**YAZILIM MÜHENDİSLİĞİ DERS NOTLARI – PDF ÖZET**

* Tüm gelişmiş ülkelerin ekonomileri yazılıma bağlıdır.
* Yazılım mühendisliği, profesyonel yazılım geliştirmeye yönelik teoriler, yöntemler ve araçlarla ilgilenir
* Yazılım harcamaları, tüm gelişmiş ülkelerde GSMH’nin önemli bir bölümünü temsil etmektedir.

**Yazılım maliyetleri**

* Yazılım mühendisliği, fiyat-performans odaklı yazılım geliştirme ile ilgilenir.
* Bir bilgisayardaki yazılımın maliyeti genellikle donanım maliyetinden yüksektir.
* Uzun ömürlü sistemler için bakım maliyetleri, geliştirme maliyetlerinin birkaç katı olabilir.

**Yazılım ürünleri**

* **Jenerik Ürünler**
  + - Pazarlanan ve satın almak isteyen herhangi bir müşteriye satılan bağımsız sistemlerdir.
    - Yazılımın özellikleri geliştirici tarafından belirlenir ve ilgili kararlar geliştirici tarafından verilir.
    - Windows OS, Randevu Sistemleri, Grafik Programları vb.
* **Özelleştirilmiş Ürünler**
  + - Belirli bir müşteri tarafından kendi ihtiyaçlarını karşılamak için yaptırılan yazılımlardır.
    - Yazılımın özellikleri yazılımı yaptıran kişi tarafından belirlenir ve kararlar müşteriye aittir.
    - Trafik İzleme Sistemi, Gömülü Kontrol Sistemleri, Hava Trafik Kontrol Yazılımı vb.

**İyi Bir Yazılımın Temel Nitelikleri**

* **Sürdürülebilirlik:** Yazılım, müşterinin değişen ihtiyaçlarını karşılayacak şekilde olmalıdır. Değişen bir iş ortamının kaçınılmaz bir gereğidir.
* **Güvenilebilirlik ve Güvenlik:** Güvenilir yazılım, sistem arızası durumunda fiziksel veya ekonomik hasara neden olmamalıdır. Kötü niyetli kullanıcılar sisteme erişememeli veya sisteme zarar verememelidir.
* **Verimlilik:** Yazılım, bellek ve işlemci gibi sistem kaynaklarını boşa harcamamalıdır. Yanıt verme, işlem süresi, bellek kullanımı vb. şeyleri içerir.
* **Kabul edilebilirlik:** Yazılım, tasarlandığı kullanıcı türü için kabul edilebilir olmalıdır. Anlaşılır, kullanılabilir ve diğer sistemlere uyumlu olması gerektiği anlamına gelir.

**Çoğu Yazılımı Etkileyen Genel Sorunlar**

* **Heterojenlik:** Sistemlerin farklı cihaz türlerini de içeren ağlar arasında dağıtılmış sistemler olarak çalışması gerekmektedir.
* **İş ve Sosyal Değişim:** Yeni teknolojilerle birlikte iş ve toplum hızla değişiyor. Mevcut yazılımların değişmesi ve yeni yazılımları hızla geliştirebilmeleri gerekiyor.
* **Güvenlik ve İtimat:** Yazılım hayatımızla iç içe olduğundan o yazılıma güvenebilmemiz çok önemlidir.

