



YAZILIM MÜHENDİSLİĞİ – Şubat 2025

Prof.Dr.Mehmet Sıddık Aktaş

GENEL BİLGİLER

BAŞARIM DEĞERLENDİRME

- Sınav tarihleri:
 - Ara sınav: 8. ders haftasında (7 Nisan 2025 Pazartesi), klasik,
 - Final sınavı: Final haftasında, test.
 - Sınavlar dersin grupları arasında ORTAK yapılacaktır.
 - 6. ve 11. haftalarda birer lab çalışması yapılacaktır.
 - **Tarihler değişebilir, bölüm sayfası ile yıldız e-postanızı izleyiniz.**
- Proje ödevi:
 - Takım çalışması olarak yapılacaktır.
 - Takımları öğrenciler belirleyecektir, ancak aynı ders grubundaki öğrenciler arasında kurulmalıdır.
 - Konular dersin yürütücüsü tarafından belirlenecektir.
 - Kodlama ve birim sınamaları içerecektir.
 - Sunumu yapılacaktır. Konular, tarih ve program sonradan bildirilecektir.
- Puanlama (değişebilir):
 - Vize * %30, Proje * %30, Final * %40



YAZILIM MÜHENDİSLİĞİ

Prof.Dr.Mehmet Sıddık Aktaş

GENEL BİLGİLER

KAYNAK KİTAPLAR

- Software Engineering: A Practitioner's Approach / Roger S. Pressman. McGraw/Hill, 2014, 8th ed.
- Applying UML and Patterns: Intro. OOAD & Iterative Development / Craig Larman. Prentice-Hall, 2004, 3rd ed.
- Software Engineering / Ian Sommerville. Pearson, 2015, 10th ed.
- UML Distilled / Martin Fowler. Addison Wesley, 2003, 3rd ed.
- ... ve diğerleri

İLETİŞİM

- Ders notları, proje bilgileri, duyurular, vb. için web sitesi duyurusu yapılacaktır
- Duyuru & Dokümanlar bağlantısı
 - Anlık duyurular için OBS sayfanıza ve Yıldız e-posta adresinize sıkça bakınız
 - E-posta adresim: aktas@yildiz.edu.tr



GENEL BİLGİLER

DERS İÇERİĞİ

- Yazılım Mühendisliğine Giriş (Pressman 2001, Chapter 1)
- Yazılım Geliştirmede Süreç Modelleri (Pressman 2001, Chapter 2)
- Gereksinim Mühendisliği (Pressman 2001, Chapter 10.5, 11.5)
- Nesneye Yönelik Çözümleme (Larman)
- Nesneye Yönelik Tasarım (Larman)
- Yazılım Sınama Teknikleri (Pressman 2001, Chapter 8.5, 17.1, 17.2, 17.3, 17.6, 18)
- Yazılım Proje Yönetimine Giriş (Pressman 2001, Chapter 3)
- Yazılım Ölçütleri (Pressman 2001, Chapter 4 & 5, 19.1, 19.2, 19.3)
- Yazılım Risk Yönetimi (Pressman 2001, Chapter 6)



- Bu yansı ders notlarının sayfa düzeni için boş bırakılmıştır.



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr.Mehmet Sıddık Aktaş

YAZILIM MÜHENDİSLİĞİNE GİRİŞ



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM

- Yazılım :
 - Herhangi bir boyuttaki herhangi bir tür donanımda çalışan bilgisayar programını VE,
 - Basılı veya elektronik ortamdaki her tür dokümanı içeren ürün.
 - Dokümanlar yazılım geliştirme ve son kullanıcıya yönelik olabilir.
- Yazılım bir üründür, ancak başka ürünler geliştirmeye veya elde etmeye yarayan bir araç da olabilir.
- Yaşam döngüsü: Yazılımın bir fikir olarak doğmasından, kullanım dışı bırakılmasına kadar geçen süreç.
- Yazılım fiziksel bir ürün olmadığı için aşınmaz, ancak zamanla yetersizleşebilir.
 - Değişim kaçınılmazdır: Yazılım, yaşam döngüsü süresince değişikliklere uğrar.
 - Değişiklikler, yazılımda yeni hatalar oluşturabilir.
 - Yeni hatalar tam olarak düzeltilmeden yeni değişiklikler gerekebilir.
- Çözüm: Yazılım mühendisliği ilkelerine uyularak daha iyi tasarlanmış yazılım.



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM TÜRLERİ

- Sistem Yazılımı :
 - Diğer programlara hizmet sunmak üzere hazırlanmış programlar.
 - Derleyiciler, işletim sistemleri, vb.
- Mühendislik Yazılımı / Bilimsel Yazılım :
 - Mühendislik ve bilimsel hesaplamalarda kullanılmak üzere hazırlanmış programlar.
 - Büyük hacimli verilerle uğraşır.
 - “Numara öğretmek / Number crunching”.
- Gömülü (Embedded) Yazılım :
 - Donanım ile çok sıkı ilişkidir.
 - Denetim amaçlıdır.
 - Gerçek zamanlı uygulamalardır.

YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM TÜRLERİ

- Uygulama Yazılımı :
 - Product-line, shrink-wrapped, (commercial) off-the-shelf, RUSP (Ready to Use Software Product), vb.
 - Bkz. TS/BS ISO/IEC 25051 COTS Yazılım Ürünleri standardı
 - Bir çok mühendislik alanında olduğu gibi Yazılım Mühendisliği alanında da tanımlanmış standartlar vardır.
 - Erişim için kütüphaneye başvurunuz.
 - Ciddi bilgilere erişim için kütüphaneler kullanılmalıdır.
 - Farklı müşteriler tarafından kullanılabilecek genel amaçlı yazılımlar
 - Cari hesap uygulamaları, çeşitli otomasyon programları, kelime işlem uygulamaları, vb.
- Kurumsal Yazılım:
 - Belirli ticari iş gereksinimlerine yönelik programlar.
 - İş süreçleri ile ilgili bilgiye sahip olmalıdır.
 - Genellikle müşteriye özel tasarlanır.
 - Veri dönüştürme ve değerlendirme uygulamaları, iş süreçlerinin kimi zaman gerçek zamanlı izlenilmesi, vb.
 - Zamanla "eski yazılım" haline dönüşür!



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

ESKİ YAZILIM (Legacy Software):

- İş sürecinin önemli bir parçası olan ve çok uzun süredir kullanılan yazılımlar.
- Eski yazılımda bulunabilecek olumsuzluklar:
 - Eksik veya hatalı dokümantasyon
 - Zamanla karmaşıklaşmış kod
 - Esnek olmayan yapı
 - Eski donanımla çok sıkı ilişki
 - Yazılım mühendisliğindeki gelişmelerden yoksunluk nedeniyle düşük kalite.
- Eski yazılımın değiştirilmesini gerektiren nedenler :
 - İş alanındaki yeni gereksinimler
 - Güncel sistemlerle birlikte çalışabilmesi için uyumluluk kazandırılması
 - Donanımın ömrünün dolması nedeniyle daha güncel ortama taşınma gerekliliği



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIMI ETKİLEYEN EĞİLİMLER

- Yaygınlaşan Bilgi-İşlem :
 - Hesaplama gücünün giderek küçülen alanlara sıkıştırılabilmesi, bilişimin günlük yaşantımızla daha kolay bütünleşmesine olanak sağlıyor.
- Yaygınlaşan Haberleşme Ağı :
 - Kablosuz ağların yaygınlaşması, bilişimin günlük yaşantımızla daha kolay bütünleşmesine olanak sağlıyor.
- Özgür / Açık Kaynak Yazılım :
 - Gevşek bir ekip tarafından geliştirilen yazılım, daha anlaşılır ve geliştirilebilir olmalıdır.
- Ayrıca:
 - Takım çalışması zorunluluğu
 - Küreselleşme
 - Ekonomik krizler



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM HAKKINDAKİ YANILGILAR: MÜŞTERİ AÇISINDAN

- Programın yazılmasına başlanması için amaçları genel olarak belirlemek yeter, ayrıntılar sonra kararlaştırılabilir. Nasıl olsa yazılım esnektir.
 - Belirsiz gereksinimler, çürük atılmış temele benzer.
- Yazılım esnektir. Değişen gereksinimler kolayca sisteme uyarlanabilir.
 - Yazılım yaşam döngüsünde ilerledikçe, değişen gereksinimleri yazılıma uyarlamanın bedeli üstel olarak artar.
- Sonuç: Yazılım esnek bir oyun hamurundan çok kil veya cam gibidir.
 - Çevik süreçlerle esnekliğin artırılması hedeflenmektedir.



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM HAKKINDAKİ YANILGILAR: PROGRAMCI AÇISINDAN

- Yazılımı tamamlayıp müşteriye teslim edince işimiz biter.
 - Yazılım üstünde harcanan çabanın yarısından fazlası, yazılımın müşteriye ilk teslimatından sonra harcanmaktadır.
- Yazılımı tamamlamadan kalitesini ölçemem.
 - Kalite güvence yöntemleri yazılım hayat döngüsünün her aşamasında uygulanabilir.
 - Çözümleme sürecinde dahi kullanılabilecek kalite ölçütleri bulunmaktadır.
- Yazılım eşittir program.
 - Gereksinim analizi başlı başına bir emektir.
 - Dokümantasyon ve sinama çalışmalarını da unutmayın!
 - Bazı durumlarda entegrasyon çalışmaları da gerekmektedir.
- Yazılım mühendisliğinin gereklerini uygulayarak boşuna çaba harcıyoruz.
 - Haritası olmayan yolunu kaybeder.
 - Kalite için harcanan çaba, karşılığını yazılım hayat döngüsünün ilerleyen aşamalarında fazlasıyla ödeyecektir.
 - Küresel ölçekte yazılım projelerinin %50'si başarısızlığa uğramaktadır.



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM HAKKINDAKİ YANILGILAR: İDARİ

- İşler yetişmiyorsa takıma yeni programcılar ekleriz.
 - Yazılım hayat döngüsü içerisinde ilerledikçe, yeni elemanların yazılıma hakim olması üstel olarak zorlaşır. İşler daha da gecikir.
- Geliştirmesini üstlendiğim yazılımı tamamen veya kısmen fason yaptırırım.
 - Proje ilerlemesini kendi içinde denetleyemeyen bir firma, dışarıya verdiği işi izlemekte de zorlanacaktır.
- Açık kaynak yazılım üretirsem kar edemem.
 - Danışmanlık hizmetleri ile kar edilebilir.
 - Başka iş modelleri de vardır.



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM SÜREÇLERİNİN GENEL ADIMLARI

- Çözümleme (Analysis)
- Tasarım (Design)
- Gerçekleme (Implementation)
- Sınama (Testing)
- Bakım (Maintenance)

ÇÖZÜMLEME

- Çözümleme: Bir şeyi anlayabilmek için parçalarına ayırmak.
- Gerçeklenecek sistemi anlamaya yönelik çalışmalardan ve üst düzey planlama eylemlerinden oluşur.
 - Uygulama alanı
 - Kullanıcı gereksinimleri
 - Program parçaları arasındaki üst düzey ilişkiler ve etkileşimler (NYP'deki parçalar: sınıflar ve nesneler)
- “Bir sorunu anlamadan çözemezsiniz.”



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM SÜREÇLERİNİN GENEL ADIMLARI

- Çözümleme
- Tasarım
- Gerçekleme
- Sınama
- Bakım

TASARIM

- Tasarım: Bir araştırma ve/veya geliştirme sürecinin çeşitli dönemlerinde izlenecek yol ve işlemleri tasarlayan çerçeve.
- Çözümleme ile anlaşılan sorun tasarım aşamasında kağıt üzerinde (!) çözülür.
- Yazılım ↔ Tasarıma yönelik şemalar (NYP'de bazı tür UML şemaları), elektronik ↔ devre şemaları, mimari ↔ kat planları

GERÇEKLEME

- Eldeki tasarım, bir programlama dili ile kodlanır.



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM SÜREÇLERİNİN GENEL ADIMLARI

- Çözümleme
- Tasarım
- Gerçekleme
- Sınama
- Bakım

SINAMA

- Sınama neden önemlidir?
 - Yazılım sürecinde ilerledikçe, ortaya çıkabilecek hataların giderilme maliyeti üstel olarak artar.
 - Aksi gibi, hataların büyük çoğunluğunun kökenleri isteklerin belirlenmesi ve tasarım aşamalarındaki sorunlara dayanır.
 - Bu yüzden: Erkenden, sık sık ve kolay sınama yapın.



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM SÜREÇLERİNİN GENEL ADIMLARI

- Çözümleme
- Tasarım
- Gerçekleme
- Sınama
- Bakım

BAKIM

- Yazılımın faaliyete geçirilmesinden sonra sistemde yapılan değişikliklerdir.
 - Yazılım hatalarının düzeltilmesi:
 - Kodlama hataları
 - Tasarım hataları (!)
 - Gereksinim ve analiz hataları (!!)
 - Sistemin işlevlerini değiştirme veya işlevlere eklemeler/çıkarmalar,
 - Yazılımın farklı bir ortama taşınması (programlama dili, işletim sistemi, donanım, iklim, vb.) (porting)



YAZILIM MÜHENDİSLİĞİNE GİRİŞ

YAZILIM SÜREÇLERİNİN GENEL ADIMLARI

- Çözümleme
- Tasarım
- Gerçekleme
- Sınama
- Bakım

BAKIM

- Yeniden mühendislik (Refactoring / Software re-engineering)
 - Teknik bakış açısı: Yazılımın işlevini değiştirmeden iç yapısını değiştirmek.
 - Olası eylemler:
 - Yazılımın belgelendirilmesi
 - Tasarımın iyileştirilmesi/değiştirilmesi
 - Yazılımın farklı bir ortama taşınması



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr. Mehmet Sıddık Aktaş

CASE YÖNTEM VE ARAÇLARI



CASE SİSTEMİ

GENEL BİLGİLER

- CASE (Computer Aided Software Engineering); yazılım mühendisliği görevlerinin, bilgisayar yardımı ile otomatik olarak (!) gerçekleştirilmesidir.
- CASE ortamı, belirli bir sorunun çözümü ya da bir kurumun yönetim bilişim sistemi için gerekli yazılımı otomatik olarak geliştiren bir bilgisayar sistemidir.
 - Bu sistem; otomatik yazılım araçlarının ve yapısal yöntemlerin bir kombinasyonu olup, yazılım geliştirme sürecini adım adım izleyerek gerçekleştirmekte ve yazılımı üretmektedir.
 - Avrupa ülkelerinde "bütünleşik programlama desteği ortamı" anlamında IPSE (Integrated Programming Support Environments) adı da verilir



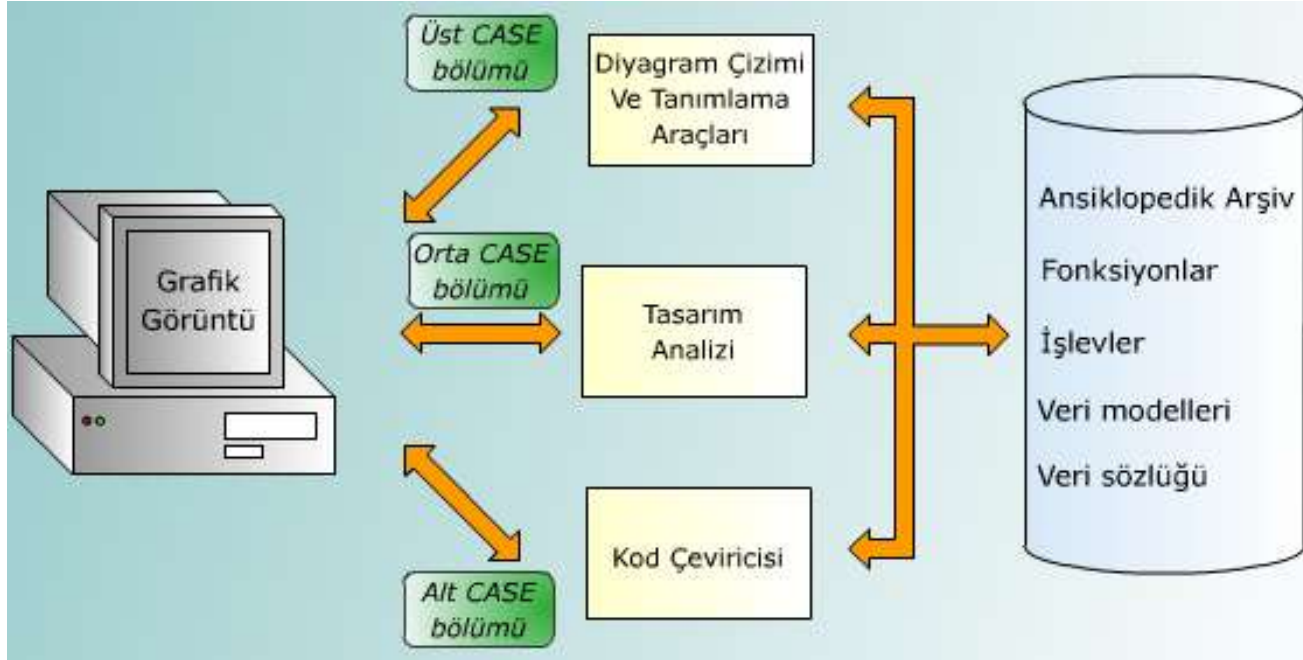
CASE SİSTEMİ

CASE SİSTEMİ ÖĞELERİ

- CASE sisteminin 3 ögesi:
 - İşlemler: Yazılımın geliştirme sürecini düzenlemekte kullanılan yöntem bilimi
 - Yöntemler: Projenin gerektirdiği standart tasarım teknikleri ve yöntemleri
 - Bütünleşik otomatik yazılım geliştirme araçları
- İşlemler ve yöntemlere ait bilgiler bir ansiklopedik arşivde saklanmaktadır.
- Bu bilgileri kullanarak, otomatik araçlar, diyagram çizimi ve betimleme, tasarım analizi, kodlama işlemlerini gerçekleştirmektedirler.
- İşlemlerin yürütülmesinde etkileşim ve ekrandan yararlanılmaktadır.
- Her basamakta elde edilen sonuçlar, sırası geldiğinde kullanılmak üzere ansiklopedik arşivde saklanmaktadır.

CASE SİSTEMİ

CASE TEKNOLOJİSİ



- Bölümler:
 - Üst CASE bölümü: Bilgisayara dayalı diyagram çizim araçları şeklinde arayüz üzerinden kullanılan, iş mantığını betimleyen gereçler.
 - Orta CASE bölümü: Sistem analizi ve tasarımı gereçleri.
 - Alt CASE bölümü: Kod üretim gereçleri
- Orta ve alt bölümlerde ölçüm ve belgelendirme ile hata ayıklama gereçleri de yer almaktadır (otomatik).

YAZILIM GELİŞTİRME YÖNTEMLERİ

DÖRDÜNCÜ KUŞAK TEKNİKLERİ

- 4. Kuşak diller (4th Generation Language: 4GL) olarak tanımlanan diller kullanılarak, grafik arayüzler üzerinden yazılımın kaynak kodunun değiştirilebilmesini sağlayan gereçlerin (4GT) kullanıldığı tekniklerdir.
- 4GL diller doğal dile yakın dillerdir. 4GT gereçler komutları yerine getirebilecek bir gösterim üzerinden (şemalar, akış diyagramları gibi) kaynak kod üretebilmektedir.
- Şema ile kod arasında çift yönlü dönüşüm söz konusudur.
- Programlama bilmesi gerekmeyen konu uzmanlarının ve üst yöneticilerin kolaylıkla yazılıma müdahale edebilmesi avantajını vaat etmektedir.
 - Yine de algoritmik düşünme yeteneği olmadan kullanılamazlar.
- Bu araçlar günümüzde sadece bazı özel alanlarda ve sınırlı olarak kullanılabilmektedir

Slayt 23

YES12

Domain Specific Languages

Yunus Emre Selçuk, 3/20/2019



- Bu yansı ders notlarının sayfa düzeni için boş bırakılmıştır.



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr. Mehmet Sıddık Aktaş

YAZILIM GELİŞTİRME SÜREÇ (MODEL)LERİ



YAZILIM GELİŞTİRME SÜREÇLERİ

- Yazılım geliştirme bir süreç olarak ele alınmalıdır.
 - Süreç: Önceden belirlenmiş adımlardan oluşan iş akışı.
- Süreç modelleri, yazılım geliştirme sürecinin yapısını ve adımlarını belirler.
 - Önceden ve iyi planlanmış bir süreç, 'zamanında' ve 'kaliteli' bir 'ürün' elde edilmesini sağlar.
- Çeşitli modellerin kendine özgü avantaj ve dezavantajları vardır.
 - Gerçeklenecek projeye uygun modelin seçilmesi gerekir.



YAZILIM GELİŞTİRME SÜREÇLERİ


ŞELELE MODELİ

- Ardışıl Model / Şelale Modeli (Sequential / Waterfall)
 - Adımlar: Çözümleme → Tasarım → Kodlama → Sınama → Bakım.
 - Bir adımın tamamlanmasından sonra diğerine geçilir.
 - Eksiklikler veya hatalar fark edilirse bir önceki adıma geçilir.
- Artılar:
 - En eski model, yaygın kullanımda.
 - İyi tanımlanmış adımlar.
- Eksiler:
 - Son ürünün eldesi uzun süreceğinden müşteri sabırlı olmalıdır.
 - Adımları geride bıraktıkça, ilerleyen aşamalarda karşılaşılan hataların düzeltilmesi üstel olarak zorlaşmaktadır.
 - Bir çok 'müşteri' de gereksinimleri eksiksiz ve kesin belirtmekte zorlanmaktadır.
- Sonuç: Hiç model kullanmamaktan iyidir!
 - Önceden bir çok kez başarıya ulaştırılmış projelere benzer yeni projelerin yürütülmesi için kullanılabilir (rutin projeler).



YAZILIM GELİŞTİRME SÜREÇLERİ

ÖN ÜRÜN MODELİ

- Ön ürün modeli / Prototip modeli
 - Adımlar: Müşteriyi dinle – Ön ürün oluştur – Müşteri ön ürünü dener –

- Artılar :
 - Kullanıcı gereksinimlerinin daha iyi elde edilmesi.
 - Kullanıcının erkenden ürünü değerlendirmeye başlayabilmesi.
- Eksiler :
 - Ön ürün mükemmel değildir.
 - Eksik ürün zaman ve maliyet kısıtlamaları nedeniyle olgunlaşmadan canlı kullanıma alınabilmektedir.
- Sonuç: Prototip oluşturmayı başlı başına bir model olarak kullanmamalı, daha olgun bir modelin analiz aşamasında kullanılacak bir araç olarak ele almalı ve prototip ürünü silip atmalı.



YAZILIM GELİŞTİRME SÜREÇLERİ

HIZLI UYGULAMA GELİŞTİRME (RAD: Rapid Application Development)

- Kısa geliştirme çevrimleri üzerinde duran artımsal bir model.
- Ön koşullar:
 - Uygulamanın yaklaşık/ortalama 3 aylık bölümlere ayrılabilmesi,
 - Yeterli sayıda bölümün eşzamanlı ilerlemesinin sağlanabilmesi,
 - Yazılımın bileşenlerden kurulabilmesi.
- Artılar:
 - Bu sürece uygun yazılım projelerinde verimliliğin artması.
- Eksiler:
 - Büyük ölçekli çalışmalarda yeterli sayıda bölümü eşzamanlı ilerletebilecek sayıda çalışanın bulunamaması.
 - Çalışanlar hıza uyum sağlayabilmelidirler.
 - Yüksek teknik risklere uygun değil.
- Sonuç:
 - Prototip geliştirmede kullanılması veya ana fikirlerinin diğer süreçlere uygulanması yerinde olacaktır.




YAZILIM GELİŞTİRME SÜREÇLERİ

BİLEŞEN TABANLI (Component Based) UYGULAMA GELİŞTİRME

- Uygulamanın hazır yazılım bileşenlerinden oluşturulmasını öngörür.
- Aşamaları:
 - Konu alanı mühendisliği (Domain Engineering)
 - Aday bileşenlerin sınıflandırılması ve seçilmesi (Qualification)
 - Seçilen bileşenlerin kendi yazılımımıza uyarlanması (Adaptation)
 - Bileşenlerin bir araya getirilmesi (Composition)
- Artılar:
 - Yeniden kullanımın özendirilmesi (azalan giderler?)
- Eksiler:
 - Uygun bileşenlerin bulunması gerekliliği (her zaman bulunmaz)
 - Bileşenlerin uyarlanması gerekliliği (göründüğü kadar kolay olmayabilir)
- Sonuçlar:
 - Özellikle hızlı uygulama geliştirme olmak üzere, ana fikirleri çeşitli süreçlere uygulanabilir.

