sıklıkla söz konusudur.

Yazılım Kalite Yönetimi

- Standartlar ile ilgili sorunlara çözümler

- Yazılım mühendisleri standart tanımlama sürecine dahil edilirler. Mühendisler bir standardın altında yatan mantığı anlarsa onu sahiplenirler.
- Standartları ve kullanımlarını düzenli olarak gözden geçirip güncelleştirme yapılmalıdır. Standartlar güncel değilse uygulayıcılar arasındaki güvenilirliği azaltır.
- Standartlara uygun yazılım araç desteği sağlanmalıdır. Bürokratik işler bu şekilde ortadan kaldırılır.

Yazılım Mühendisliği

Kaynakça

- *Ian Sommerville, Çeviri editörü: Prof. Dr. N. Yasemin Topaloğlu, Yazılım Mühendisliği, Nobel akademik yayıncılık , 2018, 10. basımdan çeviri. (Ders kitabı)*

- https://iansommerville.com/software-engineering-book/slides/