**Uygulama Türleri**

* **Bağımsız Uygulamalar:** Yerel bilgisayarlarda çalışan uygulama sistemleridir. Gerekli tüm işlevleri içerirler ve bir ağa bağlı olmaları gerekmez.
* **Etkileşimli İşlem Tabanlı Uygulamalar:** Uzak bir bilgisayarda çalışan ve kullanıcılar tarafından kendi bilgisayarlarında veya terminallerinden erişilen uygulamalardır. E-ticaret vb. gibi web uygulamalarıdır.
* **Gömülü Kontrol Sistemleri:** Bunlar, donanım cihazlarını kontrol eden ve yöneten yazılım kontrol sistemleridir.
* **Toplu İşleme Sistemleri:** Verileri büyük gruplar halinde işlemek için tasarlanmış iş sistemleridir. Çok sayıda bağımsız girdiyi işlerler.
* **Eğlence Sistemleri:** Öncelikle kişisel kullanım için olan ve kullanıcıyı eğlendirmeyi amaçlayan sistemlerdir.
* **Modelleme ve Simülasyon Sistemleri:** Bilim insanları ve mühendisler tarafından birçok etkileşimli nesneyi içeren fiziksel süreçleri veya durumları modellemek için geliştirilen sistemlerdir.
* **Veri Toplama Sistemleri:** Bir dizi sensör kullanarak ortamlardan veri toplayan ve bu verileri işlemek üzere başka sistemlere gönderen sistemlerdir.
* **Sistem Sistemleri:** Bunlar, bir dizi başka yazılım sistemlerinden oluşan sistemlerdir.

**Yazılım Mühendisliğinin Temelleri**

* Geliştirme Süreci
* Güvenilirlik ve Performans
* Özellik ve Gereksinimlerin Anlaşılması ve Yönetilmesi
* Uygun durumlarda yeni yazılım yazmak yerine mevcut yazılımın yeniden kullanılması

**Mesleki Sorumluluk Konuları**

* **Gizlilik**
* **Yetkinlik**
* **Fikri Mülkiyet Hakları**
* **Bilgisayarın Kötüye Kullanımı**

|  |  |
| --- | --- |
| Yazılım nedir? | Bilgisayar programları ve ilgili belgeler. |
| İyi bir yazılım özellikleri nelerdir? | İşlevsellik, performans, sürdürülebilirlik, güvenilebilirlik, kullanılabilirlik |
| Yazılım mühendisliği nedir? | Yazılımın üretiminin tüm yönleriyle ilgilenen mühendislik disiplinidir. |
| Temel yazılım mühendisliği faaliyetleri nelerdir? | Yazılım özellikleri, yazılım geliştirme, yazılım doğrulama ve yazılım evrimi |
| Yazılım mühendisliği ve bilgisayar bilimi arasındaki farklar nelerdir? | Bilgisayar bilimi teori ve temellere odaklanır. Yazılım mühendisliği kullanışlı yazılım geliştirme ve pratikleriyle ilgilenir. |
| Yazılım mühendisliği ve sistem mühendisliği arasındaki farklar nelerdir? | Sistem mühendisliği, bilgisayar tabanlı sistem geliştirmenin tüm yönleriyle ilgilenir. Yazılım mühendisliği, bu genel sürecin bir parçasıdır. |

|  |  |
| --- | --- |
| Yazılım mühendisliğinin karşılaştığı temel zorluklar nelerdir? | Artan çeşitlilikle başa çıkmak, daha kısa teslimat süreleri talep etmek, güvenilir yazılım geliştirmek |
| Yazılım mühendisliğinin maliyetleri nelerdir? | Yazılım maliyetlerinin kabaca %60’ı geliştirme, %40’ı test maliyetidir. Özel yazılımlarda geliştirme maliyeti genellikle test maliyetini aşar. |
| En iyi yazılım mühendisliği teknikleri ve yöntemleri nelerdir? | Bir yöntemin veya tekniğin diğerinden daha iyi olduğunu söylemek doğru değildir. |
| Web, yazılım mühendisliğinde ne gibi farklara sebep oldu? | Programlama dillerinde ve yazılımın yeniden kullanımında önemli gelişmelere yol açmıştır. |

**Yazılım Süreci Faaliyetleri**

* **Spesifikasyon:** Sistemin ne yapması gerektiğini tanımlamak.
* **Tasarım ve Uygulama:** Sistemin organizasyonunu tamamlamak ve sistemi uygulamak
* **Doğrulama:** Müşterinin istediğini yapıp yapmadığını kontrol etmek.
* **Değişim:** Değişen müşteri ihtiyaçlarına yanıt olarak sistemi değiştirmek.