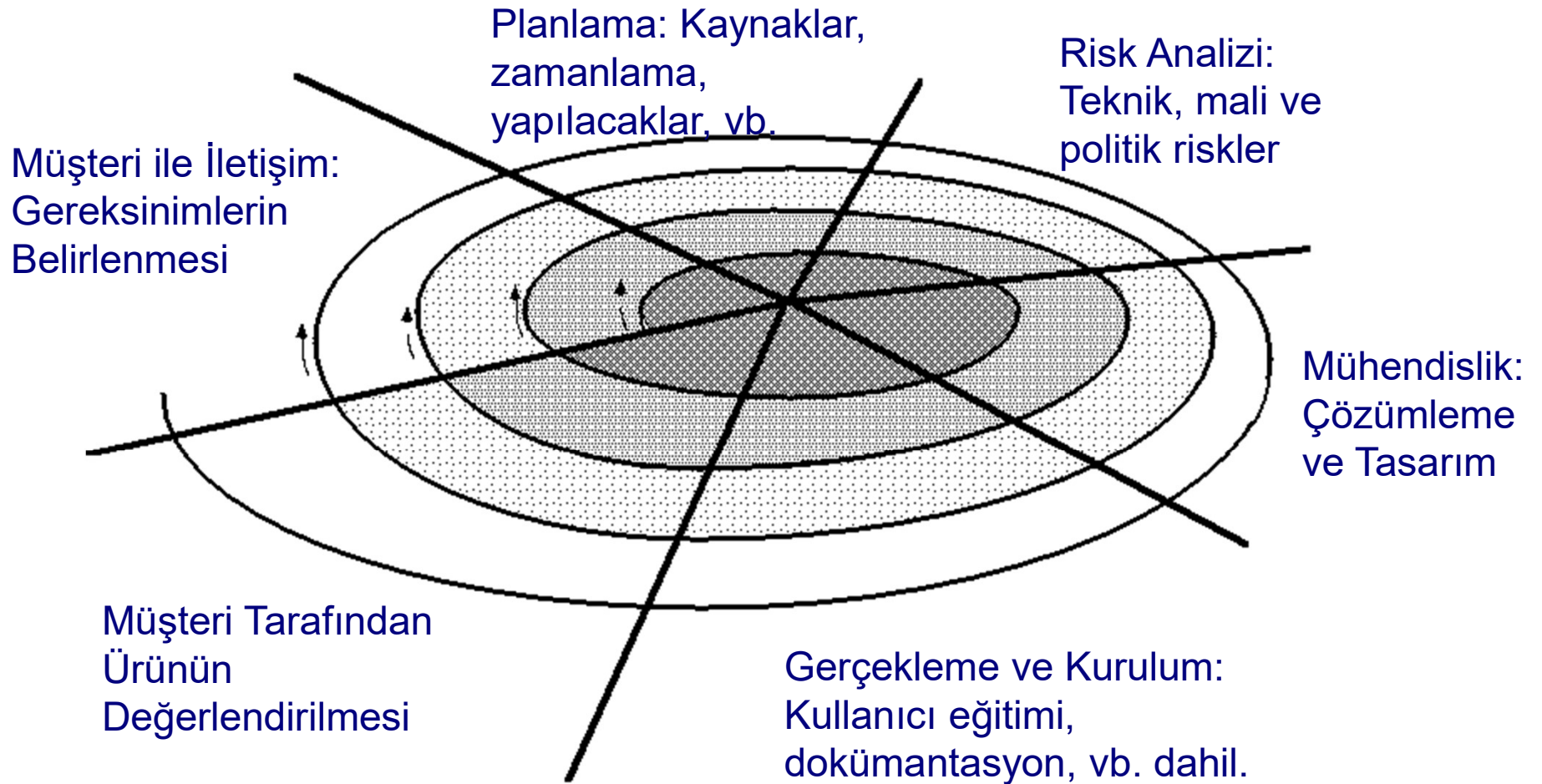
YAZILIM GELİŞTİRME SÜREÇLERİ

ARTIMSAL / YİNELEMELİ MODELLER

- Artımsal / Yinelemeli Modeller (Incremental / Iterative)
 - Adımlar: Analiz – Tasarım – Kodlama – Sınama → Bakım
 - Gereksinimler önemlerine ve birbirine bağımlılıklarına göre sıralanarak her yinelemede bunların bir kısmı tamamlanır.
- Artılar :
 - Ön ürün modeli ve ardışıl modelin güçlü yönlerini kendinde toplayarak dezavantajlarını geride bırakmıştır.
 - Nesneye yönelik programlama metodolojisi ile uyum içerisindedir.
- Eksiler : Yazılımın küçük artımlarına fazla yoğunlaşmak, sistemin geneline bakıldığında kolayca görülebilecek sorunların gözden kaçmasına neden olabilir.
- Sonuçlar:
 - Sistemin genelini göz ardı etmemek şartıyla güçlü bir modeldir.
- Örnekler: Spiral Model ve Kazan-Kazan Modeli

YAZILIM GELİŞTİRME SÜREÇLERİ

SARMAL (Spiral) MODEL



KLASİK YİNELEMELİ SÜREÇLER

Kazan-Kazan Modeli (WINWIN Model)



- Paydaş: Yazılımın başarısı ve başarısızlığının etkileyeceği kişi ve kurumlar.

YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Değişen gereksinimler, teknik riskler gibi önceden belirlenemeyen durumlara ve yazılım ürününü etkileyebilecek her tür değişikliğe karşı esneklik sağlayan süreçlerdir.
- Bireyler ve etkileşimler
- Çalışan yazılım
- Müşterinin sürece katılımı
- Değişikliklere uyum sağlamak
- Süreçler ve gereçler
- Ayrıntılı belgeler
- Sözleşme pazarlığı
- Bir planı izlemek
- Çevik süreçler, sağ taraftaki maddelerin yararını kabul etmekle birlikte, sol taraftaki maddelere daha çok önem vermektedir.
- Bir ilerleme olmaksızın yalnızca sürekli uyum sağlamak başarı değildir.
 - Yazılımın artımsal gelişimi
 - Müşteriye erken ve sık ürün teslimi
 - Başarımın birincil ölçütü doğru çalışan yazılımdır.
 - Bu nedenle sınaama çok önemlidir, test güdümlü tasarım tavsiye edilir (TDD: Test Driven Design).

YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Çevik süreci yürütecek ekibin özellikleri:
 - Yüz yüze görüşme, en etkili bilgi aktarım yoludur.
 - Takım üyeleri çevik yaklaşım hakkında eğitilmelidir.
 - Ekip üyelerinin ortak amacı, çalışan yazılım üreterek müşteriye zamanında teslim etmek olmalıdır.
 - Ekip üyeleri birbirleriyle ve müşteriyle işbirliği içinde olmalıdır.
 - Ekip üyeleri karşılıklı saygı ve güven içerisinde olmalıdır.
 - Ekipler hem teknik, hem de tüm proje hakkında kararlar verebilmelidir.
 - Boşuna harcanan çaba yoktur: Çözülen bir sorun gereksizleşse bile, çözüm sürecinde edilen deneyim ekibe ileri aşamalarda yararlı olabilir.
 - Kendi kendini düzenleme:
 - Ekibin kendisini yapılacak işe göre uyarlaması,
 - Ekibin kullanacağı süreci yerel ortama uyarlaması,
 - Üstünde çalışılan artımsal yazılım parçasını teslim etmek için gerekli çalışma zamanlamasını ekibin kendisinin belirlemesi.



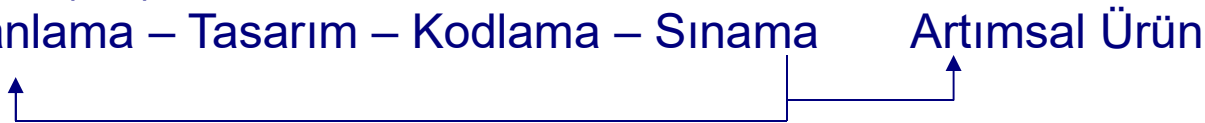
YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Çevik süreçlerin dezavantajları:
 - Uygun olmayan ekiple çevik çalışılamamaktadır.
 - Kalabalık ekip veya büyük ölçekli projeler için uygun görülmemektedir.
 - Bir dış denetleyicinin dahil olduğu ve ayrıntılı kuralların gerektiği denetlemelerin zorunlu olduğu projelerde yetersiz kalmaktadır.
 - Çevik çalışmak disiplinsizlik olarak yorumlanmamalıdır.
- Çevik Süreç Örnekleri:
 - Aşırı Programlama (XP: Extreme Programming)
 - Scrum

YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Aşırı Programlama (XP)
 - Adımlar: Planlama – Tasarım – Kodlama – Sınama 
- Planlama:
 - Müşteri, kullanıcı öyküleri oluşturur.
 - Müşteri, öyküleri önemine göre derecelendirir.
 - Yaklaşık 3 haftada gerçekleştirilemeyecek öyküler varsa, ekip müşteriden bunları alt öykülere bölmelerini ister.
 - Ekip ve kullanıcı, öykülerin sıradaki artımsal ürüne nasıl ekleneceğine karar verir:
 - Ya önce yüksek riskli öyküler gerçekleştirilir,
 - Ya da önce yüksek öncelikli öyküler gerçekleştirilir.
 - Her olasılıkta tüm öyküler kısa sürede (birkaç hafta) gerçekleştirilmelidir.

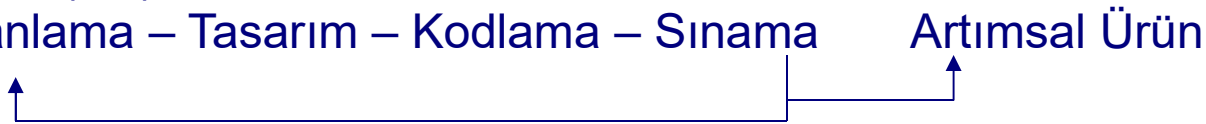
YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Aşırı Programlama (XP)
 - Adımlar: Planlama – Tasarım – Kodlama – Sınama → Artımsal Ürün
- Planlama (Devam):
 - İlk artımsal ürün projenin hızını ölçme amacıyla değerlendirilir:
 - Eldeki artımın hızına göre sonraki artımların teslim tarihleri belirlenir.
 - Aşırı sözler verildiği ortaya çıkarsa artımsal ürünlerin içeriği de yeniden kararlaştırılabilir.
 - Süreç ilerledikçe müşteri yeni öyküler ekleyebilir, eski öykülerin önceliğini değiştirebilir, öyküleri farklı şekillerde bölüp birleştirebilir, bazı öykülerden vazgeçebilir.
 - Bu durumda ekip kalan artımları ve iş planlarını uygun biçimde değiştirir.

YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Aşırı Programlama (XP)
 - Adımlar: Planlama – Tasarım – Kodlama – Sınama 
Artımsal Ürün
- Tasarım:
 - Basit tasarım karmaşık gösterimden üstündür. (KISS: Keep It Simple, Stupid!)
 - CRC (Class-Responsibility-Collaboration) kartları ile yazılımın sınıf düzeyinde incelenmesi.
 - Karmaşık bir tasarımdan kaçınılamazsa işlevsel bir ön gerçekleştirme yapılır (Spike solution).
 - Refactoring teşvik edilir.
 - Bu aşamanın ürünleri CRC kartları ve ön gerçeklemlerdir (başka ürün yok).

YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

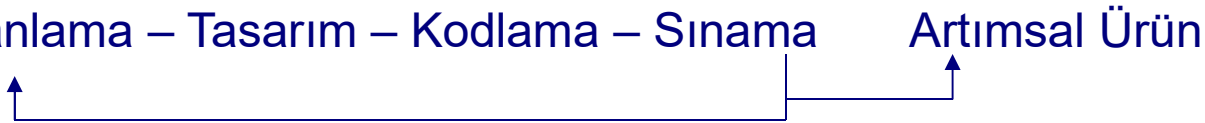
- Örnek CRC kartı:

Sınıf adları

Sınıf: Satış	
Kasada yapılan ödemeyi simgeleyen sınıf.	
Üst Sınıf(lar): Yok	
Alt Sınıf(lar): Yok	
Sorumluluk:	İşbirlikçi:
Satışın yapıldığı tarih ve saati saklamak	
Yapılan ödeme tutarını saklamak	Ödeme
Satılan malların listesine erişim	Mal

YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Aşırı Programlama (XP)
 - Adımlar: Planlama – Tasarım – Kodlama – Sınama 
- Kodlama:
 - Önce birim sınamaları hazırlanır.
 - Programcı tarafından yapılan, sınıfların (NYP'de; yapısal'da fonksiyonlar, vb.'lerin) temel işlevselliklerini sınama amaçlı kod.
 - Sadece sınavı geçmeye yarayan kod yazılır (KISS).
 - Çift kişi ile kodlama:
 - Bir programcı eldeki sorunu çözerken diğeri çözümün genel tasarıma uygunluğunu gözetir ve kodlamanın takımın karar verdiği ölçütlere (kalite, vb.) uygunluğunu denetler.
- Sınama:
 - Birim sınamalarının otomatik çalıştırılması.
 - Müşterinin artımsal ürünü denemesi.

YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Scrum:
 - Adımlar: Görev Listesi – Koşu (Sprint) – İşlev Gösterimi
- Görev Listesi = Kullanıcı öyküleri.
 - Önceliklendirilmiştir.
- Koşu:
 - Görev listesinin maddelerinden biri seçilir ve önceden belirlenmiş kısa bir süre içerisinde (Ör. 1-4 hafta) gerçekleşir.
 - Koşu süresince ekibin her gün yaptığı kısa (Ör. 15dk) toplantılar:
 - Proje lideri yönetir.
 - Cevaplanmaya çalışılan üç ana soru:
 - Son toplantıdan bu yana ne yaptınız?
 - Karşılaştığınız engeller nelerdir?
 - Yarınki toplantıda neleri başarmayı hedefliyorsunuz?
- İşlev Gösterimi: Müşterinin en yeni işlevi veya o ana dek gerçekleşen tüm işlevleri sınaması.



YAZILIM GELİŞTİRME SÜREÇLERİ

ÇEVİK (Agile) SÜREÇLER

- Çevik Modelleme
 - Bir amaç için modelleme yapın:
 - Neyi, kime, hangi düzeyde anlatmak istiyorsunuz?
 - Buna göre uygun modelin ve ayrıntılandırmanın seçimi .
 - İçerik sunumdan daha önemlidir.
 - Gerekli bilgiyi içermeyen hatasız model işe yaramaz!
 - Kullandığınız modelleme yolunun özünü ve modellerinizi oluşturmak için kullanacağınız araçları iyi öğrenin.
- DİKKAT: Önemli olan dengeyi korumaktır.
 - Çevik çalışacağız diye serseri programcı olmayın.
 - Disiplinli çalışacağız diye sırtınızda tuğla çuvalı taşımayın.

YAZILIM GELİŞTİRME SÜREÇLERİ

SÜREÇ SERTİFİKASYONU

- Olgunlaşmış bir yazılım geliştirme sürecine sahip olmayan bir yazılım firması, projelerini başarı ile sonuçlandıramaz.
- Bir yazılım firması, süreçlerinin yeterliliğini bağımsız kurumlara onaylatmayı seçebilir.
- Gerekli olduğu durumlar:
 - Bazı büyük müşteriler sertifikalı yazılım firmaları ile çalışmayı şart koşarlar.
- Gereksiz olduğu durumlar:
 - Çok küçük şirketler ve/veya projeler için ek yük olarak görülebilir.
- Güncel model ve standartlar:
 - CMMI: Capability Maturity Model Integration
 - SEI tarafından önerilmiştir (Software Engineering Institute of Carnegie-Mellon University) (kurumsal)
 - PMI: Genel amaçlı bir proje yönetimi yaklaşımı (bireysel ve kurumsal)
 - ISO 9001:2015 standartları (Genel) (kurumsal)
 - ISO/IEC 90003:2018 (Yazılım geliştirmeye özel) (kurumsal)
 - Ulusal belgelendirici firma: Denetik (Ulusal otorite: TÜRKAK, kamu)
 - Genel vs. Özel (Peynir mi üretiyoruz?)

YAZILIM GELİŞTİRME SÜREÇLERİ

CMMI DÜZEYLERİ

- CMMI düzeyleri:
 - 0. Düzey: Yürütülmeyen (Level 0: **Incomplete**).
 - 1. Düzey: Giriş düzeyi (Level 1: **Initial/Performed**). İş şansa ve anahtar kişilere kalmış. Etkinliklerin tümü yürütülüyor ama bunların tanımlanması için hiçbir sistematik girişim yoktur, her seferinde farklılaşabilirler.
 - 2. Düzey: Yönetilen/Yinelenebilir (**Managed**/Repeatable). Temel planlama ve izleme yöntemleri kullanılarak, önceki projelerdeki başarılar yeni projelerde tekrarlanabilir.
 - 3. Düzey: Tanımlanmış (**Defined**). Kişi ve risk yönetimi ile projenin yönetimi iyileştirilir. Tüm organizasyonda standart süreçler tanımlanmıştır
 - Büyük müşteriler en az bu düzey yazılım evleri ile çalışmak ister.
 - 4. Düzey: Nicel Yönetilen (**Quantitatively Managed**). Süreç ve yazılım ölçütleri kullanılarak kalite yönetimine geçilir. İlerleme sürekli izlenir, bütçe ve zaman hedeflerinden sapmalar erkenden belirlenerek gerekli önlemler alınır.
 - 5. Düzey: İyileştirilmiş (**Optimized**). Süreç yönetimi geçmiş deneyimlerin ışığında sürekli iyileştirilir.



YAZILIM GELİŞTİRME SÜREÇLERİ

CMMI DÜZEYLERİ

- CMMI, her düzeyde belli süreç alanlarının kapsanıyor olmasını ister.
 - Süreç alanları belli hedeflere ulaşmak için beklenen uygulamalardır.
 - Her firma gerekli süreç alanlarını kendine özgü süreçlerle kapsar.
- CMMI ilgi alanları:
 - CMMI-DEV (Development): Yazılım geliştirme
 - CMMI-SVC (Service): Hizmet sunumu ve yönetimi
 - CMMI-ACQ (Acquisition): Ürün ve hizmet alımı
 - Bu üç alan, CMMI v2.0 ile (2018) tek bir olgunluk modeli altında birleştirilmiştir.
- Tarihçe:
 - 1987-1997: CMM
 - 1998-2009: CMMI
 - 2010: CMMI v1.3, çevik süreçler desteklenmeye başladı
 - 2018: CMMI v2.0
 - Continuous (kısmi, sadece bazı süreç etkinliklerini kapsayan) ve Staged (bütüncül, daha değerli) ayrımı kaldırılıp tüm süreç etkinliklerini kapsayacak içerikte birleştirildi.
 - Kaynakları ücretli/abonelik tabanlı hale geldi.

YAZILIM GELİŞTİRME SÜREÇLERİ

CMMI DÜZEYLERİ

- CMMI Level 3+ sertifikası almış kamu ve özel kurumlarımıza örnekler:
 - Level 5: MilSoft
 - Level 4: Maalesef yok
 - Level 3: 24 Kurum (bir kurumun birden fazla bölümü akredite olabilir)
 - Akgün Yazılım, ASELSAN, Başarsoft, EPIAŞ, Havelsan, Koç Bilgi ve Savunma Tek., Simsoft, STM Savunma Tek., Veripark, vb.
 - Ayrıntılar: <https://cmmiinstitute.com/pars/>
 - Not: Şubat 2025 itibariyledir. Terfi eden/düşen olabilir.

YES6

Slayt 47

YES6

Terfi eden/düşen var mı?

Yunus Emre Selçuk, 3/31/2017



YAZILIM YETERLİLİK OLGUNLUK MODELİ

CMMI ve CASE

- CMMI'da Bilgisayar destekli yazılım mühendisliği gereçlerinin (CASE: Computer Aided Software Engineering tools) kullanımı:
 - Düzey 1. Başlangıç düzeyi:
 - Yazılım araçlarının kullanımına dair hiçbir standart yoktur.
 - Yazılım ölçütleri kullanılmaz.
 - Yeniden kullanılabilirlik yoktur veya minimal düzeydedir.
 - Düzey 2. Tekrarlanabilir düzey:
 - Yazılım aracı kullanımına yönelik proje standartları vardır.
 - Çalışanlara yazılım araçlarının kullanımına yönelik eğitim verilmiştir.
 - Bazı takım tabanlı yazılım araçları, diğer yazılım araçları ile tümleştirilmiştir (entegrasyon).
 - Tasarım ve kodun bir kısmı sınırlı olarak yeniden kullanılabilir.



YAZILIM YETERLİLİK OLGUNLUK MODELİ

CMMI ve CASE

- CMMI'da Bilgisayar destekli yazılım mühendisliği gereçlerinin (CASE: Computer Aided Software Engineering tools) kullanımı (devam):
 - Düzey 3. Tanımlanmış düzey:
 - Yazılım araçlarının erişebildiği, organizasyona ait veri deposu oluşturulmuştur.
 - Ortak tekrar kullanım kütüphanesi oluşturulmuştur.
 - Organizasyon standartlarına bağlı olarak yazılım ölçütleri (metrikler, ileride değinilecek) toplanmıştır.
 - Ölçütler eğilimler ve profillerin belirlenmesi gibi işlemlerde kullanılmak üzere organizasyona ait veri tabanında saklanmıştır.
 - Düzey 4. Yönetilebilir düzey:
 - Yazılım ölçütlerinin niteliksel analizi ile süreç gelişimi oluşturulmuştur.
 - Düzey 5. Optimize edilmiş düzey:
 - Yeni yazılım araçları dahil olmak üzere tüm gereçler, teknikler ve proje yönetim süreçleri değerlendirilip sürekli iyileştirmeye tabi tutulmaktadır.



- Bu yansı ders notlarının sayfa düzeni için boş bırakılmıştır.



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr. Mehmet Sıddık Aktaş

GEREKSİNİM MÜHENDİSLİĞİ



GEREKSİNİM MÜHENDİSLİĞİNE GİRİŞ

GEREKSİNİM MÜHENDİSLİĞİ

- Üzerinde çalışılmaya başlanacak projenin amaçlarını, boyutlarını ve etkilerini belirlemeye yönelik çalışmalardır.
 - Genel amaçlı proje yönetimi faaliyetleri arasında yer alan yapılabilirlik (feasibility) çalışmasına bir girdi olarak düşünülebilir.
- Müşteri ne istediğini bilmez mi? Gereksinimler zaten belli değil mi?
 - Çoğunlukla müşterinin kafasında sadece genel bir fikir vardır.
 - Yoruma açık ve ayrıntıları kesin çizgilerle belirlenmemiş gereksinimler projenin başarısızlığına davetiye çıkarır.
 - Kesin belirlenmiş gereksinimler bile zaman içerisinde değişebilir.
- Değişler:
 - Şeytan ayrıntıda gizlidir.
 - Yanlış veya eksik işi yapan mükemmel yazılım değil, doğru işi yapan iyi çözüm gereklidir.
 - SONUÇ: Gereksinim mühendisliği (Requirements Engineering) gerekli bir etkinliktir.



GEREKSİNİM MÜHENDİSLİĞİNE GİRİŞ

GEREKSİNİM MÜHENDİSLİĞİ

- Gereksinimler, SRS (Software Requirements Specification) belgesi altında toplanır.
- SRS raporuna yönelik çeşitli standartlar mevcuttur.
 - IEEE 830-1998: SRS odaklı
 - ISO/IEC/IEEE 29148:2011 Systems and Software Engineering – Life Cycle Processes – Requirements Engineering
 - Ders kapsamında, gereksinim mühendisliği ve analiz bölümlerinde bu belgede yer alabilecek artefact'ların en yaygınları üzerinde çalışacağız.
 - Yazılım geliştirme sırasında kod dışında ortaya çıkardığımız her türlü metin ve şemaya "artefact" denilmektedir.



GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Gereksinim mühendisliğinin genel adımları:
 - Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
- Gereksinim mühendisliği adımları gerçekleştirilecek yazılımın doğasına ve kullanılan sürece göre düzenlenmelidir.
- Gereksinim mühendisliği adımları süresince yazılım ekibi ve müşteri birlikte çalışmalıdır.
 - Müşterinin bir ekibinin, yazılım geliştirme sürecinin mümkün olduğunca çok adımının bir parçası olması yararlıdır.

GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
-
- Başlangıç:
 - Yazılım projesinin **ilk aşamalarının** başlatılıp başlatılmamasına karar verilen adımdır.
 - Müşterinin bir yazılım projesi başlatılmasını düşünmesine neden olan olaylar:
 - Yeni bir iş gereksiniminin belirlenmesi.
 - Mevcut iş süreçlerinde güçlüklerle karşılaşılması.
 - Müşterinin üst düzey karar vericileri ve astları arasında geçen kısa bir sözlü konuşma veya toplantı ile bile bir proje başlayabilir.
 - Bir uygulama yazılımı söz konusu ise:
 - Yeni bir pazarın veya hizmetin farkına varılması,
 - Yazılım şirketinin üst düzey karar vericileri ve teknik ekibinin sözlü konuşması ile yeni bir yazılım projesi başlatılabilir.



GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
-
- Başlangıç aşamasında paydaşlar belirlenmelidir.
 - Paydaş: Gerçeklenecek sistemden doğrudan veya dolaylı olarak yararlanabilecek ve etkilenebilecek herkes.
 - Her paydaş sisteme farklı bir açıdan bakar.
 - Projenin başarısı veya başarısızlığı paydaşları farklı şekillerde etkiler.
 - Paydaşlara sorulacak sorularla belirlenmesi gerekenler:
 - Paydaşların bakış açıları,
 - Paydaşları etkileyebilecek nedenler,
 - Söz konusu etkilerin sonuçları.

GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
-
- Bilgi toplama aşamasının genel ilkeleri:
 - Gereksinimler hakkında ayrıntılı bilgiler, tüm paydaşların etkin katılımı ile elde edilmelidir.
 - Tüm paydaşların katıldığı toplantılar yapılmalıdır.
 - Toplantılara hazırlık ve katılım kuralları belirlenmelidir.
 - Gündem belirlenmelidir: Önemli konuları atlamayacak kadar sıkı, yaratıcılığı önlemeyecek kadar açık olmalıdır.
 - Düzeni sağlayacak ve tikanıklıkları çözecek bir oturum başkanı seçilir.
 - Her ne kadar bu deneyimle keskinleştirilecek bir sanat olsa da, yararlı kaynaklardan ön çalışma yapılabilir.
 - Örneğin: H.M. Robert, et.al., "Robert's Rules of Order - Newly Revised", Hachette Book Group, Sep. 2020 (12th ed).

GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
-
- İşleme:
 - Bilgi toplama aşamasında toplanan ‘ham’ bilgilerin ‘işlenmesi’.
 - Son kullanıcının ve diğer paydaşların yazılımla nasıl etkileşimde bulunacağını belirlenmesi ve ayrıntılandırılmasını amaçlar.
 - Etkileşimler, kullanım senaryoları ile gösterilir (ileride anlatılacak).
 - İşleme kimi bilgilerin genişletilmesi, kimi bilgilerin özetlenmesi şeklinde gerçekleşir.
 - Gereksinimlerin sınıflandırılması
 - Normal gereksinimler
 - Beklenen gereksinimler: Çok temel gereksinimleri kullanıcı belirtmeyebilir. Bunların da elde edilmesi gereklidir.
 - Heveslendirici gereksinimler: Müşteri beklentilerinin ötesinde ve varlığında müşteriye sevindirecek özellikler.



GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
-
- Pazarlık:
 - Müşteriler sınırlı insan, zaman ve bütçe kaynakları çerçevesinde karşılanamayacak aşırı isteklerde bulunabilir.
 - Paydaşlar gereksinimleri farklı önem düzeylerinde görebilir.
 - Farklı paydaşların gereksinimleri birbiri ile çelişebilir.
 - Pazarlık sonucunda tüm paydaşların razı olacağı bir gereksinimler listesi elde edilir.



GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
-
- Tanımlama:
 - Gereksinimler tanımlama aşamasında, pazarlık sonucu üzerinde uzlaşılan haliyle kağıda dökülür.
 - Tanımlama araçları:
 - Konuşma dili ile yazılmış belgeler
 - Kullanıcı senaryoları: Görülecek
 - Kullanım şemaları: Görülecek
 - Formel modeller (Matematiksel gösterim, işlenilmeyecek)
 - Bir ön ürün
 - Birden fazla tanımlama aracı birlikte kullanılabilir.



GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
-
- Doğrulama:
 - Tanımlanmış gereksinimlerin tutarsızlıklara karşı sağlaması yapılır.
 - Gereksinimler açıkça ve yoruma yer bırakmayacak şekilde tanımlanmış mı?
 - Birbiri ile çelişen gereksinimler var mı?
 - Gereksinimlerde hatalar ve eksikler var mı?
 - Eksik gereksinimler var mı?
 - Gerçekçi olmayan gereksinimler var mı?
 - ...
 - Doğrulama yapma için önerilen temel yol teknik değerlendirmedir (Formal technical review, sınama teknikleri arasında anlatılacak).



GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
 - Bilgi Toplama (Elicitation)
 - İşleme (Elaboration)
 - Pazarlık (Negotiation)
 - Tanımlama (Specification)
 - Doğrulama (Validation)
 - Yönetim (Management)
-
- Yönetim:
 - Yazılım geliştirme süreci içerisinde gereksinimlerde değişiklikler olabilir:
 - Yeni gereksinimler eklenmesi
 - Mevcut gereksinimlerden bazılarının geçerliliğini yitirmesi
 - Gereksinimlerin önem sıralamasının değişmesi
 - Hatalı kestirimlerden dolayı bazı gereksinimlerden vazgeçilmesi
 - Gereksinimlerde ne tür değişikliklerin nasıl ve hangi şartlarla yapılabileceği, resmi bir sözleşme ile önceden belirlenebilir.
 - Gereksinimlerde değişiklikler müşteri ile karşılıklı anlaşma ile yapılmalıdır.

GEREKSİNİM MÜHENDİSLİĞİ

GEREKSİNİM MÜHENDİSLİĞİ ADIMLARI

- Başlangıç (Inception)
- Bilgi Toplama (Elicitation)
- İşleme (Elaboration)
- Pazarlık (Negotiation)
- Tanımlama (Specification)
- Doğrulama (Validation)
- Yönetim (Management)
- Yönetim (devam):
 - Yazılım geliştirme süreci içerisinde gereksinimlerin gerçekleşmesinin (ve varsa gereksinimlerdeki değişikliklerin) izlenmesi gerekir.
 - İzleme tablolar aracılığı ile yapılır:
 - İzlenebilirlik tabloları (Traceability table).

	B1	B2	B3	...
G1	✓		✓	
G2		✓		
G3	✓			
...				

- G1,2,...: Gereksinimler
- B1,2,...: Sisteme çeşitli bakış açıları
 - Modüller, Paketler, Sınıflar, vb.



- Bu yansı ders notlarının sayfa düzeni için boş bırakılmıştır.



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr. Mehmet Sıddık Aktaş

NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ



NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

NESNEYE YÖNELİK ÇÖZÜMLEMENİN TEMELLERİ

- Çözümleme (Analiz): Bir şeyi anlayabilmek için parçalarına ayırmak.
- Sistemi anlamaya yönelik çalışmalardan ve üst düzey planlama eylemlerinden oluşur.
 - Uygulama/problem alanının anlaşılması.
 - Kullanıcı gereksinimlerinin anlaşılması.
 - Koddaki sınıflar ve nesneler ile bunların arasındaki üst düzey etkileşimlerin belirlenmesi: Çözümleme modelinin oluşturulması.
- “Bir sorunu anlamadan çözemezsiniz.”



NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

UYGULAMA ALANININ ÇÖZÜMLENMESİ (DOMAIN ANALYSIS)

- Amaç, uygulama alanını anlamak ve elde edilen bilgileri analiz modeline taşımaktır.
- Uygulama alanı hakkında bilgi edinilebilecek kaynaklar:
 - Teknik literatür
 - Mevcut uygulamalar
 - Müşteri anketleri
 - Uzman tavsiyeleri
 - Mevcut ve gelecekteki gereksinimler
- Problem alanı hakkında bilgi edinmeden “müşterinin dilinden konuşamazsınız”.

NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

GEREKSİNİMLERİN BELİRLENMESİ

- Gereksinimler belgesi:
 - Müşterinin programdan beklentilerini anlatan, doğal konuşma dili ile yazılmış belge.
- Örnek gereksinimler belgesi:

NextGenPOS Perakende Satış Programı

Eski yazılım ihtiyaçlarımızı karşılayamadığından, yenilenecek donanım ile birlikte perakende satış programımızın da yenilenmesine gerek duyuyoruz. Program kasada yapılan alış-veriş işlemlerine yardımcı olmalıdır. Yapılan her işlem program tarafından saklanmalı; mali bilgiler harici bütçe sistemine, mal çıkış bilgileri ise harici envanter sistemine iletilmelidir. Saklanan işlemler üzerinde daha sonra raporlamalar ve analizler yapılabilmelidir. Sistem yapılan alış-verişler karşılığında müşteriye fiş vermelidir. Yapılan her satış için KDV de hesaplanarak ayrıca belirtilmelidir. Şirketimizin birden fazla şubesi olup tüm şubelerdeki işlemler merkezi sunucuya iletilmelidir.

- Doğal dille yazılmış gereksinimler belgesinden kullanım öykülerine geçiş yapılır.



NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

GEREKSİNİMLERİN BELİRLENMESİ

- Kullanım öyküleri:
 - Programın yapacağı işleri ayrıntılı adımlarla ve belli kurallara uyarak anlatan belgeler.
- Kullanım öykülerinin oluşturulmasındaki amaç:
 - Ürünün sağlaması beklenen işlevleri ve ürünün çalışma ortamını belirlemek,
 - Son kullanıcı ve yazılım ekibi arasında bir anlaşma zemini belirlemek,
 - Son kullanıcı ve sistemin birbirleri ile nasıl etkileşimde bulunacağını açık ve belirsizlikten uzak olarak tanımlamak,
 - Doğrulama testleri için bir zemin oluşturmak.
- Bir kullanım öyküsünün bölümleri:
 - Giriş bölümü: Sistemin neyi hangi koşullar ve sınırlar içerisinde yapması gerektiğini anlatır.
 - Ana senaryo / Ana başarılı akış: Her şeyin yolunda gitmesi halinde yürütülecek eylemler.
 - Alternatif senaryolar: Bir aksilik olması halinde yapılacak işlemler.

NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

KULLANIM ÖYKÜSÜ: Satış İşlemi

Birincil Aktör: Kasiyer.

İlgililer ve İlgili Alanları:

- Kasiyer: Doğru ve hızlı giriş ister, kasa açığı maaşından kesildiğinden ödeme hataları istemez
- Satıcı: Satış komisyonlarının güncellenmesini ister
- Müşteri: En az çaba ile hızlı hizmet ister. Ürün iadesinde kullanmak üzere fiş ister.
- ...

Ön Koşullar:

- Kasiyerin kimliği doğrulanır.

Son Koşullar:

- Ödeme tahsil edilir. Satış kaydedilir. Fiş yazılır.
- Dikkat: Kullanım öyküsünde yer alacak her şey, verilen ilgi alanlarına giren şeyler olmalıdır.
- Aktör: Sistem ile etkileşimde bulunan varlıklar.
 - İnsan
 - Yazılım veya donanım.

NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

KULLANIM ÖYKÜSÜ: Satış İşlemi

Ana Öykü:

1. Müşteri kasaya alacağı ürünlerle gelir.
2. Kasiyer yeni bir satış işlemi başlatır.
3. Kasiyer ürünün barkodunu girer.
4. Sistem bir satış kanalı maddesi oluşturur. Bu maddede ürün tanımı, fiyatı ve toplam bedel (aynı maldan birden fazla alınmış olabilir) yer alır.
5. Kasiyer 3. ve 4. adımları müşterinin alacağı tüm ürünler için tekrarlar.
6. Sistem toplam bedeli KDV ile birlikte hesaplar.
7. Kasiyer müşteriye toplamı bildirir ve ödeme ister.
8. Müşteri ödemeyi yapar ve sistem ödemeyi tahsil eder.
9. Sistem tamamlanan işlemin kaydını tutmayı tamamlar ve harici envanter ile mali sistemlere gerekli bilgileri gönderir.
10. Sistem fiş verir.
11. Müşteri ürünlerle birlikte ayrılır.

NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

KULLANIM ÖYKÜSÜ: Satış İşlemi

Alternatif Öyküler:

3a. Geçersiz barkod

1. Sistem uyarı mesajı verir ve kayıt girişini reddeder.

3-7a. Müşteri bir kalem malı alışverişten çıkartmak ister.

1. Kasiyer satıştan çıkarmak üzere ürünün barkodunu okutur.
2. Sistem güncel toplamı bildirir.

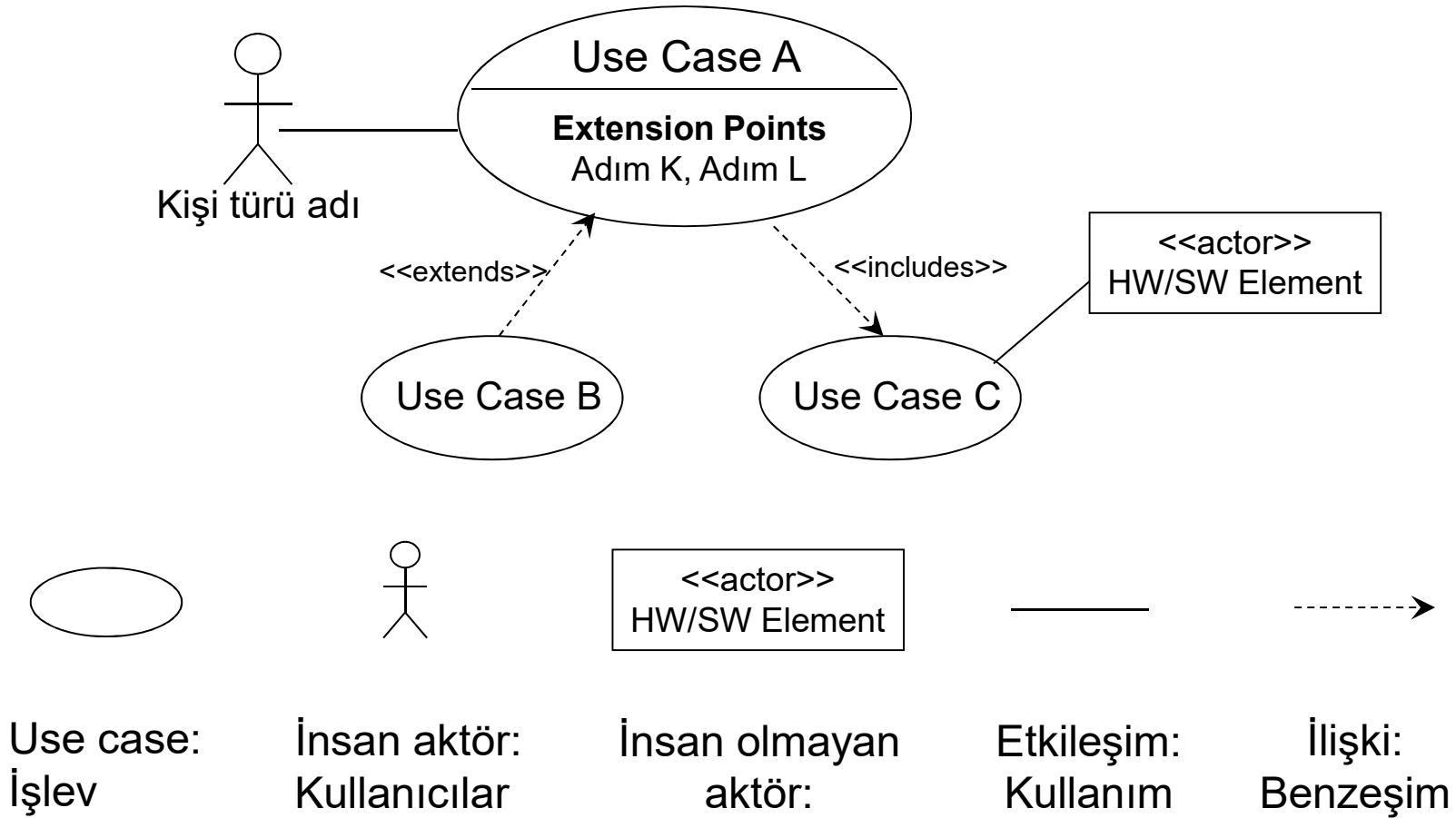
...

KULLANIM ÖYKÜLERİNİN GRAFİK GÖSTERİMİ

- Kullanım öyküleri, ayrıntılı ve uzun belgelerdir.
- Yazılımın yapacağı işlerin özet gösterimi için kullanım şemaları çizilir (use-case diagrams).
- Çizim kurallarını verdikten sonra örnek öykünün şemasını çiz.

KULLANIM ŞEMALARI – USE CASE SCHEMAS

ÇİZİM KURALLARI





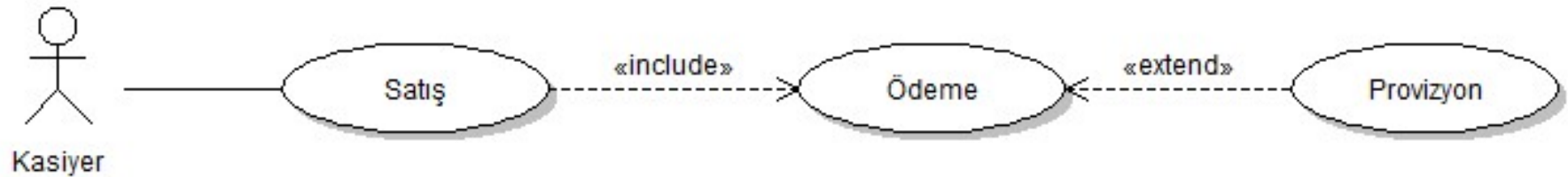
KULLANIM ŞEMALARI – USE CASE SCHEMAS

ÇİZİM BİLGİLERİ

- Benzeşim ilişkileri:
 - Ok yönü aynı zamanda ilişkiyi okuma yönüdür.
 - UC-B extends UC-A : B işlevi, A işlevi yürütülürken oluşabilecek bir sapış anlamındadır.
 - A: Ana akış
 - B: Ana akıştaki bir seçenek, ana akıştan bir sapış, alt akış
 - UC-A includes UC-C: A işlevi, C işlevini içerir.
 - A : Ana akış, içeren akış
 - C: Alt akış, içerilen akış

KULLANIM ŞEMALARI – USE CASE SCHEMAS

ÖRNEK ÇİZİM



- Bir POS yazılımının ödeme işlevini kasiyer kullanır.
- Satış işlevi, içerisinde ödeme yapma işlevini içerir.
 - Includes, çünkü: Her satış içerisinde mutlaka ödeme olur.
- Ödemenin kredi kartı ile olması halinde, provizyon alma işlemi yürütülür.
 - Extends, çünkü: Ödeme nakit ise provizyona gerek kalmaz.
 - Provizyon: Kredi kartının limitinin aşılıp aşılmadığı, çalıntı olup olmadığı, vb. gibi bilgilerin sınanması anlamında bir bankacılık terimi.



NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

SINIFLARIN BELİRLENMESİ

- Kullanıcı gereksinimleri belgesinden ve kullanım senaryolarından sınıfların elde edilmesi.
 - İsimlerin taranarak aday sınıfların elde edilmesi.
 - Adaylar aşağıdaki kurallardan birini sağlamalıdır:
 1. Saklanan bilgi: Sistemin çalışması süresince bu varlığın durumu saklanmalıdır.
 2. Gereksinim duyulan hizmetler: Bu varlığın hizmetlerine ihtiyaç duyan başka varlıklar vardır.
 3. Gerekli varlıklar: Problemin çözümü ile ilgili bilgi üreten veya problemin çözümü için bilgi tüketen varlıklar.
- Değinilen kurallardan birini sağlayamayan adayları, bir başka sınıfın üye alanı olarak değerlendirebiliriz.
- Örnek gereksinim belgesinden (metin ve senaryolardan) sınıfları oluştur.



NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

SINIFLARIN BELİRLENMESİ

- Üyelerin belirlenmesi:
 - Sıfat ve eylemlerin taranması
 - Sorumlulukların belirlenmesi (CRC kartları)
- Sorumlulukların dağıtılması:
 - Sorumlulukların bir yerde yoğunlaşmaması
 - Sorumlulukların genelden özele doğru tanımlanması (kalıtım hiyerarşisinde genelden özele gidilmesi)
 - Bir bilgi ile ilgili davranışların, o bilgi ile aynı sınıfta yer alması (encapsulation)
 - Tek bir şey hakkındaki bilginin tek sınıfta yer alması
 - Gerekli sorumlulukların paylaşılması



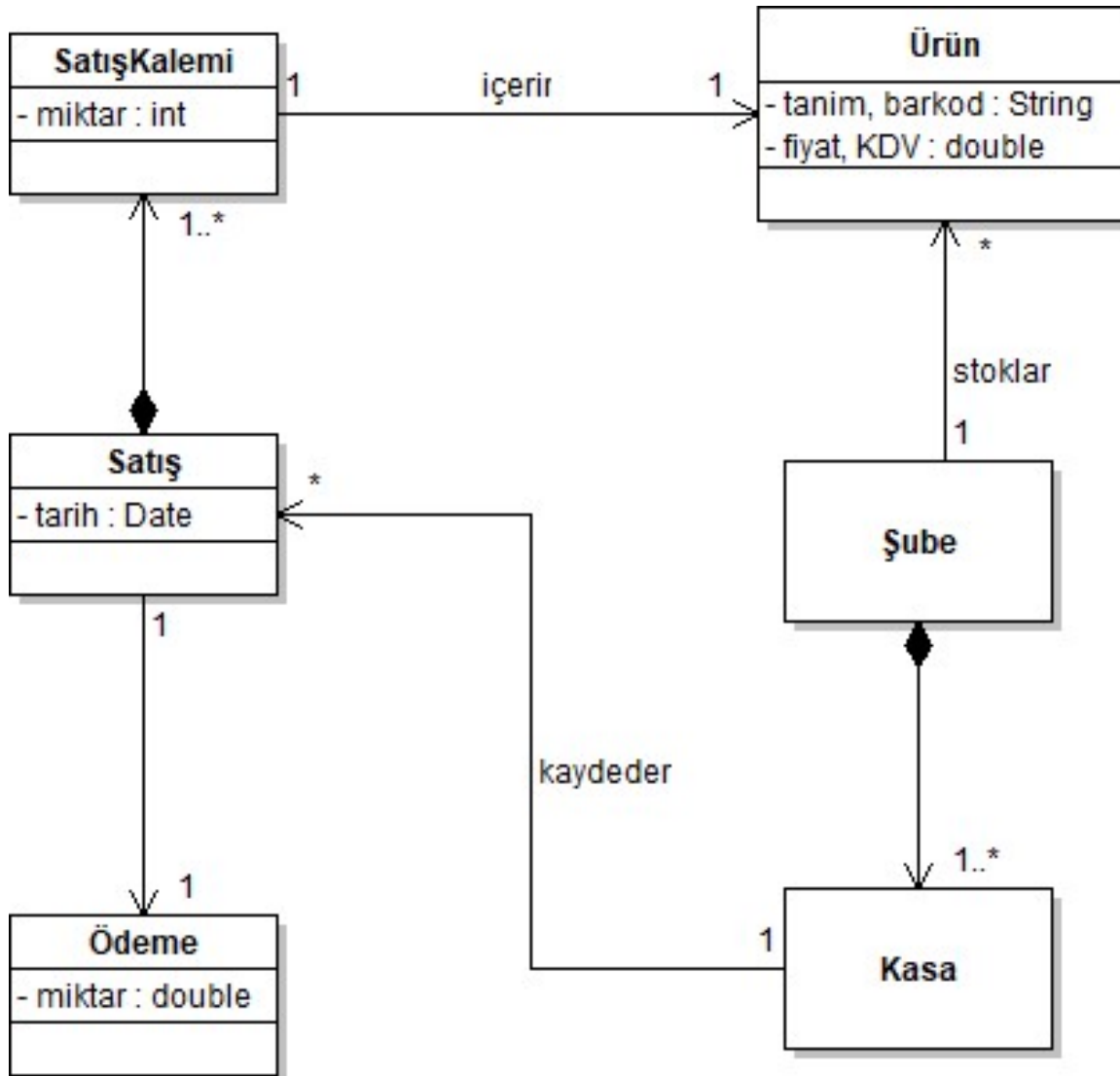
NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

ETKİLEŞİMLERİN BELİRLENMESİ

- Etkileşim: Bir nesnenin üzerine düşen sorumluluğu yerine getirmek için diğer bir nesneye mesaj göndermesi.
- Nesneler arasındaki ilişkiler
 - Bağlantı, toplama, meydana gelme.
- Sınıflar arasındaki ilişkiler
 - Özelleşme/genelleşme
- Çözümleme aşamasında ne tür etkileşimlerin olabileceği düşünülür, etkileşimlerin nasıl olacağı düşünülmez.
- Bu konuların temeli "Nesneye Dayalı Kavramlar" dersinde atılmıştır.

NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

ÖRNEK ALAN MODELİ





NESNEYE YÖNELİK ÇÖZÜMLEME SÜRECİ

ÇÖZÜMLEME SÜRECİNİN BELGELENDİRİLMESİ

- Bir nesneye yönelimli programın çözümleme sürecinin belgelendirilmesinde yer alan önemli belgeler:
 - UML Kullanım şemaları,
 - Kullanım senaryoları,
 - UML sınıf şemaları,
- Veritabanı işlemleri yapılacaksa bunlara ek olarak:
 - E-R diyagramı



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr. Mehmet Sıddık Aktaş

NESNEYE YÖNELİK TASARIM SÜRECİ



NESNEYE YÖNELİK TASARIM SÜRECİ

ANALİZDEN TASARIMA GEÇİŞ

- Tasarım sırasında çizdiğimiz çeşitli şemalar ve hazırladığımız sözleşmeler, analiz sırasında oluşturduğumuz çeşitli şemalar ve metinleri ayrıntılandırır ve/veya değiştirir.
 - Yazılım geliştirme sırasında kod dışında ortaya çıkardığımız her türlü metin ve şemaya "artefact" denilmektedir.



NESNEYE YÖNELİK TASARIM SÜRECİ

GİRİŞ

- Nasıl? sorusuna yanıt aranır.
- Nesne modeli: Analizden tasarıma.
 - Doğrudan problem alanı ile ilgili nesnelerden oluşan model, yardımcı nesnelerle zenginleştirilir.
- Ana işlem grupları:
 - Nesne tasarımı: Problem alanı ile ilgili nesneler
 - Sistem tasarımı: Alt yapıyı ve gereçleri oluşturan nesneler
- Sistem katmanında bulunabilecek bileşenler:
 - Yazılım mimarisi: İstemci - sunucu, eşler arası, olay tabanlı, vb.
 - Kullanıcı arayüzü
 - Veri yönetimi
 - Ağ programlama
- Sistem katmanını çoğunlukla kendimiz sıfırdan oluşturmayız, hazır altyapı programlarını (framework) kullanırız.



NESNEYE YÖNELİK TASARIM SÜRECİ

TASARIM ÖLÇÜTLERİ

- Tasarım ölçütleri:
 - Ayrılabilirlik: Anlamlı parçalara ayrılabilme.
 - Parça: Sınıf/sınıf grubu.
 - Üstünde çalıştığımız problem hangi düzeyde alt problemlere bölünebiliyorsa, tasarımıımız da aynı düzeyde ayrıştırılabilmelidir.
 - Birleştirilebilirlik: Bir parçanın başka tasarımlarda da kullanılabilecek şekilde diğer parçalarla birleştirilebilmesi.
 - Anlaşılabilirlik: Bir parçanın diğer parçalar hakkında bilgiye gerek duyulmadan anlaşılabilmesi.
 - Süreklilik: Yapılacak küçük değişikliklerin etkilerinin en az sayıda parçaya yayılması (tercihen tek sınıfa).
 - Koruma: Olası hataların düzeltilmesine yönelik büyük değişikliklerin etkilerinin geniş bir alana yayılmasının önlenmesi.



NESNEYE YÖNELİK TASARIM SÜRECİ

TASARIM İLKELERİ

- İyi bir tasarıma götüren iki temel ilke:
 - Düşük bağlaşım (Low coupling)
 - Yüksek uyum (High cohesion)
- Bu ilkeler hem birbirlerine hem de uygulama alanına bağımlıdır.
- Başka ilkeler de öne sürülebilir, ancak bu ikisi en temel ölçütlerdir.



NESNEYE YÖNELİK TASARIM SÜRECİ

DÜŞÜK BAĞLAŞIM – LOW COUPLING

- Bağlaşım: Bir parçanın diğer parçalara bağımlılık oranı.
 - Parça: Sınıf, alt sistem, paket
- Bağımlılık: Bir sınıfın diğerinin:
 - Hizmetlerinden yararlanması,
 - İç yapısından haberdar olması,
 - Çalışma prensiplerinden haberdar olması,
 - Özelleşmiş veya genelleşmiş hali olması (kalıtım ilişkisi).
- Çeşitli sınıf şemaları ile bağlaşım soruları sor.
 - İlişkide bulunan diğer sınıfların sayısı arttıkça bağlaşım oranı artar.
- Düşük bağlaşımın yararları:
 - Bir sınıfta yapılan değişikliğin geri kalan sınıfların daha azını etkilemesi,
 - Yeniden kullanılabilirliğin artması



NESNEYE YÖNELİK TASARIM SÜRECİ

YÜKSEK UYUM – HIGH COHESION

- Uyum: Bir parçanın sorumluluklarının birbirleri ile uyumlu olma oranı.
- Yüksek uyumun yararları:
 - Sınıfın anlaşılma kolaylığı artar.
 - Yeniden kullanılabilirlik artar.
 - Bakım kolaylığı artar
 - Sınıfın değişikliklerden etkilenme olasılığı düşer.
- Genellikle:
 - Düşük bağlaşım getiren bir tasarım yüksek uyumu,
 - Yüksek bağlaşım getiren bir tasarım ise düşük uyumu beraberinde getirir.

TASARIM KALIPLARI (Design Patterns)

- Modelleme alanında sık karşılaşılan problemlere çözüm önerileri sunarlar.
- Tasarım kalıplarına uyarak daha esnek bir tasarım elde edebiliriz.
- Design Patterns – Elements of Reusable OO Software, Erich Gamma et.al (Gang of Four), Addison-Wesley, 1994
 - Ek kaynaklardan da çalışılması yararlı olacaktır.

TASARIM KALIPLARINA GİRİŞ

TASARIM KALIBI (DESIGN PATTERN) NEDİR?

- Yazılım tasarımımda karşılaşılan belirli sorunların yalın ve güzel çözümlerinin tarifidir.
 - Çözüm somut bir gerçekleştirme olmak zorunda değildir, soyut bir tarif de olabilir.
- Amaç: Tekerleği yeniden keşfetmeden 'iyi' tasarıma ulaşmak.
 - Sorunların esnek biçimde ve yeniden kullanılabilirliği arttıracak yönde çözülmesi.
- Bir tasarım kalıbının ana bileşenleri:
 - İsim: Kalıbı en güzel şekilde anlatacak bir veya birkaç kelimelik isim.
 - Sorun: Kalıbın çözdüğü sorunun tanımı.
 - Çözüm: Sorunu çözmek için önerilen tasarımın bileşenleri; bu bileşenlerin ilişkileri, sorumlulukları, birlikte çalışmaları.
 - Tartışma/Sonuçlar: Çözümün sistemin geneline esneklik, genişletilebilirlik, taşınabilirlik gibi yönlerden yaptığı etkiler; çözümün güçlü ve zayıf yönleri, yer-zaman çelişkileri, başka kalıplar ile işbirliği olanakları, vb.



TASARIM KALIPLARINA GİRİŞ

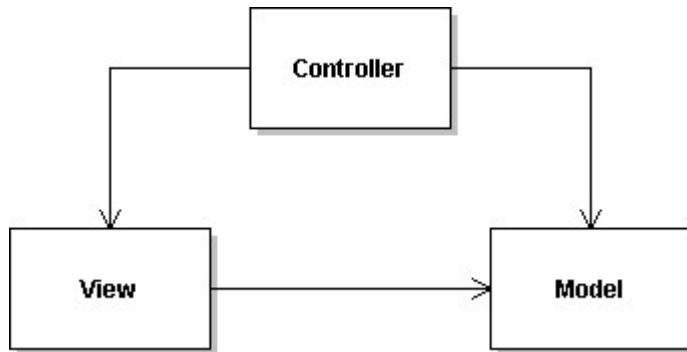
TASARIM KALIBI (DESIGN PATTERN) NEDİR?

- Yararları:
 - Tekerleği yeniden icat etmemek.
 - Üzerinde birçok deneyimli kişinin emeği geçtiğinden, 'kaliteli' bir çözüm.
 - Deneyimin yazılı biçimde aktarılabilmesi.
 - Yazılım ekibi arasında ortak bir sözlük oluşturması.
- Tartışma: Bir kişinin 'kalıp' dediğine, diğeri 'basit bir yapı taşı' veya 'temel bir ilke' diyebilir.
- Tasarım kalıplarının bilgisayar bilimlerinde popülerlik kazanması "Dörtlü Çete – GoF (Gang of Four)" lakaplı yazarların kitabı sayesinde:
 - Design Patterns – Elements of Reusable OO Software, Erich Gamma et.al, Addison-Wesley, 1994
 - Eserde GoF kalıpları amaçlarına göre 3 kümeye ayrılmıştır:
 - Creational: Yaratımsal. Nesnelerin oluşturulması, temsili ve ilişkilendirilmelerindeki bağımlılığı azaltmaya yönelik kalıplardır.
 - Structural: Yapısal. Sınıflar ve nesnelerin bir araya getirilerek daha büyük yapılar elde edilmesine yönelik kalıplardır.
 - Behavioral: Davranışsal. Sorumlulukların nesnelere dağıtılması ve algoritma seçimine yönelik kalıplardır.

TASARIM KALIPLARINA GİRİŞ

ÖRNEK TASARIM KALIBI: MODEL-VIEW-CONTROLLER (MVC)

- Sorun:
 - Aynı veriyi farklı biçimlerde gösterebilmek istiyoruz.
 - Kullanıcıya, verinin nasıl ve ne kadarının gösterileceğini seçebilme gibi olanaklar sağlamak istiyoruz.
- Çözüm:
 - Veriyi, verinin gösterimini ve gösterimin yönetimini ayrı sınıflar şeklinde ele almak.



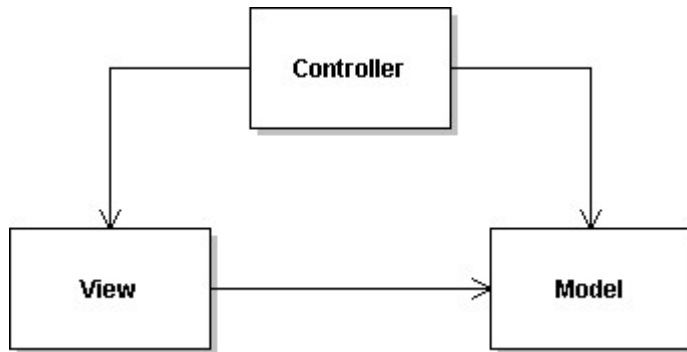
- Model sınıfı:
 - Ham veriyi simgeler.
- View sınıfı:
 - Verinin çeşitli gösterim biçimleri.
- Controller sınıfı:
 - Kullanıcı komutlarını alıp işler.

- Benzetim:
 - Bir ressam ve çizdiği insan.

TASARIM KALIPLARINA GİRİŞ

MVC TASARIM KALIBI

- Yararlar:
 - Gösterim, kontrol ve veriyi birbirlerinden ayırarak;
 - Bunları çalışma anında değiştirebilmek.
 - İlgili alanlarının ayrılmasını sağlamak.
 - Yeniden kullanımı arttırmak.



- Çeşitlemeler:
 - Model'in geçirdiği değişiklikleri View'a bildirmesi
 - Birden fazla view:
 - İç içe (nested).
 - Ayrı/tek görevcikte (thread)
 - Örtüşen/ayrık
 - Birden fazla kontrolcü:
 - Farklı denetim biçimleri (klavye, fare, ses, ...)



NESNEYE YÖNELİK TASARIM SÜRECİ

TASARIMIN BELGELENDİRİLMESİ

- Bir nesneye yönelimli programın tasarım sürecinin belgelendirilmesinde yer alan önemli belgeler:
 - Ayrıntılı UML sınıf şemaları: Vazgeçilmez.
- Tasarımın ihtiyaçlarına göre alttaki diğer belgelerin çeşitli bileşimleri de kullanılabilir:
 - Sözleşmeler
 - UML Etkileşim şemaları (Interaction diagrams)
 - Sıralama şemaları (Sequence diagrams)
 - İşbirliği şemaları (Collaboration diagrams)
 - UML Etkinlik şemaları (Activity diagrams)
 - UML Durum diyagramları (State diagrams)



NESNEYE YÖNELİK TASARIM SÜRECİ

SÖZLEŞME İLE TASARIM – DESIGN BY CONTRACT

- Sözleşme:
 - Kullanım senaryosunun ayrıntılandırılması ile elde edilir.
 - Bir nesnenin bir eylemi, bir sözleşme ile ayrıntılandırılır.
 - Her metota sözleşme yazılacak diye bir koşul yoktur.
 - Zaten kolay anlaşılabilir metotların sözleşmeye ihtiyacı yoktur.

NESNEYE YÖNELİK TASARIM SÜRECİ

SÖZLEŞME İLE TASARIM – DESIGN BY CONTRACT

- Örnek sözleşme:

Sözleşme No: 2 – Satış Kalemi Girişi

İşlem:	Kasa.ürünGir(barkod: String, adet: integer)
Çapraz Başvurular:	Satış kullanım senaryosu
Ön Koşullar:	Süregelen bir satış işleminin olması
Son Koşullar:	<ul style="list-style-type: none">- Bir SatışKalemi örneği olan satisKalemi oluşturulmuştur.- satisKalemi süregelen satis ile (Satış örneği) ilişkilendirilmiştir.- satisKalemi.miktar üyesine o malın satış adedi atanmıştır.- satisKalemi, satılan mal ile uyuşan barkod sayesinde bir Urun örneği ile ilişkilendirilmiştir.



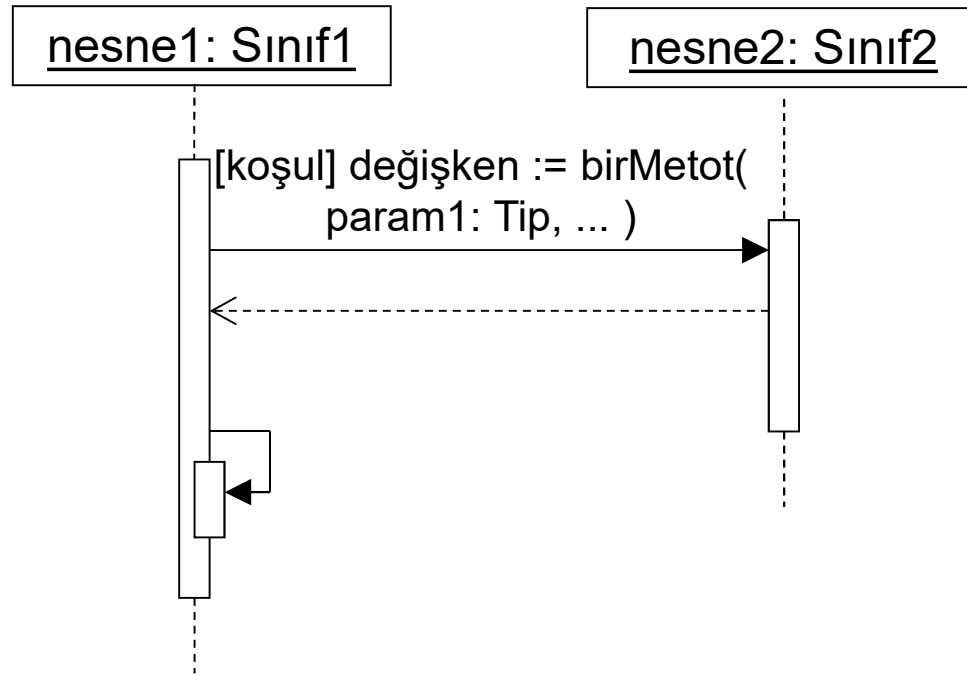
NESNEYE YÖNELİK TASARIM SÜRECİ

SÖZLEŞME İLE TASARIM – DESIGN BY CONTRACT

- Sözleşmeler hakkında bazı ayrıntılar:
 - Ön koşullar tüm sistem hakkındaki bilgilerdir.
 - Son koşullar sadece problem alanı ile ilgili nesnelerin durum değişiklikleri hakkındadır.
 - Sözleşmeler her zaman gerekli olmayabilir.
 - Son koşullarda edilgen geçmiş zaman kullanılması, bunların işlemin sonunda tamamlanmış eylemler olduğunu vurgulamak açısından yerinde olacaktır.
 - Sözleşme içerisinde ilişkilerin kurulmasını belirtmeyi unutmayın!
 - Sözleşme yazılması problem alanı çözümlemesinde güncellemelere yol açabilir.

SIRALAMA ŞEMALARI AYRINTILARI

SIRALAMA ŞEMALARI



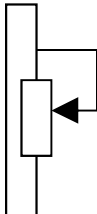
→ Mesaj (metot çağırma)

← Geri dönüş (ihmal edilebilir)

[koşul] → Koşullu mesaj (if)

* [koşul] → Döngülü mesajlar (for, while, vb.)

new → Nesne oluşturma

 Nesnenin kendi metodunu çağırması

- Bu yansı sadece bir hatırlatmadır, gerekli altyapıyı "Nesneye Dayalı Kavramlar" dersinde kazanmıştınız.



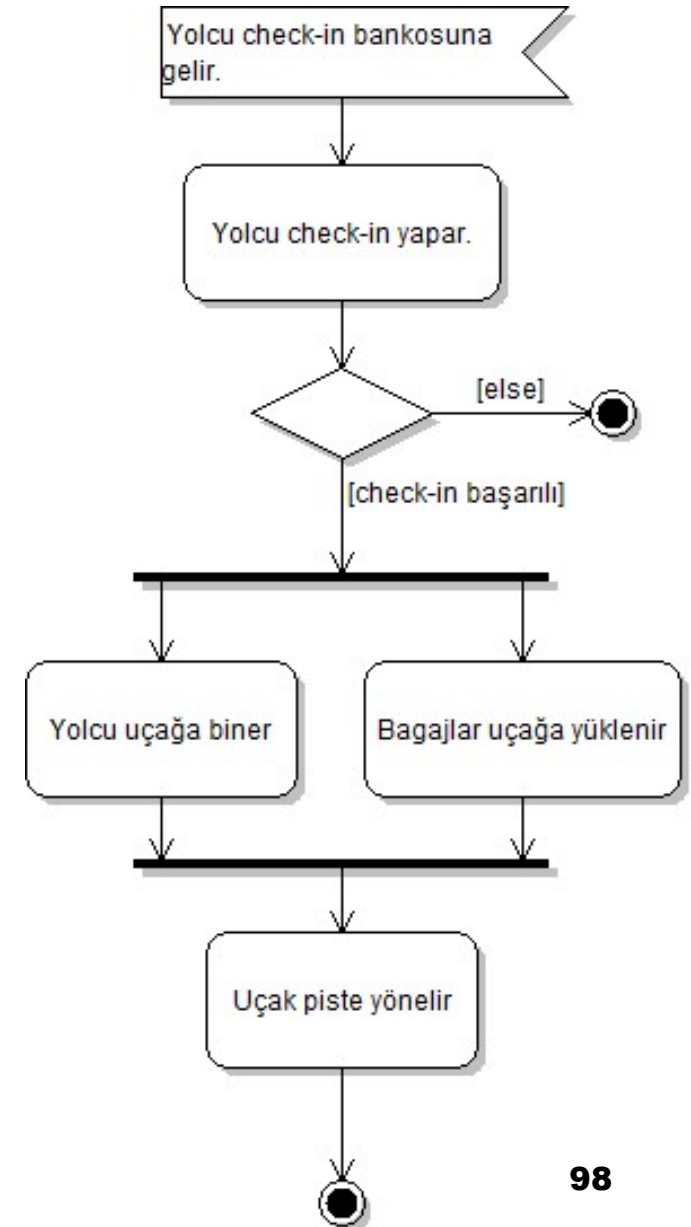
ETKİNLİK ŞEMALARI – ACTIVITY DIAGRAMS

UYGULAMA ALANLARI

- Denetim akışını olaylar üzerinden göstermeye yarar.
- İş kurallarını göstermek ve paralel çalışma (multithreaded) ayrıntılarına girmek gerektiğinde, etkileşim şemalarından daha yararlıdır.

ETKİNLİK ŞEMALARI – ACTIVITY DIAGRAMS

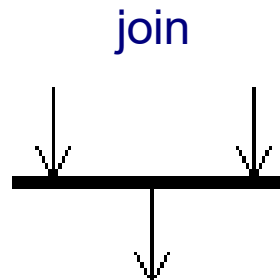
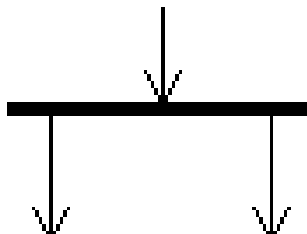
ÇİZİM KURALLARI ve ÖRNEK ÇİZİM



ETKİNLİK ŞEMALARI – ACTIVITY DIAGRAMS

ÇİZİM BİLGİLERİ

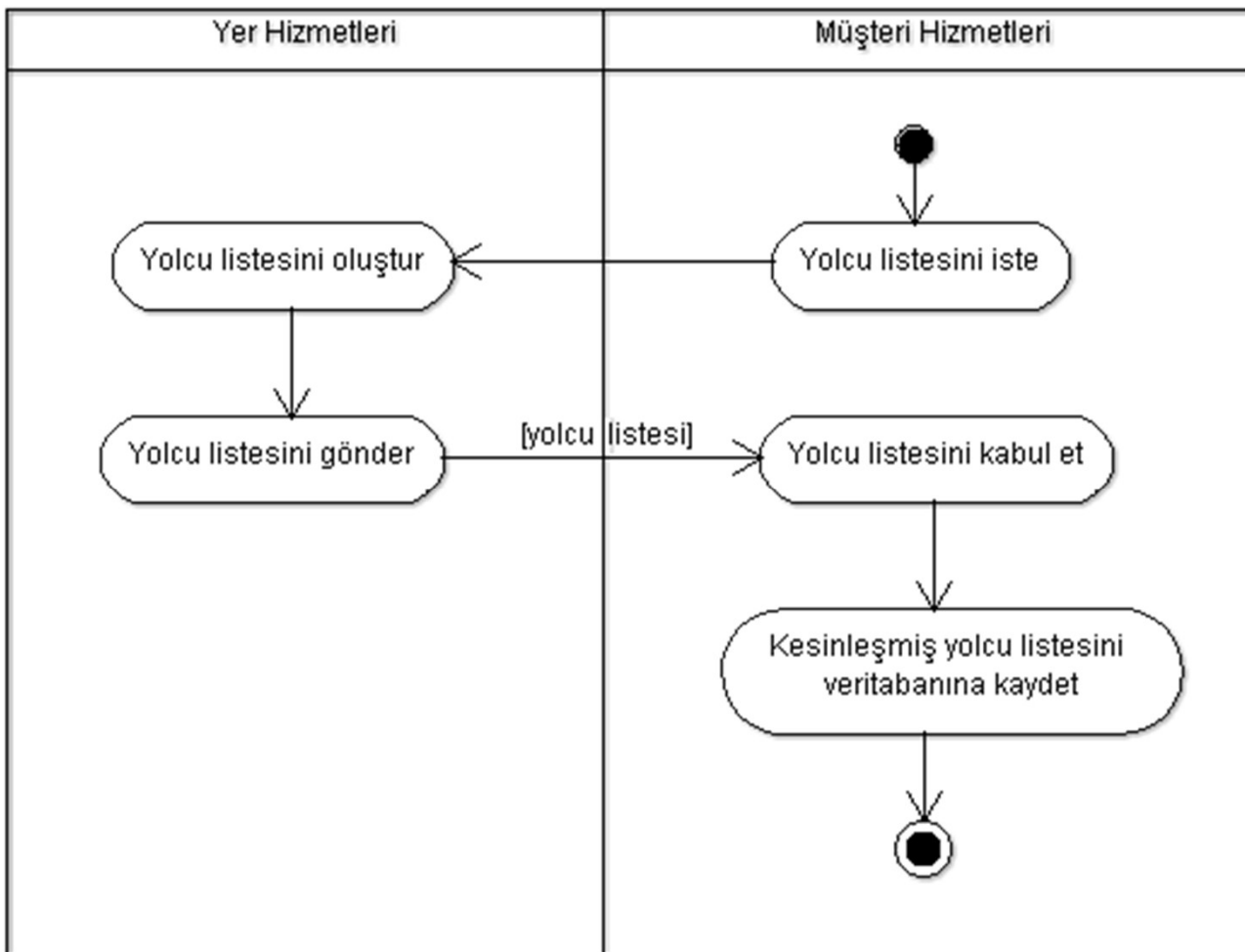
- Etkinlik şemaları başlangıç işareti veya sinyal alma işareti ile başlar.
- Sinyal alma: Beklemelidir.
 - Akış, bir sinyal alana kadar bekler.
 - Zamanlı olaylar da (timer) bununla gösterilebilir.
- Eşgüdüm: Beklemelidir.
 - Eşgüdüm çizgisine varan akış, çizgiyi geçmeden önce diğer akışların hepsini bekler.
 - fork



ETKİNLİK ŞEMALARI – ACTIVITY DIAGRAMS

ÖRNEK ÇİZİM

- Birden fazla aktörün ve aktörler arası bilgi akışının gösterilmesi:





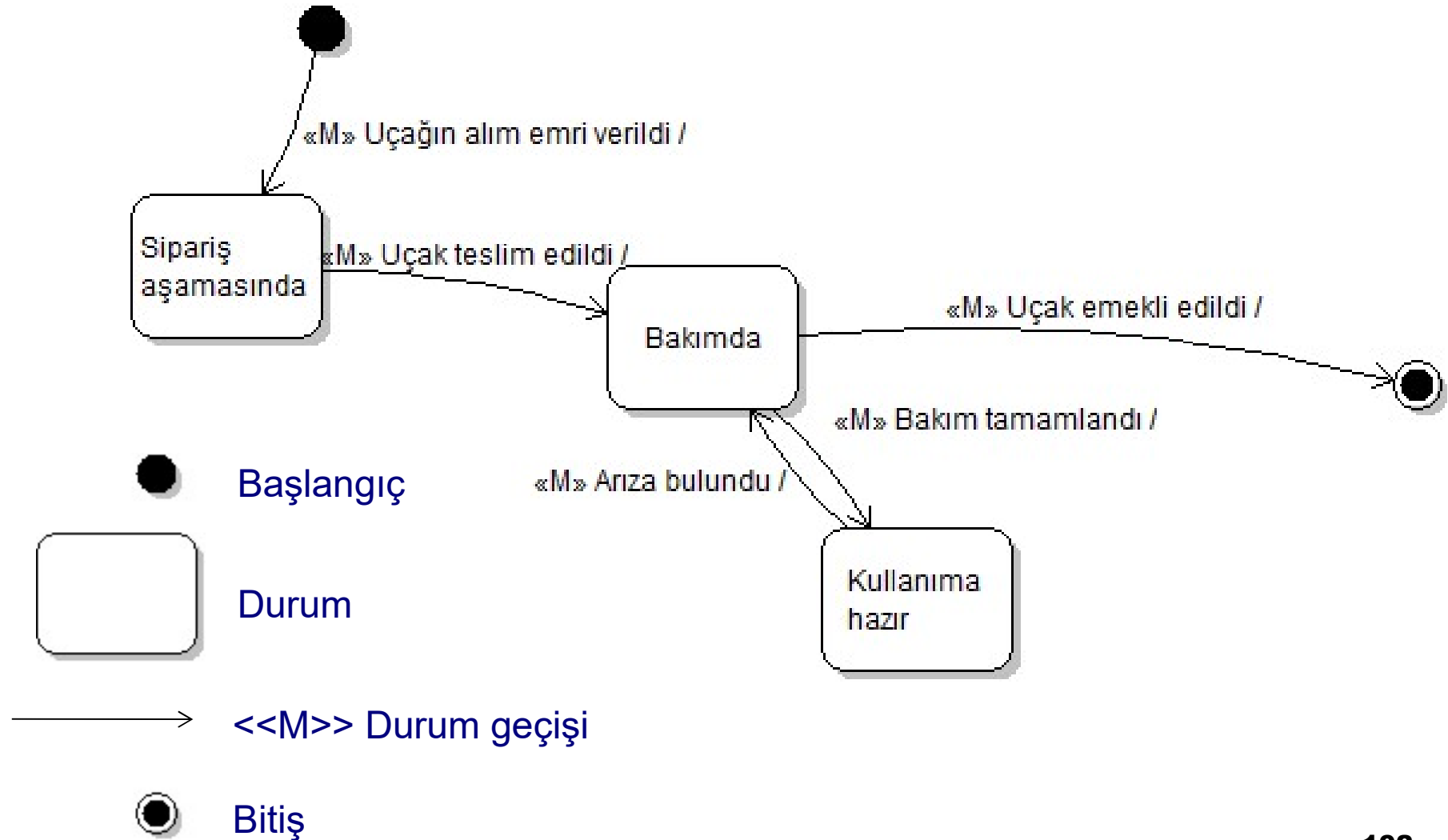
DURUM ŞEMALARI – STATE DIAGRAMS

UYGULAMA ALANLARI

- Bir varlığın içinde bulunabileceği durumları ve bu durumların birinden diğerine geçiş yapma kurallarını anlatmaya yarar.

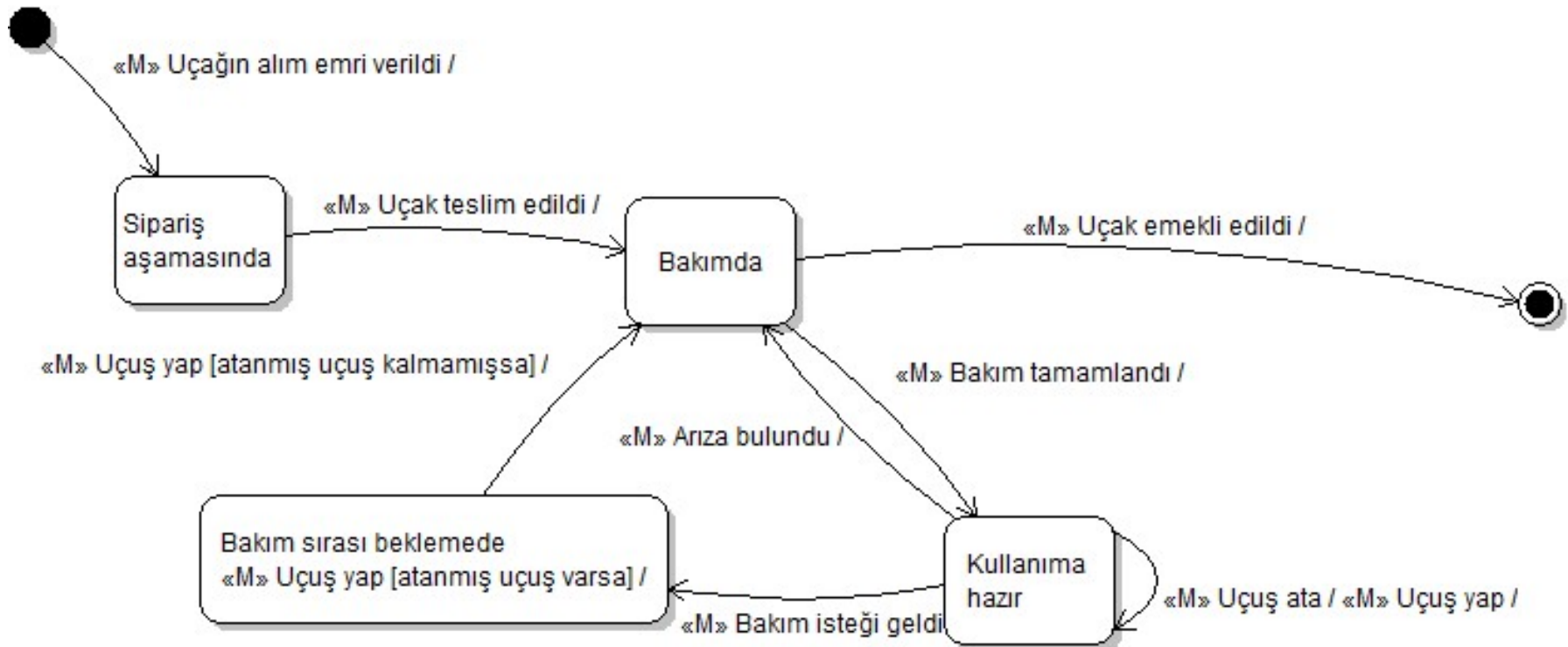
DURUM ŞEMALARI – STATE DIAGRAMS

ÇİZİM KURALLARI ve ÖRNEK ÇİZİM



DURUM ŞEMALARI – STATE DIAGRAMS

ÖRNEK ÇİZİM (2)



- Yorumlama: Kullanıma hazır bir uçak için bakım isteği gelmişse, uçak önce bakım sırasına alınır. Bu sırada önceden planlanmış uçuşları varsa onları yapar. Planlanan uçuşlar bitince uçak bakıma alınır.