**Yazılım Süreçlerinin Modelleri**

* **Şelale Modeli:** Plan odaklı modeldir. Spesifikasyon ve geliştirmenin ayrı ve farklı aşamaları vardır.
  + - * Projenin farklı aşamalara esnek olmayan şekilde bölümlenmesi, değişen müşteri gereksinimlerine yanıt vermeyi zorlaştırır. Bu nedenle bu model yalnızca gereksinimler iyi anlaşıldığında ve tasarım sürecinde değişiklikler oldukça sınırlı olduğunda uygundur.
      * Şelale modelinin temel dezavantajı sürece başladıktan sonra süreci değiştirmenin zor olmasıdır. Prensip olarak, bir sonraki aşamaya geçmeden önce aşamanın tamamlanması gerekir.
      * Şelale modelinin plan odaklı yapısı çalışmayı koordine etmeye yardımcı olur.
      * Şelale modelinde ayrı tanımlanmış aşamalar vardır:
        + Gereksinim analizi ve tanımı
        + Sistem ve yazılım tasarımı
        + Uygulama ve birim testi
        + Entegrasyon ve sistem testi
        + Operasyon ve bakım
* **Artımlı Geliştirme:** Spesifikasyon, geliştirme ve doğrulama serpiştirilmiştir. Plan odaklı veya çevik olabilir.
  + - * Sistem yapısı, yeni artışlar eklendikçe bozulma eğilimindedir ve süreç görünmez. (Zararı)
      * Değişen müşteri gereksinimlerini karşılamanın maliyeti azaltılır. (Faydası)
      * Müşteri geri bildirimi almak daha kolaydır. (Faydası)
      * Yazılımın müşteriye daha hızlı teslimi ve dağıtımı mümkündür. (Faydası)
* **Yeniden Kullanım Odaklı Yazılım Mühendisliği:** Sistem, mevcut bileşenlerden bir araya getirilmiştir. Plan odaklı veya çevik olabilir.

**Gereksinim mühendisliği,** bir yazılım spesifikasyonu geliştirme sürecidir.

**Gereksinim Mühendisliği Süreci**

* + **Fizibilite Çalışması:** Sistemi kurmak teknik ve mali olarak uygun mu?
  + **Gereksinimlerin Ortaya Çıkarılması ve Analizi:** Sistem paydaşları sistemden ne ister veya ne bekler?
  + **Gereksinim Özellikleri:** Gereksinimlerin ayrıntılı olarak tanımlanması
  + **Gereksinimlerin Doğrulanması:** Gereksinimlerin geçerliliğini kontrol etmek

**Tasarım Faaliyetleri**

* **Mimari Tasarım:** Sistemin genel yapısını, ana bileşenleri ve bunların ilişkilerini belirlediğimiz süreçtir.
* **Arayüz Tasarımı:** Sistem bileşenleri arasındaki arayüzleri tanımladığımız süreçtir.
* **Bileşen Tasarımı:** Her sistem bileşenini alıp nasıl çalışacağını tasarladığımız süreçtir.
* **Veritabanı Tasarımı:** Veri yapılarını ve bunların veritabanında nasıl temsil edileceğini tasarladığımız süreçtir.

**Test Aşamaları**

* **Geliştirme veya Bileşen Testi:** Bireysel bileşenler bağımsız olarak test edilir.
* **Sistem Testi:** Sistemin bir bütün olarak test edilmesidir. Ortaya çıkan özelliklerin test edilmesi özellikle önemlidir.
* **Kabul Testleri:** Sistemin, müşterinin ihtiyacını karşılayıp karşılamadığını kontrol etmek için müşteri verileriyle test edilmesidir.