DURUM ŞEMALARI – STATE DIAGRAMS

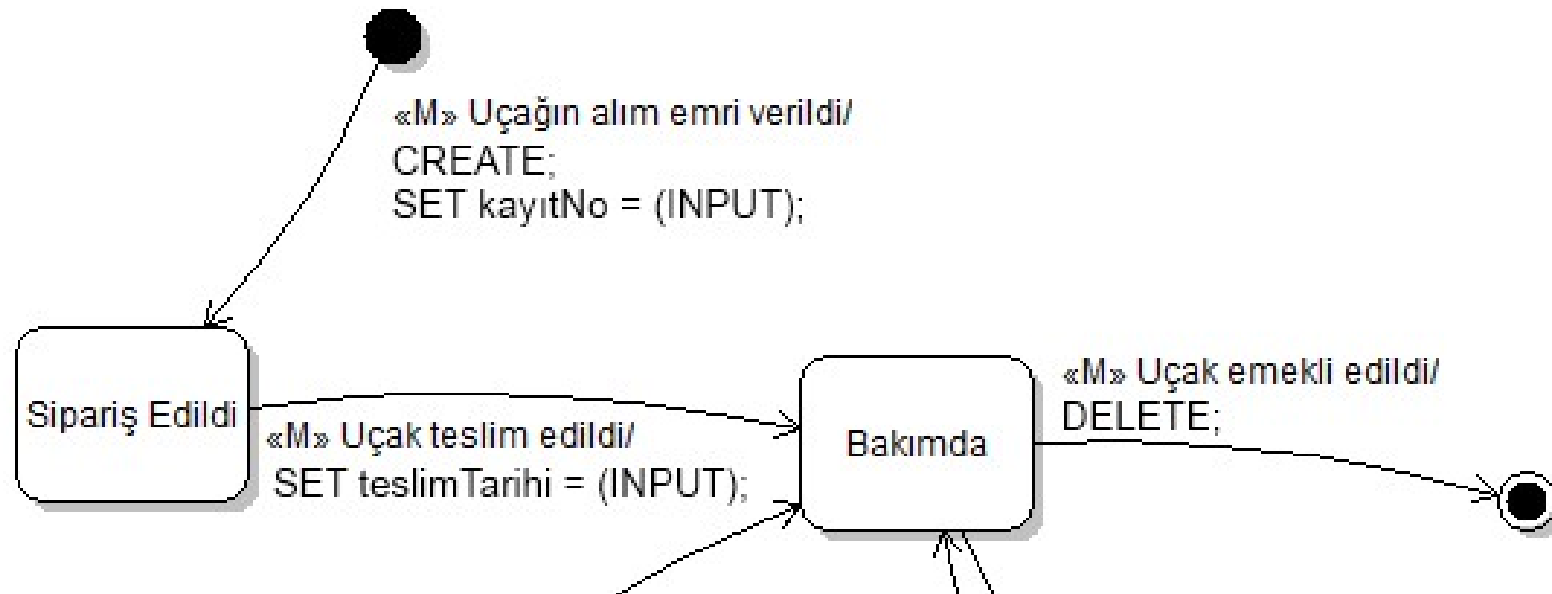
ÖRNEK ÇİZİM (2)

- Dezavantaj: Tutarlılık denetimi zor olabilir.
 - Karmaşık şemalarda mesajları takip etmek zorlaşır
 - Çünkü aynı mesaj birden fazla durumla ilişkili olabilir
 - Bu durumda aynı mesaj birden fazla yerde geçer.
 - Ör: Uçuş Yap mesajı.

DURUM ŞEMALARI – STATE DIAGRAMS

ÖRNEK ÇİZİM (3)

- Durum geçişi sırasında işlenen komutların şemada gösterilmesi:





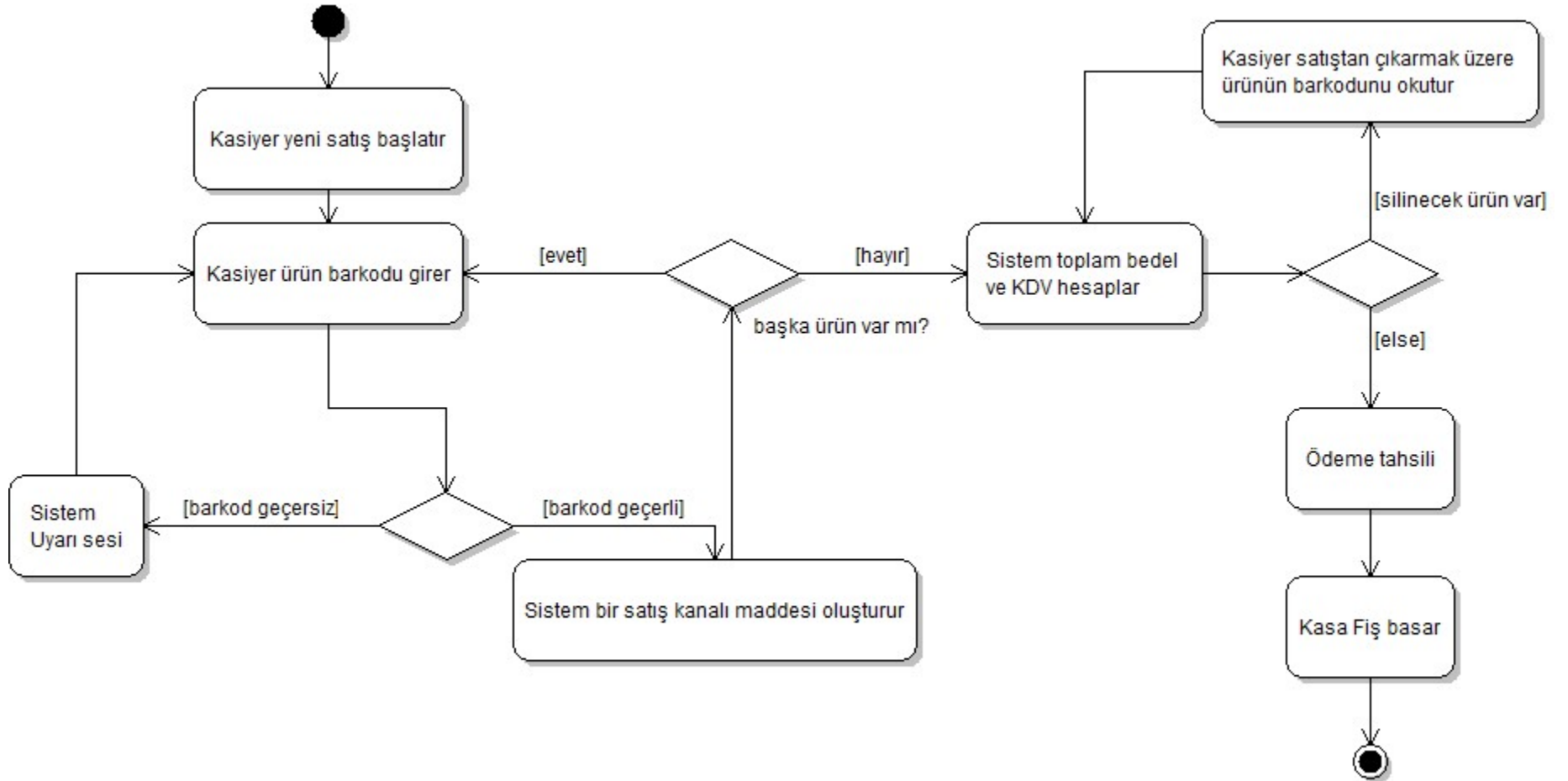
TASARIM MODELİ

NextGenPOS Tasarım Modeli

- Kullanım senaryosu metni ve alan modelinden yola çıkarak tasarım modelini oluşturalım.
 - Bu amaçla bir etkinlik, bir durum ve bir sınıf şeması çizelim.
 - Belki bu sırada keşfedeceğimiz yeni ayrıntılar olacaktır.
 - Tasarım modelindeki sınıf şemasının farkı, artık yönsüz ilişki bırakılmaması ve sınıfların metotlarının da eklenmesidir.

TASARIM MODELİ

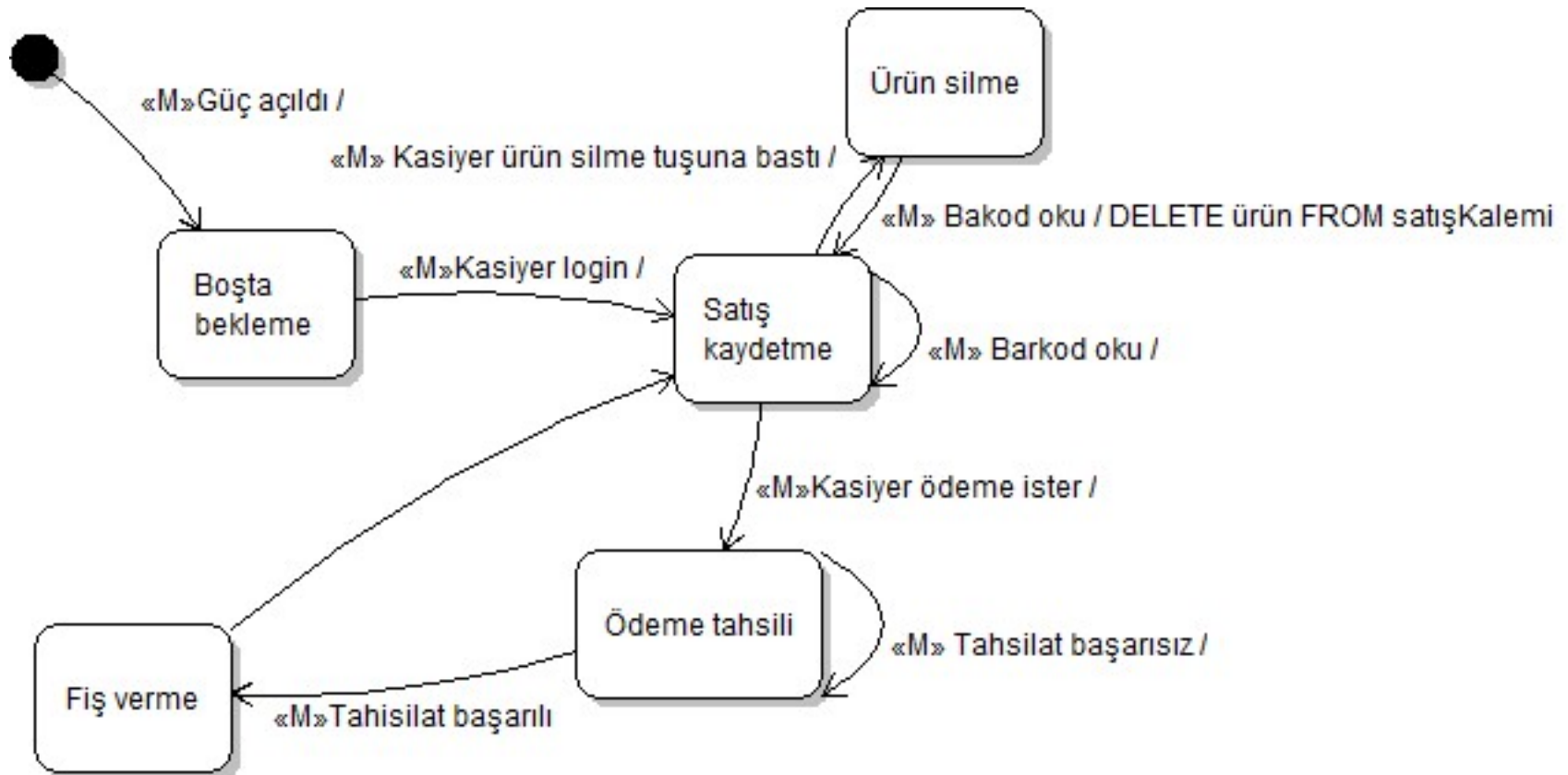
NextGenPOS Tasarım Modeli



TASARIM MODELİ

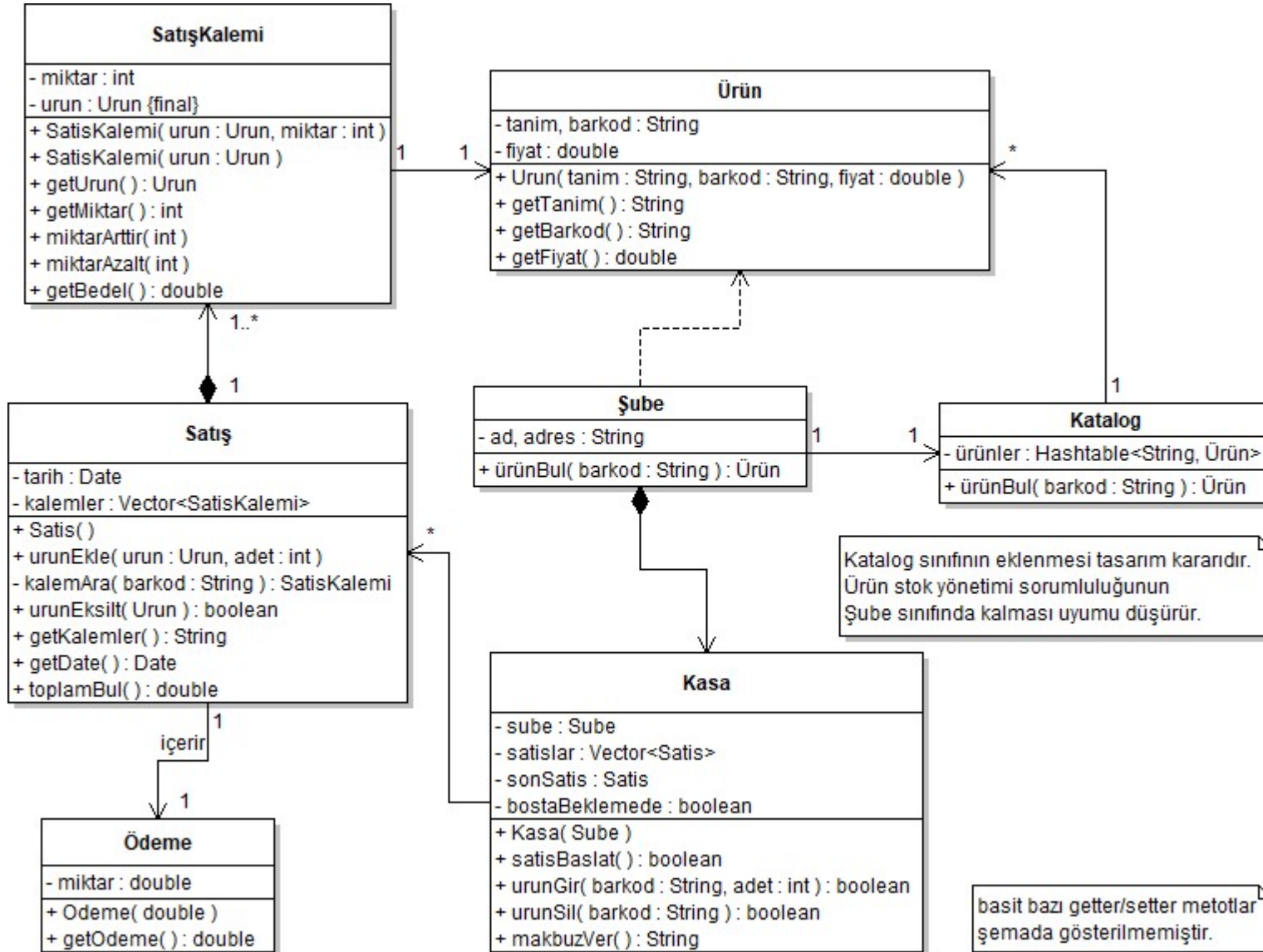
NextGenPOS Tasarım Modeli

- Kasa ile ilgili durum şeması:



TASARIM MODELİ

NextGenPOS Tasarım Modeli





- Bu yansı ders notlarının düzeni için boş bırakılmıştır.



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr. Mehmet Sıddık Aktaş

YAZILIM SINAMA TEKNİKLERİ



YAZILIM SINAMA TEKNİKLERİ

GENEL BİLGİLER

- Yazılım geliştirme karmaşık bir süreç olduğundan, hataların ortaya çıkması kaçınılmazdır.
- Yazılım, yaşam döngüsünün her aşamasında, hatalara karşı sınanır.
 - Gereksinimler arasındaki tutarsızlıklar,
 - Çözümleme şeması ile uygulama alanı arasındaki uyumsuzluklar,
 - Tasarım hataları,
 - Çalışma anı hataları,
 - vb.
- Yaşam döngüsünde ilerledikçe, hataların düzeltilmesi zorlaşacaktır.
- İyi bir sinama yaklaşımı, hataların erken belirlenmesine katkıda bulunacak ve yazılımın kalitesini arttıracaktır.
- Bir sinama yaklaşımı şu bileşenleri içermelidir:
 - Planlama
 - Tasarım
 - Çalıştırma
 - Bilgi toplama ve değerlendirme

YAZILIM SINAMA TEKNİKLERİ

GENEL BİLGİLER

- Sinama yaklaşımlarının genel özellikleri:
 - Sınamalar bileşen düzeyinde başlar ve sistem düzeyinde sonlanır.
 - Ürün tek başına bilgisayar yazılımından ibaret olmayabilir.
 - Sistemin bütününde çeşitli algılayıcılar, harici sistemler, vb. yer alabilir.
 - NYP’de sınıfların bireysel sorumluluklarından başlayıp, yazılımın üst düzey sorumluluklarına kadar tüm işlevler sinanır.
 - Yazılım yaşam döngüsünün değişik aşamalarında değişik sinama teknikleri uygun olacaktır.
 - Sınamalar hem ilgili bileşeni oluşturan kişi tarafından, hem de bağımsız kişiler tarafından yapılır.
 - Büyük ölçekli kurumlarda ayrı bir sinama ekibi bulunabilir.
 - Sinama ve hata ayıklama ayrı işlerdir, ancak hata ayıklama her sinama yaklaşımının önemli bir parçasıdır.
 - Resmi teknik değerlendirmeler sayesinde sinama öncesinde de hatalar belirlenebilir.
 - Formal technical reviews, ileride değinilecek.

YAZILIM SINAMA TEKNİKLERİ

SINAMA AMAÇLI YAPILANMA

- Bir bileşen ilk olarak onu hazırlayan programcı tarafından sınanır.
 - Bir programı en iyi olarak bilen, onu yazandır.
- Her bileşen aynı zamanda yazarı dışındaki kişilerce sınanır.
 - Yazılım evinin bağımsız bir sinama ekibi bulunabilir (ITG: Independent Test Group).
 - Aksi halde farklı projelerde çalışan ekipler, veya aynı projenin farklı alt grupları, birbirlerinin çalışmalarını sınar.
 - Programcıların çıkarları, kendi yazdıkları programın hatasız, zamanında ve bütçe dahilinde tamamlanmasını gerektirir.
 - Psikolojik açıdan bakıldığında sinama ‘yıkıcı’ bir eylemdir.
 - Bu nedenlerle geliştiriciler kendi programlarının hatalarını bulmaya yönelik değil, doğru çalıştığını ispatlamaya yönelik sinamalara eğilim gösterebilir.
- Yazılım teknik olmayan kişiler tarafından da sınanır (ileride değinilecek).
- Sonuç:
 - Müşteriler hataları önünde sonunda bulacaklardır. İyisi mi siz onlardan önce davranın! Aksi halde prestijiniz sarsılacaktır.

YAZILIM SINAMA TEKNİKLERİ

SINAMA YAKLAŞIMININ BELİRLENMESİ

- Göz önünde bulundurulması gereken konular:
 - Sinama çalışmaları, en çok çaba gerektiren yazılım mühendisliği etkinlikleri arasında yer almaktadır.
 - Sinama çalışmaları, olası tüm hataların yakalanacağı bir 'güvenlik ağı' olarak düşünülmemelidir.
 - Sinama için ne kadar çaba gösterilirse gösterilsin, tespit edilemeyen hatalar olacaktır.
 - Hedefler açıkça belirlenmelidir:
 - Testlerin kapsama alanı,
 - Sinama çalışmalarına ne kadar kaynak ayrılacağı,
 - zaman, kişi, bütçe
 - Yazılımın türüne göre belirlenecek diğer çeşitli ölçütler
 - İki hata arası ortalama süre (MTBF), hassasiyet, vb.
 - Sinamalar otomatikleştirilmelidir.
 - Sinama süreci ölçülmeli ve iyileştirilmelidir.



YAZILIM SINAMA TEKNİKLERİ

SINAMA TÜRLERİ

- Yaklaşım tarzlarına göre sinama türleri:
 - Kara kutu sinaması (Black-box testing): Sınanacak birimin iç işleyişi bilinmez, sadece birimin beklenen girdilere karşı beklenen çıktıları üretip üretilmediğine bakılır.
 - Beyaz kutu sinaması (White-box testing): Sınanacak birimin iç işleyişi bilinir ve yapılacak sinamalar buna göre belirlenir.
- Yürütülme sıralarına göre sinama türleri:
 - Doğrulama Sinamaları (verification tests): Yazılım ekibi tarafından yapılır. "Yazılımı doğru mu yapıyoruz?" sorusuna yanıt aranır.
 - Birim sinamaları
 - Tümlleştirme sinamaları
 - Geçerleme Sinamaları (validation tests): Son kullanıcılar tarafından yapılır. "Doğru yazılımı mı yapıyoruz?" sorusuna yanıt aranır.
 - Alfa sinaması
 - Beta sinaması

YAZILIM SINAMA TEKNİKLERİ

DOĞRULAMA SINAMALARI

- Birim sınamaları (Unit testing)
 - En küçük yazılım bileşeninin sınanmasıdır.
 - NYP'de bireysel sınıfların sınanmasıdır.
 - Ne zaman tasarlanır?
 - Kodlamadan önce (çevik yaklaşım) (TDD: Test Driven Design),
 - kodlama sırasında,
 - veya kodlamanın ardından.
 - Kodlama sırasında veya kodlamanın ardından yürütülebilir.
 - Bir sınıfın tek başına yürütemediği sorumlulukların sınanması için, bu sınıfın ihtiyaç duyduğu diğer sınıfların yerine geçecek kod gerekebilir.
 - Vekil, sahte, yalancı kod/sınıf, vb.
 - Stub, dummy, surrogate, proxy, vb.
 - Vekil sınıflar, sadece ihtiyaç duyulan sınıflar gerçekleşene dek kullanılır.
 - Vekil sınıfların basit tutulması, ek kodlama yükünü azaltır.
 - Bu mümkün değilse, ortaklaşa yürütülen sorumlulukların sınanması tümleştirme sınamalarına bırakılır.



YAZILIM SINAMA TEKNİKLERİ

DOĞRULAMA SINAMALARI

- Birim sınamalarında aranabilecek hata türleri:
 - Farklı veri tiplerinin karşılaştırılması veya birbirinin yerine kullanımı.
 - NYP’de: Çokbiçimliliğin yan etkileri
 - Mantıksal işleçlerin yanlış kullanımı
 - İşleçlerin önceliklerinin gözden kaçırılması
 - Değişkenlerin karşılaştırılmasındaki hatalar
 - Döngülerin hatalı sonlanması veya sonsuz döngüler
 - Değişkenlere hatalı değerler atanması
 - vb.
- Yaptığınız hatalardan ders çıkarın:
 - Yapılabilecek tüm kodlama hataları öngörülemez, ancak kariyeriniz boyunca her hata yaptığınızda bu hatanızı ‘aranabilecek hata türleri’ listenize ekleyin.



YAZILIM SINAMA TEKNİKLERİ

DOĞRULAMA SINAMALARI

- Tümlleştirme sınamaları (Integration testing)
 - Sınıflar birim sınamalarını geçmişlerse, bir araya getirildiklerinde de doğru çalışmazlar mı?
 - Yazılım geliştirme sürecinin her aşamasında her ayrıntının açıkça belirlenmesi beklenemez.
 - Ayrı ayrı programcılar, belirlenmemiş ayrıntılar üzerinde kendi karar verme yetkilerini (initiative) kullanabilir.
 - Aynı kişinin farklı ayrıntılar hakkında verdiği kararlar bile birbiri ile uyumlu olmayabilir.
 - NYP'de birim ve tümlleştirme sınamalarını birbirinden ayıran kesin çizgiler yoktur.



YAZILIM SINAMA TEKNİKLERİ

DOĞRULAMA SINAMALARI

- Tümlleştirme sinamaları türleri:
 - Tahribat sinaması (smoke testing)
 - Yüzeysel ancak başarısız olma durumunda tüm sistemin çalışmasının olanaksız olacağı sinamalardır (show-stopper errors).
 - Diğer tümlleştirme sinamalarından önce yapılır.
 - Çeşitli bileşenler tüm gerekli yazılım elemanlarını içeren (kod, yapılandırma dosyaları, dış kütüphaneler, vb.) parçalar bir araya getirilir (build) ve günlük olarak sinanırlar.
 - Geriye dönük sinama (regression testing)
 - Yazılıma yeni bir işlev veya bileşen eklendiğinde, tüm sinamaların yenilenmesidir.
 - Hangi ölçekte bir eklentinin geriye dönük sinamayı başlatacağının kararını vermek gerekir.
 - Ölçek düştükçe sinama sıklaşır ve masraf artar.
 - Otomatik sinama gereçleri kullanılarak masraflar azaltılabilir.



YAZILIM SINAMA TEKNİKLERİ

GEÇERLEME SINAMALARI

- Geçerleme sınamaları (validation testing):
 - Gereksinimler belgesinde yazılmış olan işlevsellikten yola çıkılarak, kullanıcı tarafından yapılır.
 - Son kullanıcıların yapabileceği beklenmedik davranışların tümünü, teknik ekip önceden bilemez.
 - ‘Fincan tutacağıının çalışmaması’, ‘Pencereyi açmak’, ...
 - Bir şeyi ne kadar çok kişi incelerse, ondaki kusurlar o kadar çabuk bulunur ve düzeltilir.
 - Eric Raymond: “given enough eyeballs, all bugs are shallow”
 - Alfa ve Beta sınaması olmak üzere iki türü vardır.