**Prototip,** konseptleri göstermek ve tasarım seçeneklerini denemek için kullanılan bir sistemin ilk sürümüdür.

**Yazılım Prototipi Kullanımı**

* Gereksinimlerin ortaya çıkarılması ve doğrulanmasına yardımcı olmak için
* Tasarım sürecinde seçenekleri keşfetmek ve bir UI tasarımı geliştirmek için
* Test sürecinde arka arkaya testleri çalıştırmak için

**Prototip Oluşturmanın Faydaları**

* İyileştirilmiş sistem kullanılabilirliği
* Kullanıcıların gerçek ihtiyaçlarına daha yakın bir eşleşme
* İyileştirilmiş tasarım kalitesi
* İyileştirilmiş bakım kolaylığı
* Azaltılmış geliştirme çabası

**Artımlı Dağıtımın Avantajları**

* Sistem işlevselliği daha erken kullanılabilir hale gelir.
* Erken artışlar, sonraki artışlar için gereksinimleri ortaya çıkarmaya yardımcı olacak bir prototip görevi görür.
* Genel proje başarısızlığı riski daha düşüktür.
* En yüksek öncelikli sistem hizmetleri en çok testi alma eğilimindedir.

**Artımlı Dağıtımın Sorunları**

* İhtiyaçlar, bir artış uygulanana kadar ayrıntılı olarak tanımlanmadığından tüm artışların ihtiyaç duyduğu ortak tesisleri belirlemek zor olabilir.

**Boehm’in Spiral Modeli**

* Süreç, geriye dönük izleme içeren bir faaliyetler dizisi yerine bir sarmal olarak temsil edilir.
* Spiraldeki her döngü, süreçteki bir aşamayı temsil eder.
* Spesifikasyon veya tasarım gibi sabit aşama yoktur. Gerekli olana bağlı olarak spiraldeki döngüler seçilir.
* Süreç boyunca riskler açıkça değerlendirilir ve çözülür.

**Spiral Model Sektörler**

* **Hedef Belirleme:** Aşama için özel hedefler belirlenir.
* **Risk Değerlendirmesi ve Azaltma:** Riskler değerlendirilir ve temel riskleri azaltmak için faaliyetler başlatılır.
* **Geliştirme ve Doğrulama:** Sistem için genel modellerden herhangi biri olabilecek bir geliştirme modeli seçilir.
* **Planlama:** Proje gözden geçirilir ve spiralin bir sonraki aşaması planlanır.

**RUP (Rasyonel Birleştirilmiş Süreç) Aşamaları**

* **Başlangıç:** Sistem için iş senaryosu oluşturun.
* **Detaylandırma:** Sorun alanı ve sistem mimarisi hakkında bir anlayış geliştirin.
* **İnşaat:** Sistem tasarımı, programlama ve test
* **Geçiş:** Sistemi işletim ortamında devreye alın.

**RUP’taki Statik İş Akışları**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| İş Modelleme | Gereksinimler | Analiz ve Tasarım | Uygulama | Test Yapmak |
| Dağıtım | Proje Yönetimi | Çevre | Yapılandırma ve değişiklik yönetimi | |

**Çevik Yöntemlerin İlkeleri**

* **Müşteri Katılımı:** Müşteriler geliştirme süreci boyunca yakından ilgilenmelidir.
* **Artımlı Teslimat:** Yazılım, müşteri ile aşamalı olarak geliştirilir.
* **Süreç Yapmayan İnsanlar:** Geliştirme ekibinin becerileri tanınmalı ve kullanılmalıdır.
* **Değişikliği Benimse:** Sistem gereksinimlerinin değişmesini bekleyin ve sistemi bu değişiklikleri karşılayacak şekilde tasarlayın.
* **Basitliği Koruyun:** Hem geliştirilmekte olan yazılımda hem de geliştirme sürecinde basitliğe odaklanın.

**Çevik Yöntemlerle İlgili Sorunlar**

* Müşterilerin ilgisini korumak zor olabilir.
* Ekip üyeleri uygun olmayabilir.
* Birden fazla paydaşın olduğu yerlerde değişikliklere öncelik vermek zor olabilir.
* Sadeliği korumak ekstra çalışma gerektirir.
* Sözleşmeler, bir sorun olabilir.

**Plan Odaklı Geliştirme ve Çevik Geliştirme**

* **Plan Odaklı Geliştirme**
  + Bu yaklaşım, önceden planlanan bu aşamaların her birinde üretilecek çıktılarla ayrı geliştirme aşamalarına dayanmaktadır.
  + Şelale modeli olması gerekmez. Plan odaklı, kademeleri geliştirme mümkündür.
  + Yineleme, etkinlikler içerisinde gerçekleşir.
* **Çevik Geliştirme**
  + Spesifikasyon, tasarım, uygulama ve testler birbiri ardına bırakılır ve geliştirme sürecinden elde edilen çıktılara, yazılım geliştirme sürecinde bir müzakere süreci ile karar verilir.

**Ekstrem Programlama (XP)**

Belki de en çok bilinen ve en çok kullanılan çevik yöntemdir.   
 Ekstrem programlama yinelemeli geliştirmeye “ekstrem” bir yaklaşım getirir.

* Yeni sürümler günde birkaç kez oluşturulabilir.
* Ürün sürümleri her iki haftada bir müşterilere teslim edilir.
* Tüm testler her derleme için çalıştırılmalıdır ve yapı yalnızca testler başarıyla çalıştırılırsa kabul edilir.

**Çiftler Programı**

* XP’de programcılar kod geliştirmek için birlikte oturarak çiftler halinde çalışırlar.
* Bu, ortak kod sahipliği geliştirmeye yardımcı olur ve bilgiyi ekibe yayar.
* Her bir kod satırına 1’den fazla kişi tarafından bakıldığı için gayri resmi bir inceleme süreci olarak hizmet eder.
* Tüm ekip bundan faydalanabileceği için yeniden düzenleme yapmayı teşvik eder.
* Çiftler dinamik olarak oluşturulur ve böylece tüm ekip üyeleri geliştirme sürecinde birbirleriyle çalışırlar.
  + - **Avantajları**
      * + Kod ile ilgili sorunlardan şahıslar sorumlu tutulmaz. Ekibin bir sorumluluğu vardır.
        + Gayri resmi bir inceleme olarak hizmet eder.
        + Bir yazılım geliştirme süreci olan yeniden düzenlemeyi desteklemeye yardımcı olur.

**Scrum**

* Scrum yaklaşımı genel bir çevik yöntemdir ancak odak noktası, belirli çevik uygulamalardan ziyade yinelemeli geliştirmeyi yönetmektir.
* Scrum’da üç aşama vardır:
  + Taslak planlama aşaması
  + Sprint döngüsü
  + Proje kapanış aşaması

**Avantajları**

* Ürün, yönetilebilir ve anlaşılır parçalara bölünmüştür.
* Karasız gereksinimler ilerlemeyi engellemez.
* Tüm ekip her şeyi görebilir ve sonuç olarak ekip iletişimi geliştirilir.
* Müşteriler, ürünün nasıl çalıştığına dair geri bildirim alır.
* Müşteriler ve geliştiriciler arasında güven tesis edilir.

**Sprint Döngüsü**

* Sprintler sabit uzunluktadır, normalde 2-4 haftadır.
* XP’de sistemin bir sürümünün geliştirilmesine karşılık gelirler.
* Seçim aşaması, müşteriyle birlikte çalışan tüm proje ekibidir.
* Tüm iletişimler sözde ‘Scrum ustası’ aracılığıyla gerçekleştirilir.
* Scrum ustasının rolü, geliştirme ekibini dış dikkat dağıtıcı unsurlardan korumaktır.
* Sprint sonunda yapılan işler gözden geçirilir ve paydaşlara sunulur. Bir sonraki sprint başlar.

**Büyük Sistem Geliştirme**

* Büyük sistemler; genellikle her sistemi ayrı ekiplerin geliştirdiği, ayrı ve iletişim halindeki sistemlerin koleksiyonlarıdır.
* Bu ekipler sıklıkla farklı yerlerde ve farklı zaman diliminde çalışıyorlar.

**Gereksinim Mühendisliği**

* Müşterinin bir sistemden ihtiyaç duyduğu hizmetleri, çalıştığı ve geliştirildiği sürecin kısıtlamalarını kurma süreci.
* Gereksinimlerin kendisi, gereksinim mühendisliği sürecinde üretilen sistem hizmetleri ve kısıtlamaların tanımlarıdır.
  + Gereksinimler yoruma açık olmalıdır.
  + Ayrıntılı olarak tanımlanmalıdır.
* Akıl Sağlığı Hasta Yönetim Sistemi (AS-HYS) örnektir.