YAZILIM SINAMA TEKNİKLERİ

GEÇERLEME SINAMALARI

- Alfa sinaması:
 - Yazılım firması içerisinde, kullanıcı tarafından yapılır.
 - Yazılım geliştirme ekibinin denetiminde ve izlemesi ile yapılır.
 - Yazılımın doğal kullanım ortamına en yakın koşullarda yürütülür.
- Beta sinaması:
 - Müşterinin kendi yeri içerisinde, gerçek kullanım ortamında yapılır.
 - Yazılım geliştirme ekibi müdahil olmaz.
 - Bulunan hatalar yazılım geliştirme ekibine düzenli aralıklarla ve resmi bir biçimde bildirilir.

YAZILIM SINAMA TEKNİKLERİ

SİSTEM SINAMALARI

- Yazılım tek başına sistemin bütünü oluşturmayabilir.
 - Gömülü uygulamalar, ara katman yazılımları, vb.
- Sistemin tümünü her yönüyle incelemeye yönelik sınamalardır:
 - Kurtarma (recovery) sınaması:
 - Hatalara dayanıklı (fault tolerant) sistemler için geçerlidir.
 - Bir hata ortaya çıktığında sistemin kendini toparlayarak doğru çalışmaya devam edip edemediği sınanır.
 - Kurtarma işlemi belli bir süre içerisinde tamamlanmalıdır (MTTR: Mean Time To Repair).
 - Güvenlik (security) sınaması:
 - Tek kural: Kural yok!
 - Her güvenlik önünde sonunda aşılır!
 - Zorlama (stress) sınaması:
 - Normalin dışında yüklenme durumunda, sistemin nereye kadar dayanabileceğinin sınanması
 - Başarım (performance) sınaması:
 - Gerçek zamanlı uygulamalarda özel öneme sahiptir.
 - Program kendisinden bekleneni doğru yapabilir ama zamanında yapamayabilir.



YAZILIM SINAMA TEKNİKLERİ

HATA AYIKLAMA (DEBUGGING)

- Yapılan türlü sınamaların sonucunda bulunan hatalar düzeltilmelidir.
- Sınama ve hata ayıklama çalışmaları kimileri tarafından ‘angarya’, ‘ayak işi’, ‘sıkıcı’, ‘ikinci sınıf’ olarak nitelendirilebilir.
 - Sinamanın önemini gördük
 - Yazılımınızdaki hatalar size ve kurumunuza prestij kaybettirir.
 - Düzeltilmeyen veya düzeltmesi uzun süren hatalar ise daha çok prestij kaybettirir.
 - Sınama ve güvenlik açıklarının belirlenmesi için alışlagelmiş düşünce biçiminin dışına çıkabilmek gerekir ki bu da özel bir yetenektir.
 - Saygı duyulan ve ünlü programcılar, hata ayıklamanın kodlama yapmaktan daha zor ve daha çok yetenek isteyen bir iş olduğu görüşünde birleşmektedirler.
- Bir noktada takıldığınızda, biraz ara verip sorunu yeniden incelemeniz, başarı şansınızı arttıracaktır.
- Geliştirme ortamının hata ayıklama yeteneklerinden sonuna kadar yararlanın.



YAZILIM SINAMA TEKNİKLERİ

YAZILIM GÖZDEN GEÇİRME EYLEMİ

- Yazılım gözden geçirme: Software review
 - Yazılım sinamasından önce, sinama eylemlerinden daha az masrafla, yazılım hatalarının bulunmasını amaçlar.
 - Sinama çalışmaları ile bulunabilecek tüm hatalar gözden geçirme ile bulunamaz, ancak gözden geçirme daha verimlidir.
 - Gözden geçirme resmi veya gayri resmi olabilir.
 - Çalışmalar resmi gözden geçirmelerin daha etkili olduğunu göstermiştir.
 - Gözden geçirme toplantılarının yapılması durumunda.
 - Çevik süreçlerden XP'deki eşli programlama da bir tür gözden geçirmediir.
- Resmi yazılım gözden geçirme: Formal software review
 - Toplantı şeklinde yapılır.
 - Toplantının bir yöneticisi (review leader) bulunur.
 - Bu konuda çeşitli standartlar önerilmiştir.
 - Ör: IEEE 1028 standardı

YAZILIM SINAMA TEKNİKLERİ

RESMİ YAZILIM GÖZDEN GEÇİRME ÇALIŞMALARI

- IEEE 1028 std. göre bir resmi yazılım gözden geçirme adımları:
 - Değerlendirme başlangıcı: Değerlendirme yöneticisi standart bir checklist kullanarak, verimli bir toplantı için gerekli koşulları sağlar.
 - Yönetimin hazırlanması: Sorumlu yönetim gözden geçirme için gerekli kaynakları hazırlar ve toplantının standartlara uygun yürütülmesini sağlar.
 - Gözden geçirme prosedürlerine genel bakış: Değerlendirme yöneticisi tüm değerlendiricilerin gözden geçirmenin amaçlarını ve prosedürlerini anladığından emin olur.
 - Bireysel hazırlık: Değerlendiriciler bireysel olarak inceleme toplantısına hazırlanır. Bu amaçla gözden geçirilecek malzemedeki hataya yol açabilecek olası bozukluklar (anomaly) aranır.
 - Grup incelemesi: Bireysel hazırlıkların sonuçları önceden belirlenen yer ve zamandaki toplantıda biraraya getirilir ve sonuç raporu üzerinde uzlaşıya varılır.
 - Düzenleme: İncelenen çalışmanın yazar(lar)ı veya atanacak bir başka kişi/ekip, önceki adımda belirlenen noktaları düzeltir.
 - Sonlandırma: Değerlendirme yöneticisi düzeltmelerin yeterliliğini inceler.



YAZILIM SINAMA TEKNİKLERİ

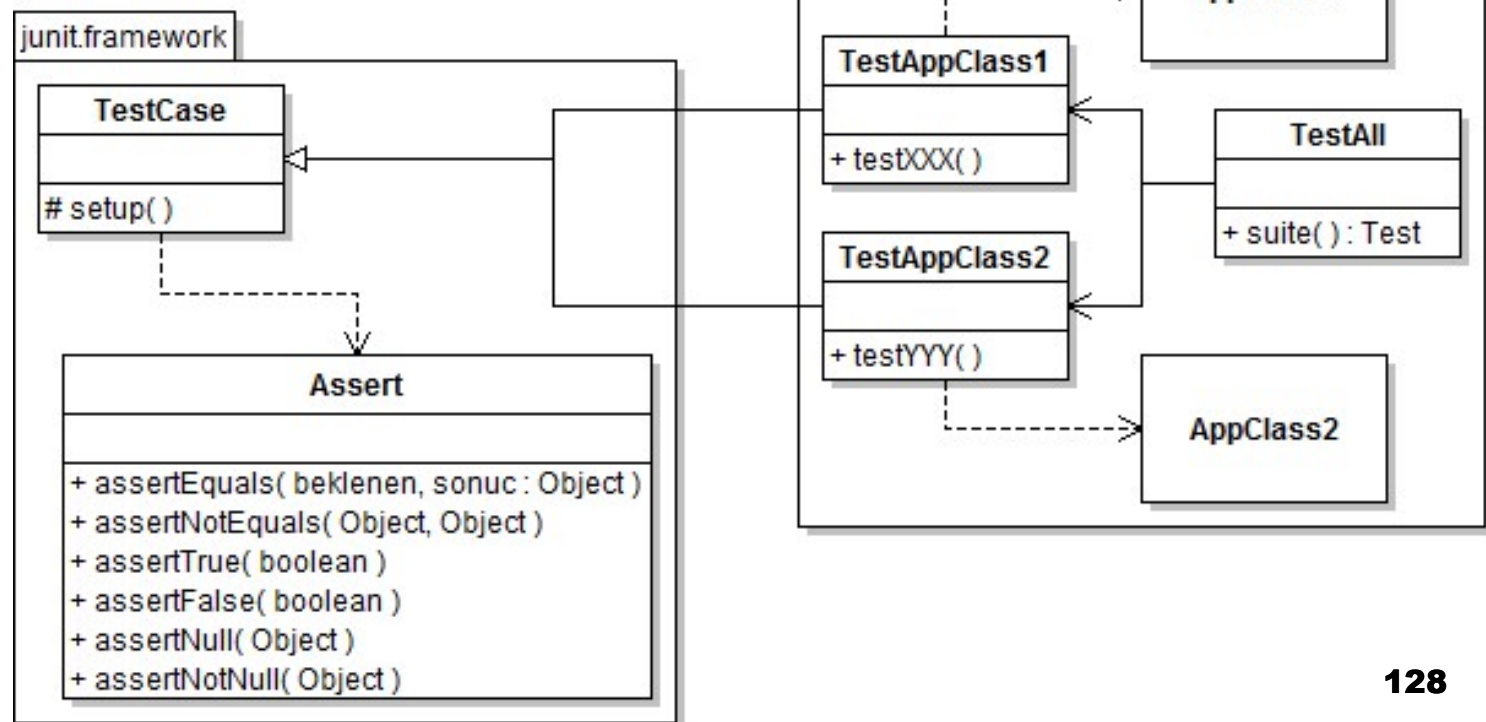
JUNIT İLE BİR BİRİM SINAMASI UYGULAMASI

- junit, Java ile yazılmış kodun birim sınamaları için bir çerçeve programdır (framework)
- Diğer diller için de sürümleri bulunmaktadır.
 - Ör. C# için csUnit
- IDE desteği:
 - Eclipse ve NetBeans'de IDE ile hazır geliyor.

YAZILIM SINAMA TEKNİKLERİ

JUNIT İLE BİR BİRİM SINAMASI UYGULAMASI

- jUnit Sürüm 3.X ile Test Case Hazırlamak:
 - Her sınıfın birim testi için ayrı sınıfta ayrı test case'ler hazırlanır.
 - Test case'leri bir test suite altında toplanabilir. TestAll.suite() metodu statiktir.
 - TestCase.setup metodu her testXXX() için yeniden çalışır.



YAZILIM SINAMA TEKNİKLERİ

JUNIT İLE BİR BİRİM SINAMASI UYGULAMASI

- jUnit Sürüm 4.X ile Test Case Hazırlamak:
 - Geriye doğru uyumluluk korunmakla birlikte, jUnit 4 sürümü ile annotation desteği gelmiştir.
 - Artık sinama sınıflarının TestCase sınıfından kalıtımla türetilmesi gerekmemektedir.
 - Artık sinama metotlarının adlarını test kelimesi ile başlatmak gerekli değildir, ilgili metotların başına @Test annotation koymak yeterlidir.
 - setup adlı metodun yerine ise @Before annotation gelmiştir.

```
@Before
public void setUp() { /*Preparations*/ }

@Test
public void testSomething() { /*Do test*/ }
```

 - Exception tanımlanan yazılımlarda atılması gereken exception'ların gerçekten ortaya çıkıp çıkmadığının sinanması da mümkün olmuştur.

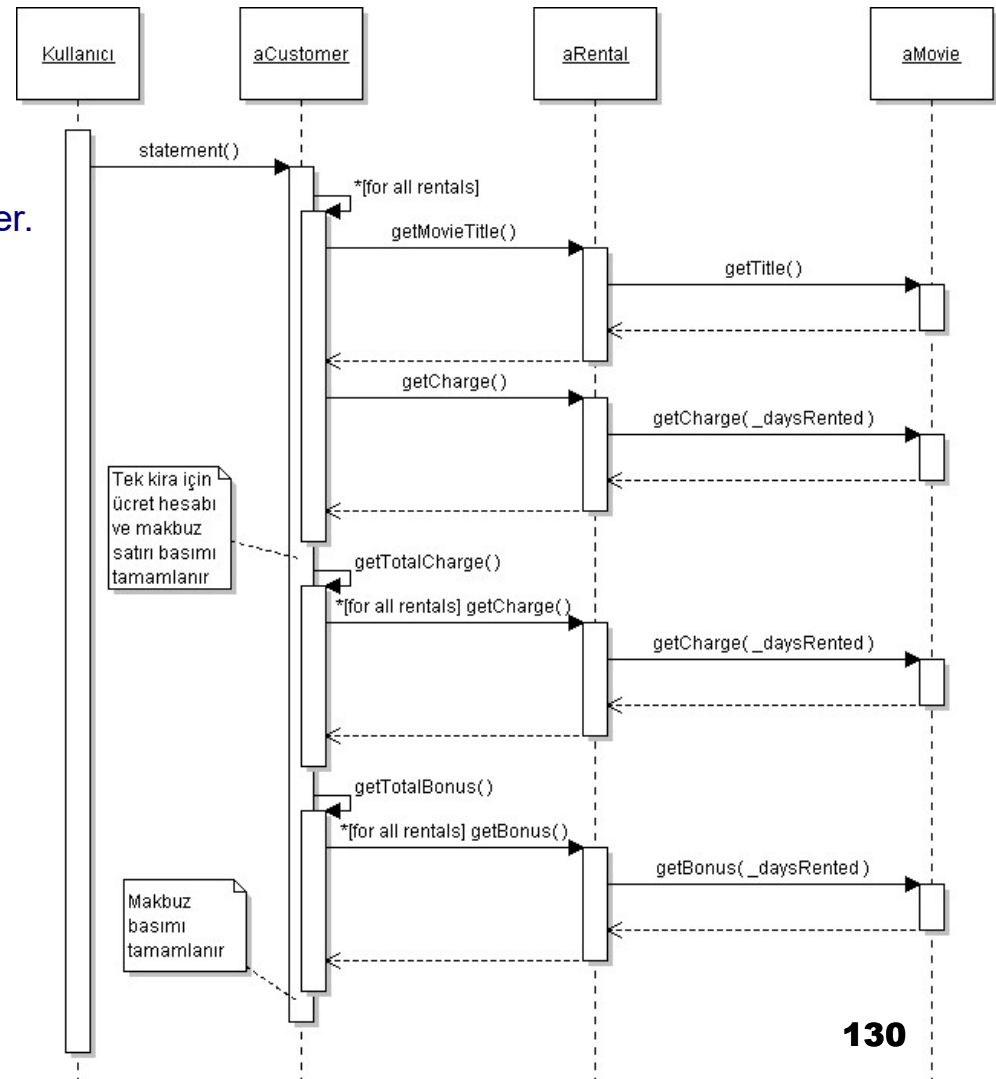
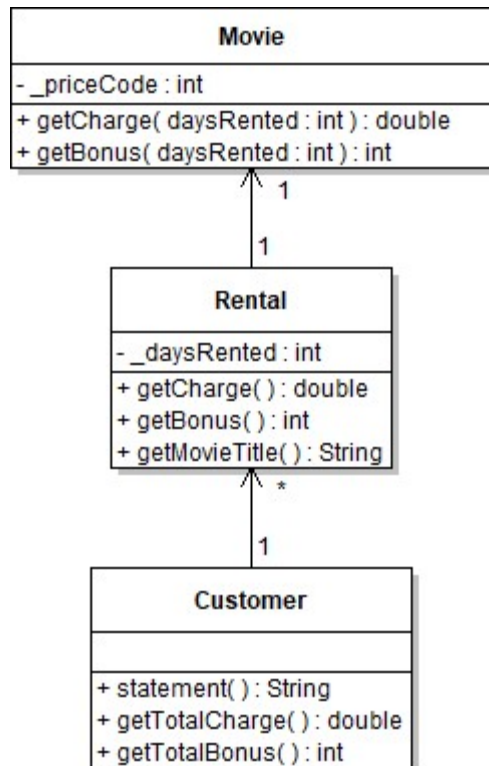
```
@Test(expected=SomeException.class)
public void testTheException() throws Exception {
    doSomethingThatCreatesTheException();
}
```

v12
y13

YAZILIM SINAMA TEKNİKLERİ

JUNIT İLE BİR BİRİM SINAMASI UYGULAMASI

- Basit bir video kaset kiralama örneği üzerinden jUnit Test Case'leri kodlayalım.
 - Kaynak: Refactoring: Improving the Design of Existing Code, Martin Fowler. Addison-Wesley, 1999



Slayt 130

y12

NextGenPos kodumuz için jUnit Test Case'leri kodlayalım.

yunus, 4/13/2021

y13

Ben buraya NDTM örneğini aldım ama NextGenPos olsa daha iyi.

yunus, 4/13/2021

YAZILIM SINAMA TEKNİKLERİ

JUNIT İLE BİR BİRİM SINAMASI UYGULAMASI

- Test case kaynak kodları:

```
package fowler_09;
import junit.framework.TestCase;
public class TestCustomer extends TestCase {
    private Customer yunus;
    private Movie matrix, monster, surrogate, terminator;
    protected void setUp() {
        yunus = new Customer("Yunus Emre Selçuk");
        matrix = new Movie("The Matrix",Movie.REGULAR);
        monster = new Movie("Monsters, Inc.",Movie.CHILDRENS);
        surrogate = new Movie("Surrogates", Movie.NEW_RELEASE);
        terminator = new Movie("Terminator Salvation",Movie.NEW_RELEASE);
    }
    public void testGetName( ) {
        String sonuc = yunus.getName( );
        assertEquals("Yunus Emre Selçuk",sonuc);
    }
    public void testStatementWhenEmpty( ) {
        String sonuc = yunus.statement();
        String beklenen = "Rental Record for Yunus Emre Selçuk\n";
        beklenen += "Amount owed is 0.0\n";
        beklenen += "You earned 0 frequent renter points";
        assertEquals(beklenen, sonuc);
    }
}
```

YAZILIM SINAMA TEKNİKLERİ

JUNIT İLE BİR BİRİM SINAMASI UYGULAMASI

- Test case kaynak kodları (devam):

```
public void testStatementWithMoviesLongRent( ) {
    yunus.addRental( new Rental(matrix, 3) );
    yunus.addRental( new Rental(monster, 4) );
    yunus.addRental( new Rental(surrogate, 2) );
    String sonuc = yunus.statement();
    String beklenen = "Rental Record for Yunus Emre Selçuk\n";
    beklenen += "\tThe Matrix\t3.5\n";
    beklenen += "\tMonsters, Inc.\t3.0\n";
    beklenen += "\tSurrogates\t6.0\n";
    beklenen += "Amount owed is 12.5\n";
    beklenen += "You earned 4 frequent renter points";
    assertEquals(beklenen, sonuc);
}

public void testStatementWithMoviesShortRent( ) {
    yunus.addRental( new Rental(matrix, 2) );
    yunus.addRental( new Rental(monster, 3) );
    yunus.addRental( new Rental(surrogate, 1) );
    String sonuc = yunus.statement();
    String beklenen = "Rental Record for Yunus Emre Selçuk\n";
    beklenen += "\tThe Matrix\t2.0\n";
    beklenen += "\tMonsters, Inc.\t1.5\n";
    beklenen += "\tSurrogates\t3.0\n";
    beklenen += "Amount owed is 6.5\n";
    beklenen += "You earned 3 frequent renter points";
    assertEquals(beklenen, sonuc);
}
```

YAZILIM SINAMA TEKNİKLERİ

JUNIT İLE BİR BİRİM SINAMASI UYGULAMASI

- Test case kaynak kodları (devam):

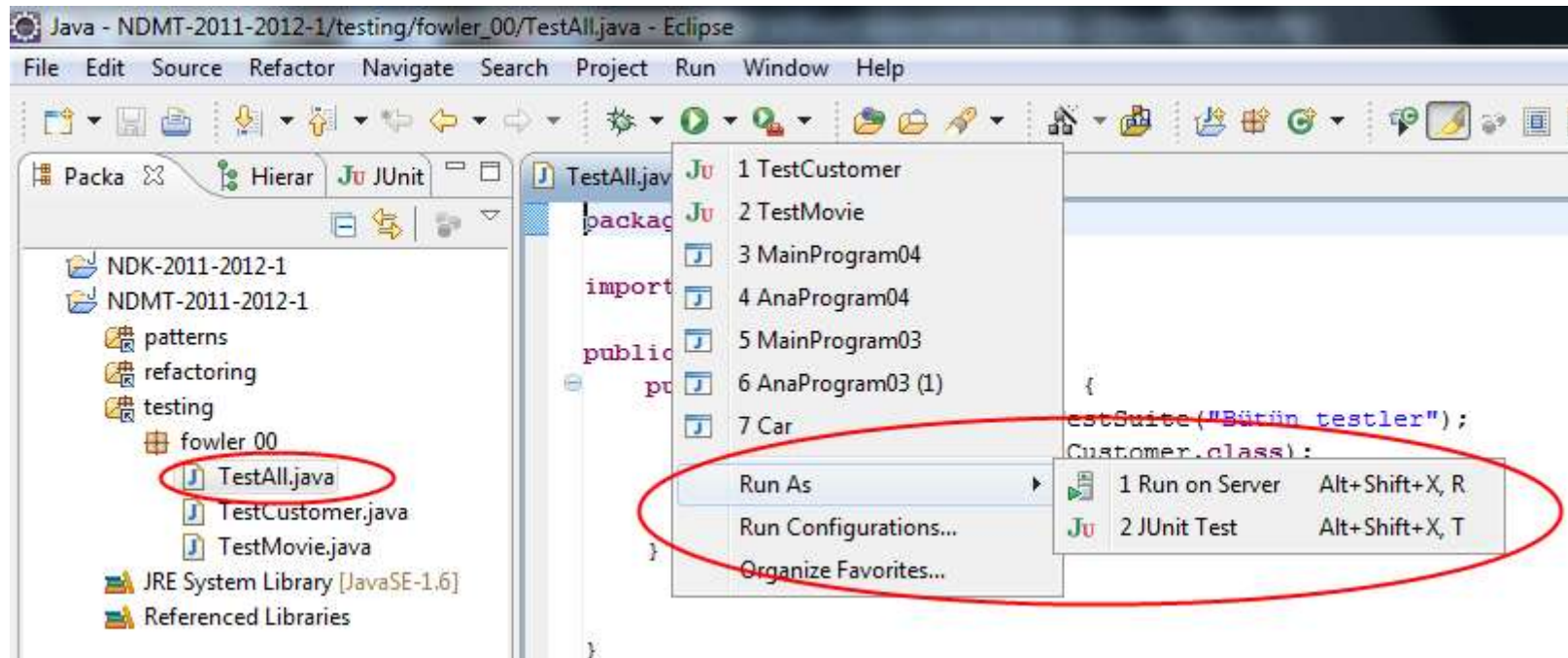
```
public void testNewReleaseRentalBonus( ) {
    yunus.addRental( new Rental(surrogate, 2) );
    yunus.addRental( new Rental(terminator, 1) );
    String sonuc = yunus.statement();
    String beklenen = "Rental Record for Yunus Emre Selçuk\n";
    beklenen += "\tSurrogates\t6.0\n";
    beklenen += "\tTerminator Salvation\t3.0\n";
    beklenen += "Amount owed is 9.0\n";
    beklenen += "You earned 3 frequent renter points";
    assertEquals(beklenen, sonuc);
}
}
```

- Test suite oluşturmak:

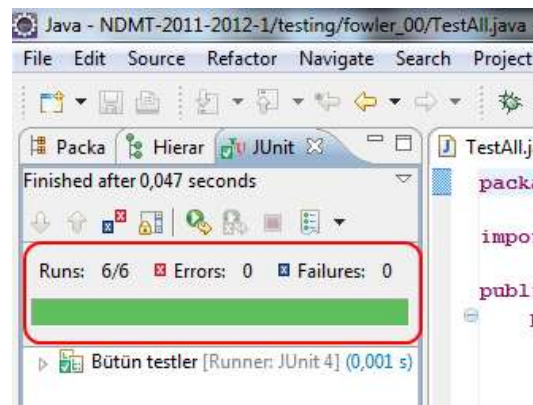
```
package fowler_09;
import junit.framework.*;
public class TestAll {
    public static Test suite( ) {
        TestSuite suite = new TestSuite("Bütün testler");
        suite.addTestSuite(TestCustomer.class);
        suite.addTestSuite(TestMovie.class);
        return suite;
    }
}
```


JUNIT İLE BİRİM SINAMALARI

- Testi çalıştıralım ...



- ... ve sonuç:



y14

Slayt 134

y14

NDTM notlarında fowler_10 vb de var. Onları da alsam? Biri tasarım kalıplı idi, bahsetsem?

yunus, 4/13/2021



KOD KUSURLARI (CODE SMELLS) VE GİDERİLMELERİ

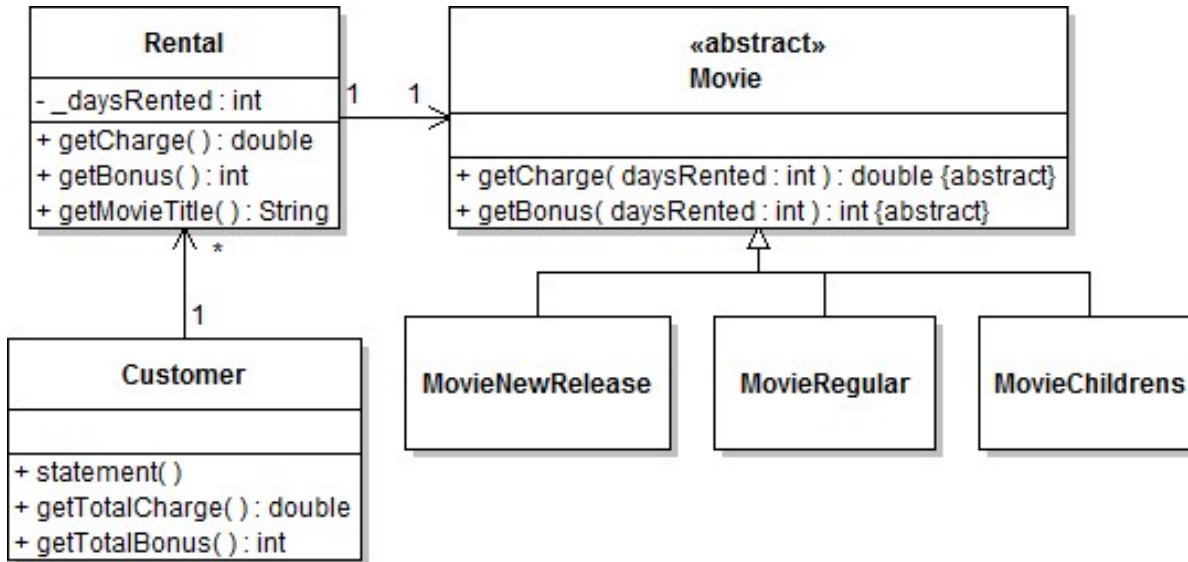
CODE SMELL NEDİR?

- Yazılan kodu iyileştirme gerektiğinin işaretleridir.
- Kod içerisinde karşılaşılan tersliklerdir.
- Kullanılan programlama yaklaşımının kusurlu kullanım örnekleridir.
- Yazılım elimizden kayıp gitmeden, tehlike işaretlerini sezip önlem almak yerinde olacaktır.
 - Bu etkinliklere yeniden mühendislik / refactoring kategorisine girer.
 - Bakım aşamasına geçilmeden önce başlanması önerilir, sınama etkinliklerinin bir parçası olarak düşünülebilir.
- Önceki video kaset kiralama örneği, çok kusurlu bir yazılımın iyileştirme adımları arasında sona çok yakın olan bir aşama idi.

KOD KUSURLARI (CODE SMELLS) VE GİDERİLMELERİ

BİRAZ DAHA REFACTORING:

- Daha iyi bir tasarım alternatifi:



- Bu alternatifiñ dezavantajı, test case'lerin değıştirilmesini gerektirecek kadar büyük ölçekli bir tasarım olmasıdır. Bu nedenle:
 - Bakım aşamasına geçilmediyse yapılması önerilir.
 - Bakım aşamasına geçildiyse, geriye doğru uyumluluđu korumak için yapılmaması önerilir.
 - Yazılımın bir önceki sürümü emekli edilecekse, yeni sürümünde yapılması önerilir.



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr. Mehmet Sıddık Aktaş

YAZILIM PROJE YÖNETİMİNE GİRİŞ



YAZILIM PROJE YÖNETİMİNE GİRİŞ

GENEL BİLGİLER

- Yazılım projeleri önemli oranda başarısızlığa uğramaktadır:
 - Yazılım geliştirmedeki zorluklar.
 - Ölçek büyüklüğünden kaynaklanan zorluklar: Yazılım ölçeği, kişi ölçeği, vb.
 - Kestirimdeki zorluklar.
 - İnsanlarla çalışmadaki zorluklar.
 - Teknolojideki değişimler.
 - Gereksinimlerdeki değişimler.
 - Politik değişimler.
 - Mali değişimler.
- Yazılım proje yönetimi, sayılan zorlukların çözümüne odaklanır.
- Önem sırasına göre proje yönetiminin ilgi alanları:
 - Kişiler
 - Ürün
 - Süreç
 - Proje

YAZILIM PROJE YÖNETİMİNE GİRİŞ

GENEL BİLGİLER

- Yazılım proje yönetimi; yazılım mühendisliği teknikleri, genel yönetim bilimi ile proje yönetimi bilgi ve deneyiminin ara kesitidir.





YAZILIM PROJE YÖNETİMİNE GİRİŞ

GENEL BİLGİLER

- Yazılım projesinin büyüklüğüne göre, uygulanacak yönetim ve denetiminin düzeyi, kullanılacak araç tipleri ve teknoloji gereği farklılık göstermektedir.
 - Küçük ölçekli projeler:
 - Çok küçük: 1-2 kişi içeren çok kısa süreli (1-2 ay) projeler
 - Küçük: 2-3 kişi içeren kısa süreli (2-3 ay) projeler
 - Basit bir kullanıcı uygulama programı tekniği ile düzenlenebilmektedir.
 - Büyük ölçekli projeler:
 - 5-20 kişilik bir ekip tarafından, 2-3 yılda tamamlanabilmektedir.
 - Çok sayıda alt sistemden oluşmaktadır.
 - Başka sistemlerle tümleştirme gereksinimleri mevcuttur.
 - İş planlama ve kod yönetim uygulamalarının kullanılması zorunludur.



YAZILIM PROJE YÖNETİMİNE GİRİŞ

GENEL BİLGİLER

- Temel amacın kullanıcılara bir yarar sağlamak olduğunu hiçbir aşamada unutmayın.
- Planlama yaklaşımları:
 - Basitlik yaklaşımı: Geleceğin getireceği değişiklikler çoğu zaman ayrıntılı planlama gereğini ortadan kaldırır.
 - Geleneksel yaklaşım: Planlama proje için bir yol haritası belirler; harita ne kadar ayrıntılı ise kaybolma olasılığı o kadar düşer.
 - Çevik yaklaşım: Ön hazırlık gereklidir ancak asıl harita proje ilerledikçe çizilir.



YAZILIM PROJE YÖNETİMİNE GİRİŞ

PLANLAMA

- Planlama ilkeleri:
 - Projenin sınırlarını belirleyin: Nereye gideceğinizi bilmezseniz kaybolursunuz.
 - Müşteriyi planlama eylemlerine katın: Öncelikleri, sınırları ve zamanlamayı müşteri belirler (bu sırada) ancak gerçekçiliği korumak amacıyla yazılım ekibi pazarlık yapar (aksi sırada).
 - Planlamanın doğası yinelemelidir: Plan asla taşa yazılmaz.
 - Bildiklerinizi kullanarak kestirimlerde bulunun: Bilginin kapsamı, doğruluğu ve belirginliği kestirimin doğruluğunu etkiler.
 - Planın her aşamasına risk değerlendirmesini ekleyin: Teknik, mali, kişisel, politik riskler.
 - Gerçekçi olun: Yazılımcı süpermen veya robot değildir.



YAZILIM PROJE YÖNETİMİNE GİRİŞ

PLANLAMA

- Planlama ilkeleri (devam):
 - Planların ölçeği: İnce ayrıntılı planlar daha kısa vadeli, genel ayrıntılı planlar daha uzun vadelidir. Zaman ilerledikçe genel ayrıntıdan ince ayrıntıya geçilir.
 - Kalite güvence eylemlerini tanımlayarak plana ekleyin.
 - Değişikliğin nasıl kabul edileceğini müşteri ile sözleşmeye bağlayın.
 - Planın gidişinden gözünüzü ayırmayın ve gerekli ayarlamaları yapın.

YAZILIM PROJE YÖNETİMİ

RISK YÖNETİMİ

- Risk ile uğraşma taktikleri:
 - Sonradan (Reactive): Risk gerçekleşince çaresine bakmak.
 - Önceden (Proactive): Riskleri daha gerçekleşmeden önlemeye çalışmak.
 - Nasrettin hoca: Kızını dövmeyen dizini döver!
 - İtfaiyeci ve işçi/elektrikçi.
 - Proactive yaklaşımları işleyeceğiz.
- Risk tanımı:
 - Tam bir uzlaşma yok.
 - Uzlaşılan özellikler:
 - Olasılık: Belirli bir risk ortaya çıkabilir veya çıkmayabilir, risk %100 olasılıkla ortaya çıkacak diye bir şey yoktur.
 - Kayıp: Risk ortaya çıktığında istenmeyen sonuçlar ve kayıplar doğurur.
- Genel risk çeşitleri:
 - Proje riskleri
 - Teknik riskler
 - İş riskleri
- Farklı risk sınıflandırmaları ve risk işleme önerileri için bkz. SEI, ISO, ANSI, makaleler, kitaplar, vb.



YAZILIM PROJE YÖNETİMİ

RISK YÖNETİMİ

- Proje riskleri:
 - Proje planını tehdit eder.
 - Gerçekleşirse zamanlama ileri tarihlere sarkar ve maliyet artar.
 - Örnekler: Bütçe riskleri, Zaman riskleri, Personel riskleri, vb.
- Teknik riskler:
 - Üretilen yazılımın kalitesini ve zamanında bitirilmesini etkiler.
 - Gerçekleşirse yazılımı gerçekleştirme zorlaşır veya imkansızlaşır.
 - Çözümleme, tasarım, gerçekleştirme ve bakım aşamaları ile ilgili risklerdir.
 - Örnek: 'Son teknoloji' ürünler yüksek teknik riske sahiptir.
 - Bug'lar var mı? Dokümantasyonu tam mı? Tam anlayabildik mi bu teknolojiyi? Yarın da bu teknoloji hayatta olacak mı?
- İşletme riskleri:
 - Pazar riskleri: Ürüne talep olur mu? Ör. Tıraş bıçağı, tıraş makinesi
 - Satış riskleri: Pazarlama ekibi ürünü nasıl satacağını biliyor mu?
 - MIS/MBA konuları!

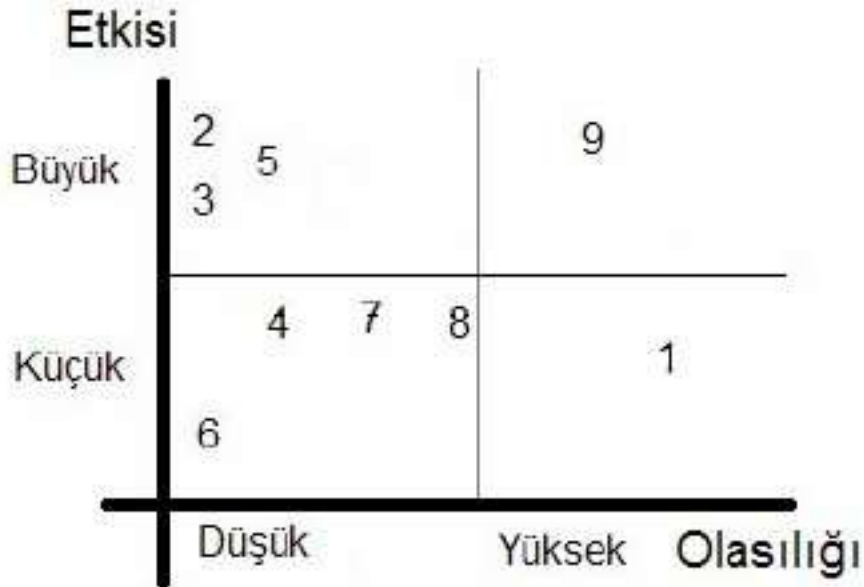
YAZILIM PROJE YÖNETİMİ

RİSK İŞLEME VE RİSK TABLOSU

- Risk tablosu oluşturma:
 - Tüm ekip veya risk işleme ekibinin tüm üyeleri olası riskleri adlandırsın.
 - Risklerin çeşitlerini belirleyin (proje, teknik, iş)
 - Herkes kendi uzmanı olduğu risk alanında riskleri alt türlere ayırsın.
 - Ör. Teknik: Planlanandan daha düşük yeniden kullanım, kullanılan gereçlerde deneyimsizlik, vb.
 - Risklerin gerçekleşme olasılıklarını belirleyin.
 - Düşük, orta, yüksek gibi kategoriler.
 - Risklerin etkilerinin büyüklüğünü değerlendirin
 - Küçük bir sıyrık, haydi gayret, siz beni bırakın.
 - küçük kıyamet, büyük kıyamet.
 - Bu bilgilerden bir tablo yapın.
 - ID, ad, çeşit, olasılık, etki.
 - Listeyi bir yerden sonra kesin (Düşük olasılık ve etkileri atın).
 - Ele alacağınız riskler için risk bilgi sayfaları oluşturun.
 - Bu sayfa risk hakkında bilgiler ve önleme yollarını içersin.
 - Önlenemeyen riskler için ise alternatif planlar içersin.

ÖRNEK ÖZET RİSK TABLOSU ve GRAFİĞİ

Risk ID	Adı	Türü,Grubu	Etkisi	Olasılık
01	Organizasyon	Proje	Küçük	Yüksek
02	Tahmin	Proje	Büyük	Düşük
03	Kullanışlılık	Teknik	Büyük	Düşük
04	Doğruluk	Teknik	Orta	Düşük
05	Güvenilirlik	Teknik	Büyük	Düşük
06	Personel	Proje	Küçük	Düşük
07	Araçlar	Teknik	Orta	Orta
08	Metot	Teknik	Orta	Orta
09	Pazarlama	İş	Büyük	Yüksek



- **Sonuç:**
 - 4, 6, 7, 8. risklerin izlenmesine gerek yoktur.
 - Diğer riskler için ayrıntılı risk bilgi sayfaları oluşturulmalıdır.

ÖRNEK RİSK BİLGİ SAYFASI

RİSK #09: Pazarlama

Olasılık: Yüksek **Etki:** Büyük **Türü:** Ticari

Açıklama: Firmamız sadece yazılım geliştiricilerden oluşan çekirdek kadroyla çalışmaktadır. Ortaya çıkaracağımız yazılımın piyasada çok fazla alternatifi vardır.

İşaretleri:

1. Müşteri adayları ile görüşmede dinleyicilerin ilgisizliği ve "Biz zaten X yazılımını kullanıyoruz. Neden sizi seçelim?" türü yorumları.
2. Bakım aşamasının ilk altı aylık süresince ürünümüzün kendisini amorti etme oranının %50'nin altında kalması

Önlemler:

1. Bir programcımıza MBA yaptırmak.
2. Gereksinim mühendisliği aşamasında konu uzmanlığı odaklı danışmanlık hizmeti almak.
3. Bir ya da iki firma ile pilot uygulama yapmak ve bu firmalara ürünü özel indirim ile satmak



YAZILIM PROJE YÖNETİMİ

PROJE YÖNETİMİNDE KİŞİ ETKENLERİ

- Takım yöneticisi:
 - Teknik ekibin bir parçası olduğundan teknik yetenekleri yüksek olmalıdır.
 - Ağırlıklı olarak insanlarla ilgili eylemlerde bulunacağından, sosyal ve yönetsel yetenekleri de yüksek olmalıdır.
- İyi bir teknik yöneticinin özellikleri:
 - Teknik ekibi istekli kılabilmelidir.
 - Kişileri ve yazılım geliştirme sürecini, üzerinde çalışılan ürüne/ürün parçasına göre düzenleyebilmelidir.
 - Düzenleme küçük veya büyük ölçekte olabilir.
 - Ürün ve süreç sınırları belli olsa da, ekibini yaratıcı fikirler üretmeye teşvik edebilmelidir.
 - İyi bir sorun çözücü olmalıdır.
 - Hem teknik hem de yönetsel sorunlarla uğraşabilmelidir.
 - Sorunlara tanı koyabilmeli ve ortaya uygun bir çözüm koyabilmelidir.
 - Seçilen çözüm tıkandığında ısrarcı olmamalıdır.
 - Sorumluluk alabilmelidir.



YAZILIM PROJE YÖNETİMİ

PROJE YÖNETİMİNDE KİŞİ ETKENLERİ

- Yazılım geliştirme ekibi (teknik ekip):
 - Takım ruhuna uygun kişilerden oluşmalıdır:
 - Takım üyeleri birbirine saygı duymalıdır.
 - Takım üyeleri ortak amaç etrafında kenetlenebilmelidir.
 - Takım üyeleri birbirlerini tamamlayan yeteneklere sahip olmalıdır.
 - Takım ruhunu bozan etkenler:
 - Telaşlı iş ortamı.
 - Sık ortaya çıkan hayal kırıklıkları ve başarısızlıkların takım üyeleri arasındaki sürtüşmeyi arttırması.
 - Doğru yönetilemeyen yazılım geliştirme süreci.
 - Takım yapısının ve rollerinin belirsiz tanımlanması.



YAZILIM PROJE YÖNETİMİ

PROJE YÖNETİMİNDE KİŞİ ETKENLERİ

- Takım yapıları:
 - Kapalı yaklaşım:
 - Geleneksel bir yetki hiyerarşisi ve kontrol mekanizmaları bulunur.
 - Geçmiş deneyimlere benzer projelerde başarılı bir yapıdır.
 - Yaratıcı fikirler ortaya çıkarmak için çok uygun değildir.
 - Rastgele (Random) yaklaşım: Serbest yaklaşım.
 - Takım üyelerinin bireysel ve teknik yeteneklerine göre kendi aralarında bir yapı kurmasıdır.
 - Yaratıcı fikirler ortaya çıkarmak için en uygun yaklaşımdır.
 - Disiplin elde etmek zor olabilir.
 - Açık yaklaşım: Kapalı ve rastgele arasında.
 - Kontrol mekanizmaları bulunur ancak yapılanma serbesttir.
 - Demokratik yapı.
 - Karmaşık sorunların çözümü için uygun.
 - Etkinliği (efficiency) sağlamak zor olabilir.



YAZILIM PROJE YÖNETİMİ

PROJE YÖNETİMİNDE KİŞİ ETKENLERİ

- Takım yapıları (devam)
 - Eşzamanlı (synchronous) yaklaşım:
 - Problemin takımın üzerine düşen bölümünün de alt parçalara ayrılabilirdiği durumlarda kullanılabilir.
 - Takım kendi içerisinde problemin alt parçalarını paylaşır.
 - Alt takımlar arasında etkileşim azdır.
- Takım içi ve takımlar arası haberleşme:
 - Resmi yollar: Yazı ile, zamanlı mesajlaşma ile, kurallı ve zamanlanmış toplantılar ile.
 - Gayri resmi yollar: Sözlü iletişim, kişisel etkileşimler, gün içerisinde gerektikçe.



YAZILIM PROJE YÖNETİMİ

PROJE YÖNETİMİNDE ÖLÇÜM

- Proje Ölçümü:
 - Proje ilerlemesi ve proje çıktısı olacak yazılımın ölçülmesidir (incelenecek).
 - Odak: Teknik düzey.
 - Amaç: İç kalite ölçütlerini yüksek tutmak
 - Yöntem: Ölçüm sonuçlarına göre, yazılım geliştirme ekibini iyiye doğru yönlendirmek.
- Süreç Ölçümü:
 - Yazılım geliştirme sürecinin ölçülmesidir.
 - Odak: Yönetimsel düzey.
 - Amaç: Dış kalite ölçütlerine yöneliktir.
 - Yöntem: Sürecin tüm aşamalarında tutulan istatistiklere göre, hem teknik hem de yönetimsel açıdan süreçleri iyileştirmek.

YAZILIM PROJE YÖNETİMİ

PROJE YÖNETİMİNDE ÖLÇÜM

- Kalite ile ilgili etkenler:
 - Kişiler: Çalışanların yetenek ve istekliliği en önemli etkindir.
 - Ürün: Ürünün karmaşıklığı kaliteye olumsuz etki eder. Karmaşıklıkla artan zorluk düzeyi, takımın maneviyatını kırabilir.
 - Teknoloji: Yazılım geliştirme araçları ve donanım bileşenlerinin kalitesi.
 - Kalite: Olgunluk, yeterlilik, etkinlik, eğitimsel belgeler, vb.
 - Çevresel koşullar: Ana etkenlerle de ilişkilendirilebilir.
 - Müşterinin özellikleri: İletişim yeteneği, işbirliği yeteneği, vb.
 - İş koşulları: İş akışı kuralları, şirket kültürü, vb.
 - vb.
- Ölçüt toplama ve gizlilik düzeyleri:
 - Bireysel düzey: Kişiye özel olmalıdır.
 - Takım düzeyi: Takıma özel olmalıdır.
 - Proje düzeyi: Tüm çalışanlara özel olmalıdır.
- Gizlilik düzeyi tartışması:
 - Üstün, astlarının ölçütlerini izleyebilmesi ancak denklemin birbirlerinin ölçütlerini görememesi ise bir başka yaklaşımdır.
 - Amaç insanları utandırmak, övmek veya tehdit etmek olmamalı.

YAZILIM PROJE YÖNETİMİ

PROJE YÖNETİMİNDE ÖLÇÜM

- Süreç Ölçütleri:
 - Hata sayısı: Bulunan hata sayısı.
 - Bakım aşamasından önce ve sonra bulunan hataların sayısı, ayrı öneme sahiptir.
 - Hata giderme etkinliği:
 - $HGE = TÖ / (TÖ + TS)$
 - TÖ: Ürünün müşteriye tesliminden önce bulunan hata sayısı
 - TS: Teslimden sonra (bakım aşamasında) bulunan hata sayısı
 - TÖ ve TS yerine, yazılım geliştirme sürecinin ardışıl adımları olan $A(i)$ ve $A(i+1)$ kullanılabilir. Örneğin:
 - $A(i)$: Çözümlemede bulunan hata sayısı
 - $A(i+1)$: Tasarımda bulunan çözümleme kaynaklı hata sayısı.
 - Doğruluk: Bulunan hata sayısının proje boyutuna oranı.
 - Bakım kolaylığı: Bir hatanın bulunması ile giderilmesi arasında geçen zaman.
 - Güvenlik: Bulunan güvenlik açıkları, açıkların ciddilik düzeyi, vb. gibi veriler üzerinde yapılan ölçümler.
 - Ve diğer dış kalite ölçütleri...



YAZILIM PROJE YÖNETİMİ

PROJE YÖNETİMİNDE ÖLÇÜM

- Süreç ölçütlerinin kullanımı:
 - Ölçütler amaç değil, araç olmalı.
 - Şirketin boyutu ve hatta proje boyutu ile ölçüm yapmaya ayrılan çaba orantılı olmalı.
 - Öncelikleri ve hedefleri belirleyip ona uygun ölçütler kullanılmalı.
 - Amaca yönelik ölçümler yapılmalı.
 - SEI (Software Engineering Institute) ve SPC (Software Productivity Center)'nin önerdiği ölçüt geliştirme yaklaşımları mevcuttur.
 - Ve daha başkaları...
 - Süreç iyileştirme modelleri çeşitli ölçütlerin kullanılmasını da gerektirir.



YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

GENEL BİLGİLER

- Yazılım geliştirme planı aşamasında proje hesaplarını yapabilmek için, daha önce tamamlanmış projelerin kesin hesaplarını örnek almak gerekmektedir.
- Planlama aşamasında yapılan proje tahminleri %100 tutarlı ve güvenli olamamaktadır.
- Buna karşın yine de, eski bilgi ve deneyim sonuçlarına dayanarak, bir tahminde bulunmak gerekmektedir.
- Bu amaçla geliştirilen çeşitli tahmin yöntemlerinden bazıları bu bölümde incelenecektir:
 1. Bilirkişi Takdiri
 2. Delphi Yöntemi
 3. Analiz Yöntemi
 4. İstatistiksel Modeller
 1. COCOMO Modeli
 2. PNR Modeli
 3. COCOMO II Modeli



YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

BİLİRKİŞİ TAKDİRİ

- Bilirkişi takdiri; daha önce tamamlanmış ve maliyeti ile bitirme süresi bilinen projeleri karşılaştırmak yoluyla yapılmaktadır.
- Karşılaştırmada eski ve yeni projenin maliyet kalemleri arasında görülen farklar yüzde olarak belirtilmektedir.
- Bu farklılıklara göre, toptan bir gider ve süre tahmini yapılmaktadır.
- Tutarlı bir tahmin yapılabilmesi için, bilirkişilerin deneyim sahibi olmaları ve iki proje arasındaki farklılıkları gerçekçi olarak saptamaları gerekmektedir.



YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

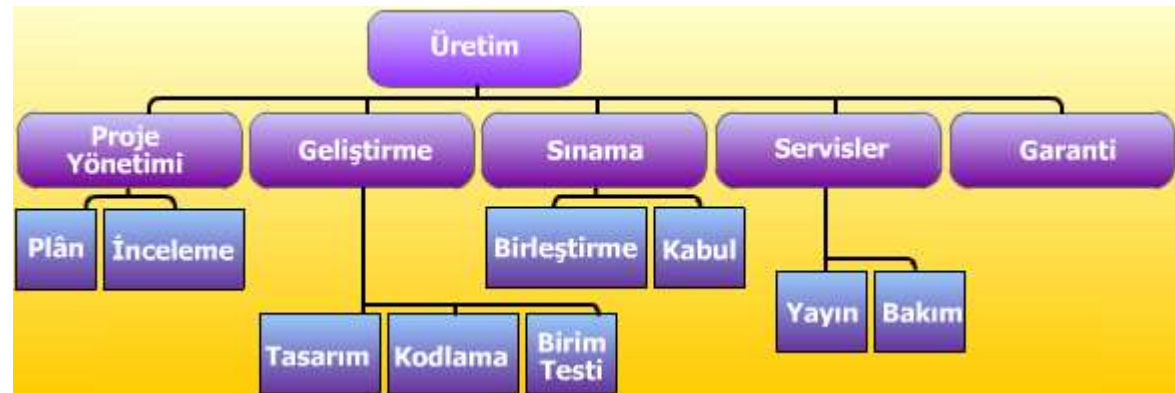
DELPHİ YÖNTEMİ

- Bir koordinatör yönetiminde bilirkişi ekibi tarafından birkaç kez yinelenen yazılı takdir miktarlarına göre ortak bir tahmin yapılmaktadır.
- Bu yöntem aslında sosyal bilimlerde kullanılan bir anket şeklinin proje takdiri için uyarlanmış halidir.
- Yöntem:
 - Koordinatör bilirkişilere "sistemi tanımlama" belgelerini ve birer "tahmin formu" verir.
 - Bilirkişiler birbirinden habersiz, nedenleri ile birlikte tahminlerini yazılı olarak koordinatöre bildirir.
 - Koordinatör ortanca ve aykırı tahmin sonuçlarını bilirkişilere vererek, yeni tahminde bulunmalarını ister.
 - Ortak bir değere yaklaşıncaya kadar bu işlem yinelenmektedir.

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

ANALİZ YÖNTEMİ

- Analiz yöntemi; ürün veya işlemi hiyerarşik olarak öğelerine ayırarak, en alt öğelerden başlayıp yukarı doğru her öğe için gider takdir etmeye dayanır.
- Bu amaçla önce, ürüne ve/veya üretime dayalı analiz kartı düzenlenir.





YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

ANALİZ YÖNTEMİ (devam)

- Hesaplama:
 - Alt basamaktaki öğelerin giderleri takdir edilir ve yukarıya doğru toplamaları alınır.
 - Gider takdirinde, LOC/FP ve (kişi * ay) ölçülerine dayanılmaktadır.
 - Gider kalemleri geçmiş projelerde gerçekleşen giderlerin ortalamaları alınarak veya birden fazla tahminin aşağıdaki ağırlıklı ortalaması ile belirlenir:
 - $E = (a + 4m + b) / 6$
 - E: İlgili gider kalemi için beklenen çaba
 - a: en kötü tahmin, m: en olası tahmin, b: en iyimser tahmin

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

İSTATİSTİKSEL MODELLER

- İstatistik modeller, doğrudan ölçülmeyen bir değeri (y), ölçülebilen bağımsız bir değişken (x) yardımı ile kestirmek amacı ile kurulan, $y = f(x)$ şeklindeki fonksiyonlardır.
- Yazılım proje tahmininde genellikle; $y = a x^b$ biçiminde üslü bir fonksiyon modeli kurulmaktadır. Bu fonksiyonda :
 - Bağımsız değişken (x) adayları:
 - Bin kod satırı (KLOC), fonksiyon noktası (FP), proje süresi (T), iş hacmi (H) değerleri alınmaktadır.
 - Kestirilmesi istenilen olası bağlı değişkenler (y) :
 - İş hacmi (H), proje süresi (I), belge sayfası (B), KLOC olabilmektedir.
 - Fonksiyon katsayıları (a ve b), geçmişteki uygulama sonuçlarına (x ve y) dayanılarak, en küçük kareler yöntemine göre belirlenmektedir.