**İhtiyaç Türleri**

* **Kullanıcı gereksinimleri**
  + Müşteriler için yazılmıştır. Doğal dildeki ifadeler, sistemin sağladığı hizmetlerin şemaları ve operasyonel kısıtlamaları.
* **Sistem gereksinimleri**
  + Sistemin işlevlerinin, hizmetlerinin ve operasyonel kısıtlamaların ayrıntılı açıklamalarını ortaya koyan yapılandırılmış bir belge. Neyin uygulanması gerektiğini tanımlar ve bu nedenle müşteri ve yüklenici arasındaki bir sözleşmenin parçası olabilir.

**İşlevsel Gereksinimler**

* Sistemin sunması gereken hizmetlerin beyanları, sistemin belirli girdilere nasıl tepki vermesi gerektiği ve sistemin belirli durumlarda nasıl davranması gerektiği.
* Sistemin ne yapmaması gerektiğini de belirtebilir.

**İşlevsel Olmayan Gereksinimler**

* Zamanlama kısıtlamaları, geliştirme sürecindeki kısıtlamalar, standartlar vb gibi sistem tarafından sunulan hizmetler veya işlevler üzerindeki kısıtlar
* Genellikle tek tek özellikler veya hizmetler yerine bütün olarak sisteme uygulanır.

**Etki Alanı Gereksinimleri**

* Operasyon alanından sistemdeki kısıtlamalar

**\*\*\*** Prensip olarak, gereksinimler hem eksiksiz hem de tutarlı olmalıdır.

\*\*\* Sistem işlevlerinin açıklamalarında herhangi bir çatışma veya çelişki olmamalıdır.

**İşlevsel Olmayan Gereksinimler için Metrikler**

* Hız
* Boyut
* Kullanım Kolaylığı
* Güvenilirlik
* Sağlamlık
* Taşınabilirlik

**Yazılım gereksinimleri belgesi**, sistem geliştiricilerinin neye ihtiyaç duyduğuna dair resmi açıklamadır. Bir tasarım belgesi değildir. Sistemin nasıl yapması gerektiğinden ziyade ne yapması gerektiğini ayarlamalıdır.

**Gereksinim Belgesinin Yapısı**

* Önsöz
* Giriş
* Sözlük
* Kullanıcı Gereksinimleri Tanımı
* Sistem Mimarisi
* Sistem Gereksinimleri Belirtimi
* Sistem Modelleri
* Sistem Gelişimi
* Ekler
* Dizin

**Doğal Dil ile İlgili Sorunlar**

* Net değil, netlik yok, netliğin olmaması : Belgenin okunmasını zorlaştırmadan hassaslık zordur.
* Gereksinim karmaşası : İşlevsel ve işlevsel olmayan gereksinimler karışık olma eğilimindedir.
* Gereksinimlerin birleşmesi : Birkaç farklı gereksinim birlikte ifade edilebilir.

**Gereksinim Mühendislik Süreçleri**

Bu süreçler, uygulama alanına, ilgili kişilere ve gereksinimleri geliştiren kuruluşa bağlı olarak büyük ölçüde değişir.

Bununla birlikte, tüm süreçler için ortak olan birkaç genel faaliyet vardır:

* Gereksinimlerin ortaya çıkarılması
* Gereksinimlerin analizi
* Gereksinimlerin doğrulanması
* İhtiyaç yönetimi

**Gereksinim Analizi Sorunları**

* Paydaşlar gerçekten ne istediklerini bilmiyorlar.
* Paydaşlar gereksinimleri kendi terimleriyle ifade ederler.
* Farklı paydaşların birbiriyle çelişen gereksinimleri olabilir.
* Organizasyonel ve politik faktörler, sistem gereksinimlerini etkileyebilir.
* Analiz sürecinde ihtiyaçlar değişir. Yeni paydaşlar ortaya çıkabilir ve iş ortamı değişebilir.