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

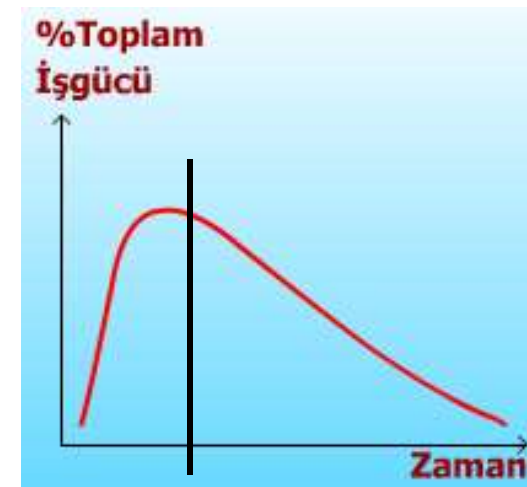
İSTATİSTİKSEL MODELLER (Devam)

- Örnek İstatistik model:
 - Walston-Felix; 4000-467000 satır kodlu olan 60 yazılım projesi sonuçlarına dayanarak aşağıdaki istatistik fonksiyonları kurmuştur:
 - İş hacmi (ayda-kişi/adam-ay): $H = 5.2 \text{ KLOC}^{0.91}$
 - Proje süresi (ay): $T = 4.1 \text{ KLOC}^{0.36}$
veya : $T = 2.47 H^{0.35}$
 - İşgücü talebi (kişi): $I = 0.54 H^{0.06}$
 - Belge hacmi (sayfa): $B = 49 \text{ KLOC}^{1.01}$

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

PNR MODELİ

- Putnam-Norden-Rayleigh modelinde yıllık iş hacmi (yılda işgücü, H); bin kod sayısı (KLOC), teknoloji katsayısı (C_k) ve yıl olarak geliştirme sürecine (t) bağlı şu istatistik fonksiyonu kullanılır:
 - $H = KLOC^3 / (C_k^3 t^4)$
 - C_k teknoloji katsayısı:
 - Düşük teknoloji ortamında 2000,
 - İyi teknoloji ortamında 8000
 - otomatik yazılım araçlarının kullanıldığı ortamda 12000 alınmaktadır.
 - Eski verilere dayanarak da, en küçük kareler yöntemi ile kestirilebilir.
- PNR modeli, işgücü ihtiyacının proje süreci basamaklarına, böylece de sürecin başından bu yana geçen zamana (t) bağlı olduğunu kabul eder.





YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

COCOMO MODELİ

- H. Boehm; COCOMO (COConstructive COst MOdel) adını verdiği bir yazılım proje hesabı tahmin modeli geliştirmiştir.
- Model, hiyerarşik olarak üç basamak halinde uygulanmaktadır:
 - Temel COCOMO
 - Ayrıntıları sonraki yansıdadır.
 - Orta COCOMO
 - Temel modele EAF (Effort Adjustment Factor: Çaba düzeltme çarpanı) serbest değişkeni eklenir
 - Ayrıntıları sonraki yansıdadır.
 - İleri COCOMO
 - Kullanılan yazılım geliştirme sürecinin her aşaması için ayrı ayrı özel fonksiyonlar kullanılır.

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

COCOMO MODELİ (Devam)

- Temel COCOMO
 - Bin kod satırına (KLOC) dayanarak iş hacmi (H) kestirilmekte ve H serbest değişkeninden de proje süresi (T) hesaplanır
 - $H = a \text{ KLOC}^b$
 - $T = c H^d$
 - Boehm; 63 yazılım projesinin sonuç değerlerine dayanak ve Delphi yöntemini uygulayarak, COCOMO modeli için ürün çeşitlerine göre değişen katsayılar saptamıştır:
 - Uygulama programı için :
 - $H = 2.4 \text{ KLOC}^{1.05}$
 - $T = 2.5 H^{0.38}$
 - Yardımcı programlar için:
 - $H = 3.0 \text{ KLOC}^{1.12}$
 - $T = 2.5 H^{0.35}$
 - Sistem yazılımları için :
 - $H = 3.6 \text{ KLOC}^{1.20}$
 - $T = 2.5 H^{0.32}$

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

COCOMO MODELİ (Devam)

- Orta COCOMO
 - Temel modele EAF (Effort Adjustment Factor: Çaba düzeltme çarpanı) serbest değişkeni eklenir.
 - Boehm, Orta COCOMO basamağında da düzeltme faktörünün (EAF), duruma göre 0.90-1.40 arasında alınmasını önerir.
 - Örneğin, bir mikro işlemcide uzaktan haberleşme amacı ile kullanılan 10.000 satır kodlu gömülü (embedded) sistem yazılımında, düzeltme faktörü $EAF = 1.17$ alınarak:
 - İş Hacmi : $H = 3.6 \cdot 10^{1.20} \cdot 1.17 = 66.8$ programcı / ay
 - Proje Süresi $T = 2.5 \cdot 66.8^{0.32} = 9.6$ ay

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

COCOMO II MODELİ

- COCOMO Modelinin çarpanlarının ayrıntılandırılması ve bazı ek değişiklikler ile elde edilmiştir.

$$\text{Çaba} = A \times \text{Büyüklik}^E \times \prod_{i=1}^n \text{ÇabaÇarpan}_i$$

- $A = 2,94$ $n = 17$

$$E = B + 0,01 \sum_{j=1}^5 \text{ÖlçekEtken}_j \quad B = 0,91$$

- Not: Sonraki yansılarda verilecek çarpan ve etken kısaltmalarını ezberleme gerekmemektedir.

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

COCOMO II MODELİ: ÇABA ÇARPANLARI

- Çaba Çarpanları (Ürün kökenli):
 - RELY: Required Software Reliability. Hedeflenen güvenilirlik düzeyi arttıkça daha fazla çaba gerekir.
 - DOCU: Documentation Match to Life-Cycle Needs. Belgelendirme yükü arttıkça çaba da artar.
 - vb.
- Çaba Çarpanları (Platform kökenli: Hedef donanım ve altyapı yazılımı):
 - TIME: Execution Time Constraint. Ürünün çalışmasına ayrılan sürenin donanımın toplam çalışma süresine oranı.
 - vb.
- Çaba Çarpanları (Çalışan kökenli) (Çabaya olumlu etki ederek azaltırlar) :
 - ACAP: Analyst Capability.
 - PCAP: Programmer Capability.
 - vb.
- Çaba Çarpanları (Proje kökenli):
 - TOOL: Use of Software Tools. Olgun bir sürecin izlenmesini destekleyen gereçlerin tümleşikliği arttıkça işler kolaylaşır.
 - vb.

YAZILIM PROJE YÖNETİMİNDE MALİYET TAHMİN YÖNTEMLERİ

COCOMO II MODELİ (ÖLÇEK ETKENLERİ)

- Ölçek etkenleri:
 - PREC: Precedentedness. Geçmiş projelere benzerlik arttıkça yeni proje daha az çaba ile tamamlanabilir (Firma düzeyi deneyim).
 - FLEX: Development Flexibility. İyi belirlenmemiş gereksinimler “esneklik” olarak olumsuz yönde etki eder.
 - vb.

COCOMO II MODELİ (HESAPLAMA/ENİYİLEME)

- Her bir çaba çarpanı ve ölçek etkeninin 1-5 arasında (bazıları için 1-4 arasında) Lickert ölçeğine göre değerlendirmesi yapılır ve buna göre ilgili sayısal değer ilgili formülde kullanılır.
 - Değerler için konu ile ilgilenen arkadaşlar ek araştırma yapabilirler.
- Bazı firmalar daha hassas tahminleme için modeli eniyilemeye (optimizasyon) çalışabilir:
 - A, B katsayıları değiştirilerek daha hassas tahminleme yapabilir.
 - Farklı firmalar kendi çalışma alanlarını dikkate alarak bazı çarpan ve/veya etkenlerin değerlerini değiştirebilirler, silebilirler, yenilerini ekleyebilirler.
 - Yapılan değişikliklerin geçerlenmesi için çeşitli istatistiksel yöntemler kullanılabilir.



YAZILIM PROJE YÖNETİMİ

YAZILIM PROJELERİNİN VAZGEÇİLMEZ ARAÇLARI

- IDE'ler.
- UML modelleme araçları.
 - İki yönlü dönüşüm yeteneğine sahip olması (model ve kod arasında) tercih edilir.
- Sürümlendirme yazılımı (version control systems)
- Sınama yazılımı (testing framework)
 - Bir yapılandırma yazılımı (build system) ile tümleşik olması tercih edilir.
- İş kalemleri izleme yazılımı (work item tracking)
- Tüm araçların IDE tümleşik olması tercih edilir.
 - Java: IBM Rational Application Developer
 - .NET: Microsoft Team System (2020: Azure DevOps)



- Bu yansı ders notlarının sayfa düzeni için boş bırakılmıştır.



YAZILIM MÜHENDİSLİĞİ DERS NOTLARI

Prof.Dr. Mehmet Sıddık Aktaş

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

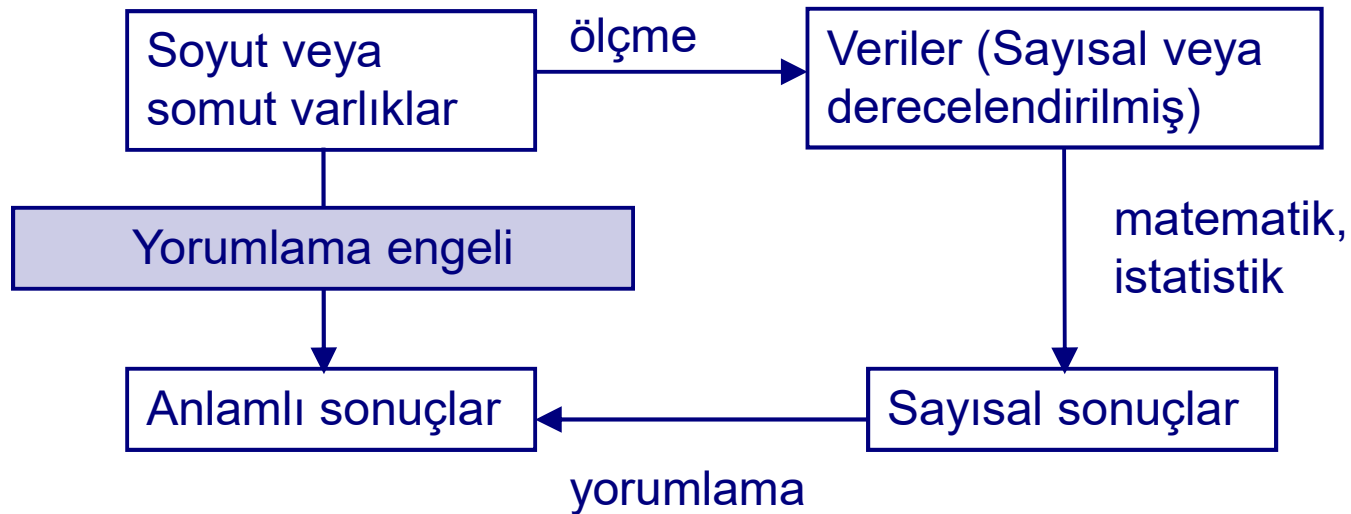
GENEL BİLGİLER

- Ölçme (Measuring): Somut veya soyut bir varlığın sahip olduğu bir özelliğini, sayısal veya derecelendirilmiş bir veri olarak ifade etmek.
 - Benim boyum 163 santimetredir.
 - Hava bugün 22 santigrat derecedir.
 - İlk ara sınav çok zordu.
- Ölçüt (Metric): Varlığın ölçülecek özelliğini ölçme biçimi.
 - Mesafe ölçütleri: Bir labirentteki Öklid ölçütü (Pisagor teoreminden) ve kuş uçuşu ölçütü.
 - Sıcaklık ölçütü: Santigrat ve Fahrenheit
- Ölçüm (Measurement): Belli bir ölçüte göre yapılan ölçme eyleminin sonucu.
- Ölçme/ölçüt/ölçüm karışıklığı
 - İngilizce'de daha da zor
 - Measure –ment ve –ing son eklerini ben özellikle koydum
 - Türkçe'de daha kolay
 - Yine de neyin isim, neyin sıfat, neyin eylem olduğunu karıştırmamalı.
- Neden ölçeriz?
 - Gerçek dünya ile ilgili, işimize yarayacak, anlamlı sonuçlar elde etmek için.

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

GENEL BİLGİLER

- Yorumlama engeli (Intelligence barrier):
 - Ölçmenin sonucu, aradığımız sonuçları elde etmek için doğrudan bir yol sunmayabilir,
 - ya da yapacağımız yorumlama zor olabilir.
 - Örnek: Otostopçunun galaksi rehberi'nde hayatın anlamı: 42!





YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM ÖLÇÜMÜ

- Yazılım ölçümü zordur:
 - Bir başka deyişle, yorumlama engeli yüksektir.
 - Zorluğun nedenleri:
 - Yazılımın karmaşıklığı
 - Ölçütlerin nicel doğası
- Yazılımı neden ölçeriz?
 - Ne kadar iyi bir ürün ortaya çıkardığımızı anlamak
 - Ne kadar iş yapacağımızı kestirmek
 - Böylece ne kadar zaman ve para harcayacağımızı anlamak
 - Ölçülemeyen ilerleme yönetilemez: Proje yönetiminde yazılım ölçütleri kullanılır.



YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM ÖLÇÜMÜ ve ÖLÇÜM BİÇİMLERİ

- İki temel ölçüm biçimi vardır:
 - Doğrudan (Ör. gider, süre, komut satırı, hız, bellek genişliği, hata miktarı)
 - Dolaylı (Ör. işlev, nitelik, karmaşıklık, etkinlik, güvenilirlik, dayanıklılık)
- Ortak amaç; verimlilik, kalite, gider ve belgelemenin hesaplanmasıdır
- Doğrudan ölçülen niceliklerden tanımlanan bazı ölçütler:
 - $LOC = \text{Line of Code}$ (Kod satırı sayısı)
 - $KLOC = 1000 * LOC$ (K:Kilo)
 - $\text{Verimlilik} = KLOC / (\text{kişi} * \text{ay})$
 - $\text{Kalite} = \text{Hata} / KLOC$
 - $\text{Gider} = \text{Toplam gider} / KLOC$
 - $\text{Belgeleme} = \text{Belge sayfası} / KLOC$

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM ÖLÇÜMÜ: FONKSİYON NOKTASI YÖNTEMİ

- Bir dolaylı ölçüm örneği: Fonksiyon noktası ölçümü (FP: Function Point)
 - Tablo 1, 2 ve ardındaki eşitlikler kullanılacaktır:

Tablo 1. Fonksiyon noktası bileşenleri

i	Nicelik	Sayısı (S)	Ağırlık Faktörü (AF)			FPi
			Basit	Orta	Karmaşık	
1	Kullanıcının yazılıma giriş sayısı (user input)		3	4	6	$S * AF$
2	Kullanıcının aldığı çıktı sayısı (output)		4	5	7	
3	Kullanıcının sorgulama sayısı (query)		3	4	6	
4	Kütük sayısı (record)		7	10	15	
5	Dış arabirim sayısı		5	7	10	
					$\sum FPi:$	Hesapla

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM ÖLÇÜMÜ: FONKSİYON NOKTASI YÖNTEMİ

Tablo 2. Yazılım parametreleri

i	Parametre	Fi (0-5 arası)
1	Güvenli yedekleme ve geri yükleme gerekli mi?	
2	İletişim altyapısı gerekli mi?	
3	Dağıtılmış işleme fonksiyonları var mı?	
4	Performans kritik mi?	
5	Sistem yükü fazla mı?	
6	Çevrimiçi veri girişi var mı?	
7	Çok ekranlı hareket girişleri var mı?	
8	Ana dosyalar çevrimiçi güncelleniyor mu?	
9	Giriş, Çıkış ve Sorgular karmaşık mı?	
10	İçsel işlemler karmaşık mı?	
11	Yeniden kullanılabilirlik var mı?	
12	Yükleme tasarıma dahil mi?	
13	Farklı şirketlerde de çalışması söz konusu mu?	
14	Uygulama kullanıcı tarafından kolayca değiştirilebilir mi?	
	ΣF_i :	Hesapla

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM ÖLÇÜMÜ: FONKSİYON NOKTASI YÖNTEMİ

- FP hesabı:
 - Tablo 2.1 her satırı için $FP_i = \text{Sayısı} * \text{Ağırlık Faktörü}$
 - Sayı toplamı: $\sum FP_i$ hesaplanır.
 - Yazılımın basit-orta-karmaşık oluşu tahmin yoluyla kestirilmektedir
 - Tablo 2.2 yazılımın 14 özelliğine göre "karmaşıklık düzeltme değeri" $\sum F_i$ hesaplanır
 - Her özellik 0 (Uygulanamaz/geçersiz) ile 5 (mutlaka gerekli) arasında değerlendirilerek toplama işlemi yapılır
 - $FP = \sum FP_i (0.65 + 0.01 \sum F_i)$
- FP hesabı üzerinden tanımlanan bazı ölçütler:
 - Verimlilik = $FP / (\text{kişi} * \text{ay})$
 - Kalite = $Hata / FP$
 - Gider = $\text{Toplam gider} / FP$
 - Belgeleme = $\text{Belge sayfası} / FP$

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM ÖLÇÜMÜ: FONKSİYON NOKTASI YÖNTEMİ

- Kıyaslama ve eleştiriler:

LOC ÖLÇEĞİ	FP ÖLÇEĞİ
– Programlama diline bağımlı (kısmen)	+ Programlama dilinden bağımsız
– İyi tasarlanmış ama kısa yazılımları yeterince değerlendirememekte	– Sübjektiftir
– İşlemsel olmayan dillerdeki yazılımlara kolayca uyarlanamaz	– Doğrudan fiziksel bir ölçüt değildir
	– Veri toplaması güçtür

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM KALİTE ÖLÇÜTLERİ

- Nicel kalite ölçütleri farklı kişilerce farklı şekillerde öbeklenmekte ve farklı dallara ayrılmaktadır.
- ISO 9126 kalite ölçütleri:
 - İşlevsellik
 - Uygunluk, doğruluk, güvenlik, ...
 - Güvenilirlik
 - Olgunluk, hata bağıışıklığı, ...
 - Kullanılabilirlik
 - ...
 - Verimlilik/Etkinlik
 - ...
 - Bakım kolaylığı
 - ...
 - Taşınabilirlik
 - ...
- McCall ve arkadaşlarının kalite ölçütleri:
 - İşlevsel ölçütler
 - Doğruluk, Güvenilirlik, Bütünlük, Kullanılabilirlik, Verimlilik
 - Değiştirilme ölçütleri
 - Bakım kolaylığı, Esneklik, Sınanabilirlik
 - Taşınma ölçütleri
 - Taşınabilirlik, Yeniden Kullanılabilirlik, Birlikte Çalışabilirlik
- McConnell'a göre kalite ölçütleri:
 - İç kalite ölçütleri
 - Dış kalite ölçütleri

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM KALİTE ÖLÇÜTLERİ

- Dış kalite ölçütleri: Yazılımı kullananları ilgilendiren ölçütler.
 - Doğruluk(Correctness): Yazılımın hatalar içermemesi, gereksinimlerde belirtildiği şekilde çalışması.
 - Etkinlik(Efficiency): Bellek ve işlemci gibi sistem kaynaklarının en az oranda kullanımı.
 - Güvenilirlik(Reliability): Sistemin her koşulda istenildiği gibi çalışması, hatalar arasındaki ortalama zaman aralığının (MTBF) yüksek olması.
 - Güvenlik(Security): İzinsiz ve yetkisiz işlemler mümkün olmamalı.
 - Bütünlük(Integrity): Veriler ve işlemler arasındaki tutarlılığın korunması.
 - Uyarlanabilirlik(Adaptability): Sistemin değişik uygulamalar veya ortamlarda kullanılabilmesi için mümkün olduğunca az değişiklik gerektirmesi.
 - Hassaslık (Accuracy): Sistemin kendisinden beklenen işi mümkün olduğunca iyi yapabilmesi.
 - Sağlamlık(Robustness): Aykırı girişlere veya güç çalışma ortamlarına karşılık sistemin çalışmayı sürdürebilmesi.
 - Kullanılabilirlik(Usability): Yazılım kolay kullanılabilir olmalıdır.
 - ... y17
- Bu ölçütler örtüşebilir, bazı durumlarda birbirinden daha iyi veya daha zor ayrılabilir.

Slayt 183

y17

Interoperability: Ekle. Başka?

yunus, 6/14/2021



YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

YAZILIM KALİTE ÖLÇÜTLERİ

- İç kalite ölçütleri: Yazılımı geliştirenleri ilgilendiren ölçütler.
 - Yeniden kullanılabilirlik(Reusability): Sistemin parçalarının başka sistemlerde kullanılabilmesinin kolaylığı.
 - Bakım kolaylığı (Maintainability): Yazılıma yeni yetenekler eklemenin, yazılımdaki hataları gidermenin veya yazılımın başarımını attırmanın mümkün olduğunca kolay olması.
 - Esneklik(Flexibility): Yazılımın orijinal olarak tasarlandığı uygulama alanının dışında çalışabilmesi için gerekli olan değişikliklerin olduğunca az olması.
 - Taşınabilirlik(Portability): Yazılımın farklı donanım ve işletim sistemleri gibi değişik çalışma ortamlarına kolaylıkla aktarılabilmesi.
 - Okunabilirlik(Readability): Kodun kaynak kodunun incelenmesinin kolay olması.
 - Anlaşılabilirlik(Understandability): Yazılımın sistem, bileşen ve kod düzeylerinde anlaşılabilirliğinin mümkün olduğunca kolay olması. Okunabilirlik sadece kod düzeyinde anlaşılabilirliği sağlar.
 - Sınanabilirlik(Testability): Sistemin istenen gereksinimleri karşılayıp karşılamadığının sınanabilmesinin bileşen ve tüm sistem çapında mümkün olduğunca kolay olması.



YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

ÖLÇME İLKELERİ

- Ölçme eyleminin içermesi gereken adımlar:
 - Tanımlama (Formulation): Ölçütler ölçülecek yazılıma uygun bir şekilde tanımlanır
 - Kullanılan yaklaşım: Yapısal programlama, NYP, vb.
 - Yazılımın türü: Gerçek zamanlı, gömülü, uygulama, vb.
 - Toplama (Collection): Tarif edilen ölçütlerin gerektirdiği verileri elde etme.
 - Hesaplama (Analysis): Ölçütlerin hesaplanması = Ölçümlerin elde edilmesi.
 - Matematiksel araçlar kullanılabilir.
 - Hesaplama mümkün olduğunca otomatik yapılmalıdır.
 - Yorumlama (Interpretation): Elde edilen ölçüm değerlerinden yararlı anlamlar çıkartılması.
 - Geri besleme/Kullanma (Feedback): Çıkartılan sonuçların yazılım ekibine bildirilmesi ve ekibin sonuçları kullanarak yazılımı iyileştirmesi.



YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

ÖLÇME İLKELERİ

- Bir ölçütün sahip olması arzu edilen özellikler:
 - Uygun matematiksel özelliklere sahip olmalı:
 - Anlamlı bir ölçekte olmalı. Ör. 0-1 arası sonuçlar üretmeli. Aksi halde bir değerlendirme aralığı verilmeli.
 - Doğru (veya ters) orantıya sahip olmalı. Sonucun yükselmesi, ölçülen özelliğin iyi bir sonuca doğru ilerlemesi (gerilemesi) anlamına gelmeli.
 - Deneysel olarak doğrulanabilmeli
 - Doğrulanmasının ardından kullanılmalı.



YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

ÖNERİLEN YAZILIM ÖLÇÜTLERİ

- Nesneye yönelik ölçütler:
 - Kaliteli bir yazılıma götüren tasarım ilkelerine yöneliktirler.
 - NYP'de çözümleme ve tasarım arasında kopukluk olmadığı için, aynı ölçütler çözümleme ve kodlama aşamalarında da kullanılabilir.
 - Böylece yazılım ekibi, 'kaliteli bir ürüne giden yolda' iz üstünde olup olmadıklarını anlayabilir.
 - Proje yöneticisi de, başka ölçütlerle birlikte, kestirimlerde bulunabilir.
- Chidamber ve Kemerer'in ölçütleri (CK metrics suite):
 - WMC: Sınıftaki ağırlıklı metot sayısı (Weighted Methods per Class).
 - DIT: Kalıtım ağacının derinliği (Depth of Inheritance Tree).
 - NOC: Alt sınıf sayısı (Number of Children)
 - RFC: Sınıfın yanıt kümesinin eleman sayısı (Response For a Class)
 - Good (RFC=[0,50]), regular (RFC=[51,100]), bad (RFC>100)
 - CBO: Sınıflar arası bağlaşım (Coupling Between Objects)
 - LCOM: Uyum eksikliği (Lack of COhesion in Methods)
 - Good (LCOM=0), regular (LCOM=[1-20]), bad (LCOM > 20).

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

CK ÖLÇÜTLERİ ÖRNEĞİ:

- WMC:
 - C1 sınıfının M1...Mn metotlarının karmaşıklıkları c1..cn.
$$WMC = \sum_{i=1}^n c_i$$
- Eleştiriler:
 - Metot karmaşıklığı neye göre belirlenecek?
 - Belirlemedeki öznellik güçlü yön mü, zayıf yön mü?
- Ölçütün rehberliği:
 - Bir sınıfın karmaşıklığını belirler.
 - Çok sayıda metodu olan sınıf:
 - Çok fazla sorumluluk yüklenmiştir, dağıtılması uygun olabilir.
 - Yüksek uyumun olup olmadığına tekrar bakılmalıdır.
 - Uygulamaya özeldir, yeniden kullanılabilirliği düşüktür.

YAZILIM KALİTESİ VE YAZILIM ÖLÇÜTLERİ

DİĞER KALİTE ÖZELLİKLERİNE YÖNELİK ÖLÇÜTLER

- Bazı bağlaşım ölçütleri
 - COMIAS
 - CBMC
- Sınanabilirlik ölçütleri
 - Halstead ölçütleri (Tartışmalı)
 - Binder'in seçtiği ölçütler
- Bakım kolaylığı ölçütleri
 - IEEE Std. 982.1-1998'de yazılım olgunluk ölçütü

NE YAPILABİLİR?

- Ölçütlerin büyük çoğunluğu mükemmel değildir.
- Yine de bu zayıf noktalar genellikle çok özel durumlarda ortaya çıkar.
- Bu nedenle ölçütler kullanılmalı, ancak tabulaştırılmamalı, sadece (çok da hassas olmayan) bir rehber olarak kullanılmalı.
- Belli bir kalite ölçütüne yönelik olarak, şimdiye dek önerilen ölçütlerden bazıları seçilip, sezgisel olarak bir araya getirildikten sonra piyasada sinanarak iyileştirilebilir.
 - Sezgisel yetenek nasıl bulunacak?
 - Özel sektör gerekli çabaya nasıl ikna edilecek?