**Senaryolar**

* Bir sistemin nasıl kullanılacağına dair gerçek hayat örnekleridir.
* Dahil etmeliler:
  + Başlangıç durumunun bir açıklaması
  + Normal olay akışının bir açıklaması
  + Neyin yanlış gidebileceğinin bir açıklaması
  + Diğer eş zamanlı faaliyetler hakkında bilgi
  + Senaryo bittiğinde durumun açıklaması

**\*\*\*Etnografi,** mevcut süreçleri anlamak için etkilidir ancak bir sisteme eklenmesi gereken yeni özellikleri belirleyemez.

**Gereksinim Kontrolü**

* **Geçerlilik :** Sistem, müşterinin ihtiyaçlarını en iyi şekilde karşılayan işlevleri sağlıyor mu?
* **Tutarlılık :** Herhangi bir gereksinim çatışması var mı?
* **Tamlık :** Müşterinin gerektirdiği tüm işlemler dahil mi?
* **Gerçekçilik :** Mevcut bütçe, teknoloji ve zaman çerçevesi göz önüne alındığında gereksinimler uygulanabilir mi?
* **Doğrulanabilirlik :** Gereksinimler kontrol edilebilir mi?

**Sistem Modelleme**

Her modelin, o sistemin farklı bir görünümü veya perspektifini sunduğu bir sistemin soyut modellerini geliştirme sürecidir.

**Sistem Perspektifleri (Süreçleri)**

* **Harici Perspektif :** Yazdığımız işletim sistemi farklı sistemlerle etkileşimde bulunabilir, wifi kullanabilir, fare ile etkileşimde bulunabilir 🡪 Başka bileşenlerle donanımlarla etkileşimde bulunabilir
* **Yapısal Perspektif :** Mimariyi kullanarak modeller geliştirilir. Veritabanı, class'lar nerede olacak vb. modellenmelidir.
* **Davranışsal Perspektif :** Yazdığımız fonksiyonlar ne yapacak? Bağlan dediğinde ya da kaydet dediğinde nasıl davranacak ne yapacak ??

**UML diyagramları** içerisinde

**activity** yani faaliyet diyagramları vardır

**use case diagrams** --> Aktörlerimiz neler? Kişiler, fonksiyonlar aktör olabilir. Kim neyi kullanacak kim ne yapacak? Bunları gösterir. Tasarıma yardımcı olur.

**sequence diagrams** --> Sistemimizle aktörlerimiz ve fonksiyonlarımız arasındaki etkileşimi yani içerdeki bütün bileşenlerin birbirleriyle etkileşimlerini gösterir

**class diagrams** --> Sınıf içerisinde hangi değişkenler kullanılıyor, metotlar kullanılıyor bunlar burada yer alır ve gösterilir. Diyelim ki uzaktan iş aldınız ve yaptığınız tasarımı Amerika'daki bir ortağınız nasıl anlayacak? Bu yüzden her şey gösterilir açıklanır

**State diagrams** --> Hangi durumlarda ne olacak? Hangi şartı sağlamış ise sonrasında ne olacak?

**\*\*Context Models\*\***

Benim hareket alanım ne? Yapacağım sistem nereyi kapsayacak? Sınırlarım neler? Nereyle etkileşimde bulunacağım? Context model bunları gösterir.

Özetle sistemimizin sınırlarını belirlemek, şeklini ortay koymaktır.

Sistemin içinde ve dışında olacak olaylara yine burada karar veriyoruz.

**\*\*Interaction Models\*\***

En önemli unsurlardan bir tanesidir. Bileşenler birbirleriyle haberleşerek çalışır. Sadece kullanıcılardan değil farklı sistemlerden aldığımız etkileşimler de vardır.

Use case diagrams kullanılır

**\*\*Structural Models\*\***

Yapı mimariyle ilgilidir. Sistemin mimarisini gösterecek. Nasıl bir kurgu olacağını gösterecek.

Mimariyi oluşturma dinamik de olabilir statik de olabilir. Sistem mimarisini oluşturmaya yarıyor.

Class diyagramları kullanılır